# a2ps, version 4.8.1

ASCII to PostScript converter, and pretty-printer
Edition 4.8.1, 16 April 1997

**Akim Demaille**
**Miguel Santana**

Copyright © 1988, 89, 90, 91, 92, 93 Miguel Santana

Copyright © 1995, 96, 97 Akim Demaille, Miguel Santana

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

# 1 Introduction

This document describes a2ps version 4.8.1. The latest versions may be found on (decreasing order of service quality):

1. 'http://www-stud.enst.fr/~demaille/a2ps.html'.
2. 'http://www-inf.enst.fr/~demaille/a2ps.html'.
3. 'http://www.enst.fr/~demaille/a2ps.html'.

Please, send us emailcards :). Whatever the comment is, or if you like a2ps, write to us at: ⟨Miguel.Santana@st.com⟩ and ⟨demaille@inf.enst.fr⟩. If you need to bug report, send them to ⟨demaille@inf.enst.fr⟩.

## 1.1 Description

a2ps formats named files for printing in a PostScript printer; if no file is given, a2ps reads from the standard input. The output may be sent to the printer or to the standard output or saved into a file.

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.

Compared to the previous versions, a2ps version 4.8.1 provides:

1. totally different options, to be closer to that of GNU enscript (see Section 3.1 [Options], page 6);
2. various configuration files (see Section 6.1 [Configuration files], page 26)
3. some powerful meta-sequences to define the headers the way you want (see Section 3.2 [Meta sequences], page 13);
4. a rather strong support of Latin 2, 3, 4, 5 and 6 encodings, thanks to Ogonkify(see section "Overview" in *Ogonkify manual*), written by Juliusz Chroboczek.
5. fully customizable output style: fonts (in the limit of the 13 standard fonts), background and foreground colors, line numbering style etc. (see Section 6.2 [Designing prologues], page 28).
6. and of course, the ability to pretty-print sources written in quite a few various languages (see Chapter 4 [Pretty printing], page 17).

## 1.2 Vocabulary

This section settles some terms that are going to be used through out this document.

*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.

*Virtual page*
Area on a physical page in which a2ps draws the content of a file.



*Page* A single face of a sheet. There may be several virtual pages on a physical page.

*Sheet* The physical support of the printing: it may support one or two pages, depending on your printing options.

*Face* A virtual style given to some text. For instance, *Keyword*, *Comment* are faces.

*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.

# 2 User Guide

This chapter is devoted to people who don't know yet a2ps: we try to give a soft and smooth introduction to the most useful features. For a reference manual, see Chapter 3 [Invocation], page 6.

## 2.1 Purpose

a2ps is a program that takes an ASCII file (i.e., human readable), and makes a PostScript file out of it. *PostScript* is a page description language, in other words, it is programming language, in which 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.

## 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:

```
a2ps doc.txt
```

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 does as if you had given the option '`-2`', standing for two virtual pages.

Say now that 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:

```
a2ps foo.h bar.c -4ofoobar.ps
```

The option '`-4`' tells a2ps to make four virtual pages: two rows by two columns. The option '`-o`' (which is the short version of '`--output=foobar.ps`') specifies the output file. Since the option '`-4`' does not have arguments, it can be grouped behind the same '`-`' as the one for '`-o`'. The other way round would fail, producing a file '`foobar.ps4`'.

Note too that the options may be 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 ever it knows the (programming) language in which your file is written, it will try to make it nice looking 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:

```
a2ps -A4 foo.h bar.c --prologue=gray -P lw
```

Here '`-A`' is used to specify that you Allow a2ps to put several files on the same sheet, '`-P lw`' means to print on the printer named '`lw`', and finally, the long option '`--prologue`' asked to use one of the alternative printing styles of a2ps. There are other available prologues (see option '`--prologue`' in Section 3.1.5 [Input], page 9), and you can even design yours (see Section 6.2 [Designing prologues], page 28).



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:

```
gargantua ~ $ a2ps --list-options
General information on a2ps:
  version             = 4.7.20
  library path        = /inf/soft/infthes/demaille/share/a2ps
  verbosity level     = 0
Sheets:
  default paper         = A4
  layout per page       = 2 x 1 (portrait)
  borders               = yes
  compact mode          = yes
  inside margin         = 0
More stuff deleted here
```

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 parametrizable 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 3.1.2 [Sheets], page 7, option '`--media`').

**The encoding**
Make sure that the *encoding* a2ps uses is the same as the standard alphabet in your country (See Section 3.1.5 [Input], page 9, option '`--encoding`').

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

## 2.4 Localizing

a2ps provides some Native Language Support, that is speaking your mother tongue. It uses two special features wrt non-English languages: on the one hand, the interface between you and it, and on the other hand, the format of the dates and times.

To use these features, you may set your environment variable LANG to the standard code representing your language:

| | |
|---|---|
| ca | Catalan |
| cs | Czesh |
| es | Spanish |
| fr | French |
| it | Italian |
| pl | Polish |

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 which `awk` and such.

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

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

Note that the '`make install`' *must* have been done in order to have NLS support.

## 2.5 Interfacing with other programs

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

### 2.5.1 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 -2gEmail %s`'. The option '`-g`' is given because there are some special escapes that `a2ps` understands, and replaces with the correct character.

### 2.5.2 pine

Jan Chrillesen ⟨`jan@chrillesen.dk`⟩ suggests us how to use `a2ps` with the Pine mail-reader. Add the following to '`.pinerc`' (ofcourse you can put it in '`pine.conf`' as well):

```
# Your printer selection
printer=a2ps -Email -R -1 -P Your-printer

# Special print command if it isn't one of the standard printers
personal-print-command=a2ps -Email -R -1 -P Your-printer
```



# 3 Invocation

Calling `a2ps` is fairly simple:

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

## 3.1 Command line options

`a2ps` uses GNU `getopt`, hence:

- the order between options and files does not matter: '`a2ps -4m main.c`' and '`a2ps main.c -4m`' are identical.
- options may be grouped together: '`a2ps -4mg main.c - Pprinter`'
- when there are no ambiguities, long options can be abbreviated, e.g., '`--re`' will be understood as '`--report`',
- '`--`' ends the options. Anything behind '`--`' is considered to be a file: '`a2ps -- -2`' prints the file '`-2`'.

Here after *bool* stands for boolean. It is considered as true (i.e. setting the option on), if *bool* 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.

### 3.1.1 Global behavior

'`-q`'
'`--quiet`'
'`--silent`'
: be really quiet

'`-v`'
'`--verbose`'
: tell what we are doing

'`-V`'
'`--version`'
: print version and exit successfully

'`-=`*shortcut*'
'`--user-option=`*shortcut*'
: use the *shortcut* defined by the user. See Section 6.1 [Configuration files], page 26, item '`UserOption:`'.

'`--report[=`*language*`]`'
: generate PreScript report on all/*language* style. This file must be later processed by `a2ps` using the *PreScript* style.

'`--guess`'
: `a2ps` won't print anything, but will act as `file` does: it returns for each file the language in which `a2ps` believes the file is written. This can be very useful on broken systems to understand why a file is written with a bad style sheet (see Section 4.2 [Automatic style], page 17).

'`-h`'
'`--help`' print a short help, and exit successfully.

'`--list-options`'
Give a extensive report on `a2ps` configuration and installation.

'`--list-features`'
Known medias, encodings, languages, prologues, printers and user option shortcuts are reported.

### 3.1.2 Sheets

'`-M`'
'`--media=`*media*'
use output media *media*. Currently *media* may be '`A3`', '`A4`', '`A5`', '`B4`', '`B5`', '`Letter`', '`Legal`', '`Tabloid`', '`Ledger`', '`Statement`', '`Executive`', '`Folio`', '`Quarto`', '`10x14`'. '`A4dj`', '`Letterdj`' are also defined for Desk Jet owners, since that printer needs bigger margins.

'`-r`'
'`--landscape`'
print in landscape mode

'`-R`'
'`--portrait`'
print in portrait mode

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

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

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

'`-1`' alias for '`--columns=1 --rows=1 --portrait --columns-per-page=80 --major=row`'

'`-2`' alias for '`--columns=1 --rows=1 --landscape --columns-per-page=80 --major=row`'

'`-3`' alias for '`--columns=3 --rows=1 --landscape --major=row`'

'`-4`' alias for '`--columns=2 --rows=2 --portrait --columns-per-page=80 --major=row`'

'`-5`' alias for '`--columns=5 --rows=1 --landscape --major=row`'

'`-6`' alias for '`--columns=3 --rows=2 --landscape --columns-per-page=80 --major=row`'

'`-7`' alias for '`--columns=7 --rows=1 --landscape --major=row`'

'`-8`' alias for '`--columns=4 --rows=2 --landscape --columns-per-page=80 --major=row`'

'`-9`' alias for '`--columns=3 --rows=3 --portrait --columns-per-page=80 --major=row`'



'`-j`'
'`--borders=`*bool*'
print borders around virtual pages.

'`-A`'
'`--compact=`*bool*'
Compact mode for a sequence of files. This option allows the printing of more than one file in the same physical page. This option makes sense only when the number of virtual pages is greater than 1. Otherwise, the beginning of each file will be printed in a new sheet.

'`--margin[=`*num*`]`'
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.

### 3.1.3 Pages

'`--line-numbers=`*number*'
precede each *number* line with its line number.

'`-C`' Alias for '`--line-numbers=5`'.

'`-f`'
'`--fontsize=`*size*'
scale font to *size* for body text. *size* is a float number. To change the base font, change the current prolog (see Section 6.2 [Designing prologues], page 28.

'`-l`*num*'
'`--columns-per-page=`*num*'
Set the number of columns per page. *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*'
'`--lines-per-page=`*num*'
Set the lines per virtual page for printing. The font size is automatically scaled up to fill in the whole page. This is useful for printing preformatted documents which have a fixed number of lines per page. The scaling of the font size will be suppressed if this option is used with option '`--font`'. 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 and a warning message is printed on the terminal.

'`-m`'
'`--catman`'
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.

'`-T`*num*'
'`--tabsize=`*num*'
set tabulator size to *num*. This option is ignored if `--interpret=no` is given.

'`--non-printable-format=`*format*'
: specify how non-printable chars are printed. *format* can be '`caret`' ('`^A`', '`M-^B`' etc.), or '`space`' (a space is written instead of the non-printable character).

### 3.1.4 Headers

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 meta sequences. See Section 3.2 [Meta sequences], page 13 for details.

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

'`-B`'
'`--no-header`'
: no page headers at all.

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

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

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

### 3.1.5 Input

'`-c`'
'`--truncate-lines=`*bool*'
: 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`'
'`--interpret=`*bool*'
: interpret tab, bs and ff chars

'`-X`*name*'
'`--encoding=`*name*'
: use input encoding *name*. Currently `a2ps` supports:

  '`ascii`' Regular ASCII code.



  '`latin1`' Which real ISO name is ISO8859-1.

  '`latin2 (ISO8859-2)`'
  '`latin3 (ISO8859-3)`'
  '`latin4 (ISO8859-4)`'
  '`latin5 (ISO8859-9)`'
  '`latin6 (ISO8859-10)`'
  : The support for these encodings is provided thanks to *Ogonkify* (see section "Overview" in *Ogonkify manual*). Ogonkify is a tool that lets you extend the font support for most encodings, i.e., for most non-western-Europe alphabets. If you think the support is not good enough, you may help the extension of Ogonkify: See section "Adding new characters" in *Ogonkify manual*, for details.

  '`hp8`' The 8 bits Roman encoding for HP.

  '`mac`' For the Macintosh encoding. The support is not sufficient, and a lot of character may be missing at the end of the job. This is planed to be fixed.

  '`ibmpc`' For the regular PC encoding. Not all the characters will be printed.

  '`pcg`' For the PC Graphic encoding.

  '`cp1250`' MicroSoft's CP-1250 encoding (aka CeP).

'`-t` *filename*'
'`--title=`*filename*'
: Give the name *filename* to the files read trough the standard input.

'`--prologue=`*prologue*'
: Use *prologue* as the PostScript prologue for `a2ps`. *prologue* must be in a file named '*prologue*`.pro`', which be in a directory of your library path (see Section 3.5 [Library files], page 16). Currently the available prologues are:

  '`black+white`'
  : the regular presentation

  '`gray`' same presentation, but with gray shades for comments etc.

  '`gray2`' same as gray, but light declaration (such as function calls etc.) are mapped on the same shape as the strong declarations (such as function declaration etc.).

  '`color`' still the same idea, but with colors.

'`--print-anyway=`*bool*'
: force binary printing. By default, a 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.

### 3.1.6 Output

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

'`-n`*num*'
'`--copies=`*num*'
: print *num* copies of each page

'`-D`'
'`--setpagedevice[=`*key*`:`*value*`]`'
: Pass a page device definition to the generated PostScript output. If no *value* is given, *key* is removed from the definitions.

  For example, command

  ```
  a2ps -DDuplex:true report.pre
  ```

  prints file '`report.pre`' in duplex (two side) mode.

  Page device operators are implementation dependent but they are standardized. See Section 3.3 [Page device options], page 15 for details.

'`-S`'
'`--statusdict=`*key*`[:`*value*`]`'
'`--statusdict=`*key*`[::`*value*`]`'
: Pass a statusdict definition to the generated PostScript output. `statusdict` operators and variables are implementation dependent; see printer's documentation for details. See Section 3.4 [Statusdict options], page 16 for details.

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

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

  ```
  a2ps -Ssetpapertray: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 -Spapertray::1 quicksort.c
  ```

  produces

  ```
  /papertray 1 def
  ```

  in the PostScript.

'`-k`'
'`--page-prefeed`'
: 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`'
'`--no-page-prefeed`'
: disable page prefeeding. See item '`DefaultPrinter:`' in Section 6.1 [Configuration files], page 26 and results of option '`--list-options`' to see the bindings between printer names and commands.

'`-P` *name*'
'`--printer=`*name*'
: send output to printer *name*. See item '`Printer:`' and '`Unknown printer:`' in Section 6.1 [Configuration files], page 26 and results of option '`--list-options`' to see the bindings between printer names and commands.

'`-d`'

'`-s`*num*'
'`--side=`*num*'
: number of sheet sides. *num* is of course 1 or 2.

### 3.1.7 Pretty printing

'`-g`'
'`--graphic-symbols=`*bool*'
: set symbols graphical conversion

'`-E`*language*'
'`--language=`*language*'
: use the *language* pretty print style sheet. See Section 4.3 [Known languages], page 18, for the available style sheets.

'`-K`'
'`--automatic-style=`*bool*'
: `a2ps` will try to guess in what language is written your files, and use the corresponding style sheet.

'`--strip-level=`*num*'
: Depending on the value of *num*:

  0. everything is printed;
  1. regular comments are not printed
  2. strong coments are not printed
  3. every comment is canceled.

  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.

## 3.2 Meta sequences

Headers, footers, titles and the water mark are strings in which some meta sequences are later replaced by their actual values. In general '`%`' are related to general information, while '`$`' meta sequences are related to the current file.

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 meta sequences (e.g., '`$L`', '`$l`').

Currently `a2ps` support the following meta sequences:

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

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

'`$?#`*cond*`#`*if_true*`#`*if_false*`#`'
this may be used for conditional assignment. The separator (presented here as '`#`') may be any character. *if_true* and *if_false* may be defined exactly the same way as regular headers, included meta sequences and the '`$?`' construct.

*cond* may be:

'`l`' true if printing in landscape mode,

'`v`' true if printing on the back side of the sheet (verso).

'`1`'
'`2`'
'`3`' true if tag 1, 2 or 3 is not empty. See item '`$t1`' for explanation.

'`$(`*var*`)`' value of the environment variable *var*

'`%a`' the localized equivalent for '`Printed by` *User Name*'

'`%A`' the localized equivalent for '`Printed by` *User Name* `from` *Host Name*'

'`%a{`*username*`}`'
the localized equivalent for '`Printed by` *username*'

'`%A{`*username*`@`*hostname*`}`'
the localized equivalent for '`Printed by` *username* `from` *hostname*'.

These two are provided in the case `a2ps` would be used by the print service: since neither of the user name nor the host name can be known at the time the files reach `a2ps`, these options should be used.

'`%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`' 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`' current date in '`dd.mm.yyyy`' format.

'`$F`' file modification date in '`dd.mm.yyyy`' format.

'`%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 user login name

'`$n`' input file name without the directory part

'`%N`' the user's pw_gecos field up to the first '`,`' character

'`$N`' the full input file name

'`%p`' current page number

'`$p`' page number for this file

'`%P`' total number of pages printed

'`$P`' number of pages 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`' current sheet number

'`$s`' sheet number for the current file

'`%S`' total number of sheets

'`$S`' number of sheet 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`'

'`%*`' current time in 24-hour format with seconds '`hh:mm:ss`'

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

'`$v`' the sequence number of the current input file

'`$V`' the total number of files

'`%W`' current date in '`mm/dd/yy`' format

'`$W`' file modification date in '`mm/dd/yy`' format

All format directives can also be given in format

*escape width directive*

where *width* specifies the width of the column to which the escape is printed. For example, escape '`$5%`' will expand to something like '`    12`'. If the width is negative, the value will be printed left-justified.

## 3.3 Page device options

Page device is a PostScript level 2 feature that offers an uniform interface to control 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 can not. Refer to the documentation of your printer for supported options.

Here are some usable page device options which can be selected with the '`-D`' option ('`--setpagedevice`'). For a complete listing, see *PostScript Language Reference Manual*: section 4.11 Device Setup.

`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

## 3.4 Statusdict options

The `statusdict` is a special storage entity in PostScript (called a *dictionnary*), in which some variables and operators determine the behavior of the printer. As for page device, those variables are device dependent: 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.

## 3.5 Library files

`a2ps` uses several files: its PostScript prologue, encoding related files, and such. `a2ps` looks for these files in the following path:

```
$HOME/.a2ps:/usr/local/share/a2ps
```

You may change this default path with the environment variable `A2PS_LIBRARY`.

If you plan to define yourself some header files for `a2ps` (option '`--prologue`'), they should be in one of those directory, with a '`.pro`' suffix.

## 3.6 Exit status

The following exit values are returned:

'`0`' `a2ps` terminated normally.

'`1`' an error occured.

'`2`' bad argument was given.

'`3`' unknown language was given.

# 4 Pretty printing

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

- basic (option '-E') in which wath you print is what you wrote.
- with graphic substitution (option '-g'). In this mode, 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.

## 4.1 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*, i.e., sequences of characters which are opened by a given marker, and ended the same way. For instance in C, the comments are opened by '/*', and ended by '*/'.

It is for the same reason that you can't expect a2ps to highlight the function definitions in C.

## 4.2 Automatic style

The heuristic a2ps uses to find out in which language is written a file is fairly simple:

1. a2ps tries the user defined filename matching rules (see Section 6.1.2 [Your guess rules], page 26). Upon success, the corresponding language style is used.
2. a2ps tries to find out thanks to the file suffix and its own abbreviation databases (see Section 4.4.1 [Name and abbreviations], page 19).
3. It asks file(3) its opinion on the file. If the answer looks like a language, a2ps trusts file, *but* for the C, since file has the bad habit to consider that quite everything is written in C.
4. Plain style will be used.

Two things are to retain from this:

1. a2ps won't guess anything for a file given through the standard input.
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 (see Section 6.1.2 [Your guess rules], page 26).



## 4.3 Known languages

a2ps version 4.8.1 supports the following languages:

Ada

C

C++

caml (ml)

Claire (cl)
: More information on this language can be found on
'http://www.ens.fr/~laburthe/claire.html'

coq-vernacular (coq)

Common-lisp (lsp)

Eiffel (e)

Fortran (f)
: This style was developed by Denis Girou ⟨Denis.Girou@idris.fr⟩.

java

lace (ace)

mailfolder (mail)
: Support for mail folders is rather good (see Section 2.5 [Interfacing], page 5). This style also suits news files.

Modula-3 (m3)

68000

o2c

Oberon

Octave/MATLAB
: This was written by Craig P. Earls ⟨cpearls@mit.edu⟩.

Pascal (pas)

perl (pl)

PostScript (ps)

PreScript (pre)

prolog (pro)

promela (pml)
: This style was written thanks to Jean-Philippe Cottin ⟨cottin@inf.enst.fr⟩.

python (py)

Sather (sa)

Scheme (scm)

`SDL-88 (sdl)`
: This style was written thanks to Jean-Philippe Cottin ⟨`cottin@inf.enst.fr`⟩.
`SDL` may be *the* language for which '`--strip-level=2`' is the most useful: it cancels the graphical information left by graphic editors. Only the pure specification is then printed.

`sh`

`SQL family`
: These styles were written by Pierre Mareschal ⟨`pmaresch@be.oracle.com`⟩.
1. SQL92 (`sql92`)
2. Oracle SQL (`sql`)
3. Oracle PL/SQL (`pks`)
4. Oracle SQL-PL/SQL-SQL*Plus (`oracle`)
5. Oracle Init.ora parameter file (`initora`)

`tcl`
`tk`
: Since everything, in those languages, are strings, what prints `a2ps` is not always what you would like (see Section 4.1 [Syntactic limits], page 17).

`Unity`
: This style was written thanks to Jean-Philippe Cottin ⟨`cottin@inf.enst.fr`⟩. Note that the graphic conversion of the symbols ('`-g`') is just perfect for this language.

`Verilog`
: This style was written by Edward Arthur ⟨`eda@ultranet.com`⟩.

`VHDL`
: This style was written by Thomas Parmelan ⟨`Thomas.Parmelan@efrei.fr`⟩.

`zsh`

## 4.4 Definition of the style sheets

`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.

### 4.4.1 Name and abbreviations

Every style has a unique name (reported with '`--list-features`'). It has also a list of abbreviations (which should include usual suffix for the language, such as '`e`' for the Eiffel language).

Any of the name or abbreviations can be used with option '`-E`'. When automatic pretty-printing is enabled, first `a2ps` calls `file` to see whether the language is recognizable. Otherwise, the suffix of the file is compared with every name and abbreviation. On failure, plain style is used.

### 4.4.2 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.



### 4.4.3 Keywords and regular symbols

A keyword if recognized, is just written in a special font. A symbol, when recognized, is replaced by the corresponding character in the symbol font. To be recognized, both need to start by a character in the first alphabet and to be immediately followed by a character which does not belong to the second alphabet.

### 4.4.4 Special symbols

They need not to be preceded and followed by characters belonging to the alphabet. For instance in '`caml`', '`not`' is a regular symbol: it represents boolean negation; but in neither '`nota`' or '`notnot`', the three first letters represent '`not`'. On the contrary '`<>`' always means 'not equal', even in '`a<>b`'.

Both regular and special symbols are transformed only if option '`--graphic-symbols`' was given.

### 4.4.5 Sequences

*Sequence*s are string is a string between two *marker*s. A marker is a fixed string. Typical examples are strings (with usually '"' as opening and closing markers), comments etc. Three fonts are used: one for the initial marker, one for the core of the sequence, and a last one for the final maker.

### 4.4.6 Escapes and verbatims

*Escape*s are immediately copied when found in a sequence. Their main use is to avoid a sequence from being terminated too soon, e.g., the string '"\""' is legal in C, hence it is necessary to specify in `a2ps` that '\\' and '\"' have to be written has such when in a sequence: they are escapes.

*Verbatim*s are immediately copied wherever met. It has been implemented for ada in which ''''' is the constant character ''''. `a2ps` must not understand it as opening of a sequence (first quote), closing of the sequence (second quote), opening of a sequence (third quote). Nor is it possible to specify that '''' is an escape...

### 4.4.7 Case sensitivity

If the style is case insensitive, the case does not matter for the keyword, symbol and sequence recognition. Other categories are not concerned by case sensitivity.

## 4.5 Writing new style sheets

To be able to define new style sheets one needs `GNU m4`. Also, never update '`styles.c`' by hand, but its precursor '`styles.c.in`'.

There is a couple of things to know about the process. But in anyway, please use the already defined languages as examples, and see the prologue of '`styles.c.in`' where many `m4` macros are defined to ease the implementation. Read also '`styles.h`' where information is given on the `struct` used for the style sheets.

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.

For the particular case of the special symbols, note that you may have to protect the symbols which are prefixed by them. For instance if in `Pascal` you define '`=`' to be a special symbol, you have to protect the '`=`' in '`:=`'. Otherwise `a2ps` will see a '`:`', written such as, but will transform the '`=`' in the symbol you specified. Hence declare '`a2_not_symbol(:=)`'.

The opening and closing markers of a sequence may use '`^`' and '`$`' escapes: the former to specify beginning of line, and the latter for end of line. Both may loose their special meaning thanks to '`\`', which is **always** discarded when in first or/and last position.

E.g., the marker '`^---cut-here---$`' will only match lines equal to '`---cut-here---`'. On the other hand '`\^money$\`', matches '`^money$`' anywhere in the line, and '`\\^money$\\`' matches '`\^money$\`' everywhere in the line.

It is also possible to specify several closing markers, just by putting them one after the other, for instance (taken for the `ada` style sheet):

```
a2_sequence(procedure , keyword_strong, label, ' is', keyword_strong),
a2_sequence(procedure , keyword_strong, label, ' (', courier),
a2_sequence(procedure , keyword_strong, label, '(', courier),
a2_sequence(procedure , keyword_strong, label, '$', courier),
a2_sequence(procedure , keyword_strong, label, ';', courier),
```

Please note that:

1. the order between sequences does matter. For instance in java, '`/**`' introduces strong comments, and '`/*`' comments. '`/**`' *must* be declared before '`/*`', or it will be hidden.
2. closing alternatives must be grouped together (no sequence with a different opener must occur while defining alternative closers).
3. closing markers may have different faces (like '` is`' and '`(`' above).
4. '`$`' is used instead of a hard coded '`\n`'. Some encodings do not use '`\n`' as end-of-line, hence a style using '`\n`' would not be portable.

Please, respect the alphabetical order in which the languages are introduced. Don't forget to update '`STYLE`' in '`styles.h`', and to add your language in the macro `a2_styles` at the bottom of '`styles.c.in`'.



# 5 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 a LaTeX-like syntax.

The main advantages for using `PreScript` are:

- it is fairly simple,
- it is LaTeX compatible (that is, correctly written `PreScript` files do compile with LaTeX)
- `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.).

## 5.1 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, though it is perfectly legal in LaTeX, the following line will be badly interpreted by `a2ps`:

```
\textbf{Problems using \textit{PreScript} instead of \LaTeX}
```

The correct way to write this in `PreScript` is

```
\textbf{Problems using} \textbi{PreScript} \textbf{instead of} \LaTeX.
```

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

## 5.2 `PreScript` commands

### 5.2.1 Structure 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. These instructions are said to be hard-coded. They may be removed in a future definition of PreScript. The following instructions are prefered.

'`\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 (i.e. font Times).

'`\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 Section 5.2.2 [Printed symbols], page 23).

'`\header{`*text*`}`'
'`\footer{`*text*`}`'
: Use *text* as header (footer) for the current page. At run time, to override the defined headers and footers by these dynamic headers, the option '`--dynamic-header`' must be set. If several headers or footers are defined on the same page, the last one is taken into account.

### 5.2.2 Printed symbols

There is a big menagerie of symbols. To get a list, it is suggested to print `a2ps`'s report about `PreScript` using the command line:

```
a2ps --report=PreScript | a2ps -EPreScript
```

The reader should be aware that 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 `PreScript`, it should be written

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

for LaTeX. Since in `PreScript` every '`$`' is discarded (unless quoted by a '`\`'), the second form is also admitted.

### 5.2.3 Special characters

'`$`' is ignored in `PreScript` 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.

### 5.2.4 Quotation

There is a general mechanism to avoid special sequences from being interpreted: the commands '`\verb+`*quoted string*`+`', where '`+`' can be any symbol in '`+`', '`!`', '`|`', '`#`', '`=`'.

## 5.3 Differences with LaTeX

### 5.3.1 LaTeX commands ignored in `PreScript`

Some LaTeX commands have no equivalent in `PreScript`, but are simply skipped at compilation time (if immediately followed by an end-of-line):

- '`\begin{document}`'
- '`\end{document}`'
- '`\begin{itemize}`'
- '`\end{itemize}`'
- '`\begin{enumerate}`'
- '`\end{enumerate}`'
- '`\begin{description}`'
- '`\end{description}`'

Everything between '`%%PreScript:skip`' and '`%%PreScript:piks`' will be ignored in `PreScript`, so that there can be inserted command definitions for LaTeX exclusively.

### 5.3.2 Commands available in PreScript but not in LaTeX

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

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

There is no way in `PreScript` 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 `PreScript` and LaTeX.

'`header`' and '`\footer`', are not understood by LaTeX.

Some symbols are available in `PreScript` but not in LaTeX:

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

A good example of a LaTeX preamble to get LaTeX to compile `PreScript` files is given by the report of `a2ps`.

## 5.4 Examples

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

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



# 6 Customizing a2ps

## 6.1 Configuration files

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

1. the system configuration file (usually '`/etc/a2ps.cfg`')
2. the user's home configuration file ('`$HOME/.a2ps/a2psrc`')
3. the local file ('`./.a2psrc`')

In these 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.

### 6.1.1 Your default options

'`Options:` *options*`+`'
: Give `a2ps` a list of command line options. *options*`+` is any sequence of regular command line options (see Section 3.1 [Options], page 6).

  The quoting mechanism is a bit painful: you need to quote the whole *options*`+`. For instance

  ```
  Options: --right-title='Page $p'
  ```

  is wrong. Write

  ```
  Options: '--right-title=Page $p'
  ```

### 6.1.2 Your guess rules

'`Pattern:` *pattern language*'
: Specify that any file which name matches *pattern* (usual Unix file name patterns) should be treated with the style *language*. For instance, `Promela` programmers use the suffix '`pr`' for their files, though `a2ps` hard codes '`pr`' to `SDL`. To override this rule, use:

  ```
  Pattern: *.pr promela
  ```

### 6.1.3 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.

'`DefaultPrinter:` *destination*'
: Tell `a2ps` where to send the PostScript it produced. See next item for information on *destination*. The *DefaultPrinter* is used when option '`-d`' is given.

'`Printer`: *name destination*'
'`UnknownPrinter`: *destination*'
Tell `a2ps` where to send the PostScript it produced. Both entries are related to the '`-P` *printer*' option, the first one being triggered when *printer* is equal to *name*, and the second when *name* as no `Printer`: entry.

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

'`|` *command*'
in which case the PostScript is piped in *command*

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

In each case, the argument given to option '`-P`' is substituted to '`%s`', and '`%`' to '`%%`' in *destination*.

For instance

```
DefaultPrinter: | lp -d dominique
UnknownPrinter: | lp -d %s
Printer: foo > foo.ps
Printer: wc | wc
Printer: lw | lp -d printer-with-a-rather-big-name

# Here we address a printer which is not PostScript
Printer: deskj | gs -q -sDEVICE=ljet3d -sOutputFile=- - \
| lpr -P flr2 -h -l
```

MS-DOS users, and non-PostScript printer owners should take advantage in getting good configuration of these entries.

### 6.1.4 Your shortcuts

'`UserOption`: *shortcut Option+*'
define a *shortcut*, i.e., when `a2ps` is called with '`--user-option=`*shortcut*', then consider *Option+*.

### 6.1.5 Your PostScript magic number

`a2ps` produces full DSC compliant PostScript which is something that only PS wizards may understand[1]. Adobe said

Thou shalt start you PostScript DSC compliant 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[2]. For instance:

`OutputFirstLine`: *magic-number*

where *magic-number* is '`%!PS-Adobe-2.0`', or just '`%!`'.

[1] Basically it means the the PostScript that `a2ps` generates is **really** modular. In other words, any kind of post-processing may be applied to the files it produced.
[2] In other words, there are no PostScript printers that don't understand these files.



## 6.2 Designing 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 "meanning" in the text. `a2ps` let's you change the way the faces look.

### 6.2.1 a2ps faces

`a2ps` uses 7 faces:

'`Plain (p)`'
This corresponds to the text body.

'`Keyword (k)`'
'`Keyword_strong (K)`'
These are related to the keywords that may appear in a text.

'`Comment (c)`'
'`Comment_strong (C)`'
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 (l)`'
'`Label_strong (L)`'
These are used when a point of extreme importance, or a sectioning point, is met. Typically, functions declarations etc.

'`String (str)`'
Used mainly for string and character literals.

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

### 6.2.2 Definition of the faces

There are three things that define a face:

*Its font* There is a uniform font naming scheme. They all start by '`f`', as in *font* :), then a letter for the name of the family of the font ('`c`' for Courier, '`t`' for Times, and '`h`' for Helvetica, and finally an optional letter for the declination of in the font family ('`b`' for bold, '`i`' for italic, and '`e`' (as in emphasize) for bold-italic).

There are two steps in specifying a face's font:

1. tell `a2ps` the font you use. This is achieved by a comment line:

   `%Face:` *face real-font-name*

2. tell the PostScript the font you want to use. This achieved calling `setfont`, preceded by the short font name:

   *short-font-name* `setfont`

*Its background color*

There are two cases:

1. You want a background color, then give the *RGB* (see Section 6.2.3 [Colors in PostScript], page 29) 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
   ```

3. Its foreground color As `BG`, call `FG` with an *RGB* ratio:

   ```
   0 0.5 0 FG
   ```

### 6.2.3 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 upper 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.

### 6.2.4 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-features`'), to drop it in one of `a2ps` directories (say '`$HOME/.a2psrc`', 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 Hawaïan sea green (you are definitely a fine art *amateur*).

Then you need to look for '`k`' and '`K`':



```
%Face: Keyword Courier
/k {
  false BG
  0 0 0.9 FG
  fc setfont
  Show
} bind def

%Face: Keyword_strong Courier-Bold
/K {
  false BG
  0 0 0.8 FG
  fcb setfont
  Show
} bind def
```

and turn it into:

```
%Face: Keyword Helvetica
/k {
  0.2 1 0.2 true BG
  1 0.2 1 FG
  fh setfont
  Show
} bind def

%Face: Keyword_strong Times_boldItalic
/K {
  0.4 0.2 0 true BG
  0.5 1 1 FG
  fte setfont
  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 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.



# 7 Various

In this chapter is collected what had no place in the rest of this document.

## 7.1 Non PostScript printers

Here are some tips we have on how to use a non PostScript printer. If somebody feels like writing a more precise documentation, he really is welcome.

### 7.1.1 From Clayton Weaver

(for a bj200ex on os2 from linux) This is a shell script called as a filter from an ":if=/usr/local/bin/filtername" entry in /etc/printcap on linux:

```
#!/bin/bash

gs -q -sDevice=bj200 -sOutputfile=- -r360 -dNOPAUSE -dSAFER -
```

The device is a gs device name, input is the ending "-" (stdin), output is stdout, -q is "no messages". Can be used with

```
lpr -Pprintername -Ffilter_path_and_name infile.ps
```

or just '`lpr -Pprintcap_printername infile.ps`' if the filter is specified in the printcap '`:if=`' entry for that "printcap_printername". Either way, lpr on linux runs it in a subshell, so you don't want "exec" before the gs command (that kills the subshell and leaves gs running with no way to communicate its stdout back to lpr).

You can print directly from gs, I've done it on OS/2 by specifying "lpt1" as the output device, but printing to a remote printer I need lpr and lpd to handle transmitting the job, so this way works better for me.

Regards, Clayton Weaver ⟨`cgweav@eskimo.com`⟩ (Seattle)

### 7.1.2 From Pierre Juillot

To print toto.ps on my deskjet, I hit:

```
> gsdj toto.ps
```

where '`gsdj`' is

```
#!/bin/sh
#                              05-06-1996
# to print a PostScript file using ghostscript onthe
# deskjet 500 (dj) or the printer specified via  the
# environment variable LPOPTS
# using ghostscript
# exec gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=djet500 \
exec gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=deskjet \
     -sOutputFile="|lp -onb -ddj -or $LPOPTS " $* quit.ps
```

### 7.1.3 These tips in a2ps

To use these examples, first see Section 6.1 [Configuration files], page 26, items '`DefaultPrinter:`', '`UnknwonPrinter:`' and '`Printer:`'. The translations would be in one of your configuration files, say '`$HOME/.a2ps/a2psrc`':

```
Printer: lpt1 | gs -q -sDevice=bj200 -sOutputfile=- -r360 -dNOPAUSE \
        -dSAFER > lpt1
UnknownPrinter:  | gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=deskjet \
     -sOutputFile="|lp -onb -ddj -or %s " quit.ps
```

## 7.2 Frequently asked questions

### 7.2.1 Printing goes beyond the frame of the paper

You are currently printing with a bad media, for instance using A4 paper within `a2ps`, while your printer uses Letter paper. See Section 3.1.1 [Global behavior], page 6, option '`--media`' for more.

### 7.2.2 Helping on the translation

Please, help us internationalizing `a2ps`. If you want to do a profound translation work, contact us. Having a look at the files '`po/*.po`' is also very informative.

If you are ready to help just a bit, it is simple too: on `a2ps`'s web page, in the internationalization section, there are some pointers to files. Filling some blanks in thoses files is definitely a great help.

### 7.2.3 Why not having used `yacc` and such

There are several reasons why we decided not to use grammars to parse the files.

First it would have made the design of the style sheets much more tricky, and today `a2ps` would know only 4 or 5 languages.

Second, it limits the number of persons who could build a style sheet.

Third, mixing several parsers within one program is not easy. Moreover, its size would have ten times what it is.

Fourth, we did not feel the need for such a powerful tool: handling the keywords and the sequences is just what the users expect.

And last but not least, using a parser requires that the sources are syntactic bug free, which is a strong 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.



### 7.2.4 Man pages

There are no man pages because now the documentation is written in Texinfo. What you are reading comes from this initial Texinfo file.

Nevertheless, man pages may be back from Hell the day a `texi2roff` filter appears.

### 7.2.5 Can I change the base font

Well, no. We worked on the 13 standard fonts, that are available in any printer. You could easily make the PostScript prologues download another fonts, but the problem is that the fonts characteristics (width of the characters) are hard coded in `a2ps`. This is because it is easier, faster, and because `a2ps` is defintitely not Word.

If you need to deal with several various fonts, then GNU `enscript` is exactly the tool you need.

## 7.3 What next?

### 7.3.1 EPSF

`a2ps` should be able to generate EPS files in a short while.

### 7.3.2 Spelling checking, syntax fixing.

When printing a human file (only English may be supported), spelling should be checked. When pretty-printing, syntax errors should be underlined.

No just kidding :). Do it yourself...

# Appendix A Genesis

## A.1 History

The initial version was a shell program written by ⟨evan@csli⟩ (Evan Kirshenbaum). It was very slow and contained many bugs.

A new version was written in C by Miguel Santana for improve speed execution 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 known 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 quite a lot other things.

Since the very beginning of a2ps, many people have brought help:

- Oscar Nierstrasz ⟨<oscar@iam.unibe.ch⟩
- Tim Clark ⟨T.Clark@warwick.ac.uk⟩
- ⟨tullemans@apolloway.prl.philips.nl⟩
- Johan Vromans ⟨jv@mh.nl⟩
- ⟨craig.r.stevenson@att.com⟩
- ⟨erikt@cs.umu.se⟩
- ⟨wstahw@lso.win.tue.nl⟩
- ⟨mai@wolfen.cc.uow.oz⟩
- Johan Garpendahl ⟨garp@isy.liu.se⟩
- John Interrante ⟨interran@uluru.stanford.edu⟩
- Chris Adamo ⟨adamo@ll.mit.edu⟩
- Larry Barbieri ⟨lbarbieri@ll.mit.edu⟩
- Denis Girou ⟨Denis.Girou@idris.fr⟩
- Thomas Parmelan ⟨Thomas.Parmelan@efrei.fr⟩

Some people helped to the translation:

- Daniele Ghiotti (Italian)
- Tomek Burdziak (Polish)
- Miguel A. Varo ⟨mvaro@dlsi.ua.es⟩ (Maintainer for Spanish and Catalan)
- Michael Wiedmann ⟨mw@miwie.in-berlin.de⟩ (Maintainer for German)
- Christian Kirsch ⟨ck@held.mind.de⟩ (German)
- Erwin Dieterich ⟨debaycrk@ibmmail.com⟩ (German)
- Juliusz Chroboczek ⟨jec@dcs.ed.ac.uk⟩ (Polish)
- Marcel van der Laan ⟨marcel@spase.nl⟩ (Dutch)



## A.2 Thanks

Juliusz Chroboczek worked a lot on the integration of Ogonkify's products (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').

A big thank must be given to Markku Rossi (the author of GNU enscript), for the advises he gave, and for portions of his code which are now used in a2ps. Even portions of this very documentation are taken from the GNU enscript man pages.

# Appendix B 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


## $

'$' ... 21

## .

'.a2ps' ... 26

## ^

'^' ... 21

## A

'a2psrc' ... 26
Alphabets ... 19
ASCII ... 9

## C

Command line options ... 6
Configuration files ... 26
Copying ... 37
CP-1250 ... 10

## D

'DefaultPrinter:' ... 26

## E

elm ... 5
Encoding ... 1
Encodings ... 9
Escapes ... 20
Exit status ... 16

## F

Face ... 2

## H

Headers ... 9
HP encoding ... 10

## L

Latin 2 ... 10
Latin 3 ... 10
Latin 4 ... 10
Latin 5 ... 10
Latin 6 ... 10
latin1 ... 10
Library files ... 16

## M

Macintosh encoding ... 10
Markers ... 20
Meta sequences ... 13

## N

Non PostScript printers ... 26

## O

Ogonkify ... 10
Options ... 6

## P

Page device ... 11
Page prefeed ... 11
PC encoding ... 10
PC graphic encoding ... 10
pine ... 5
PreScript ... 22
Pretty printing ... 17
'Printer:' ... 27
Prologue ... 10

## Q

Quotation in PreScript ... 24

## R

Regular symbols ... 19

## S

Sequences ... 20
setpagedevice ... 11
Special symbols ... 20
statusdict ... 11
Style sheet ... 2
Style sheets ... 19
Symbol conversion ... 17
Symbols ... 23

## U

Under lay ... 9
'UnknownPrinter:' ... 27
'UserOption:' ... 27

## V

Verbatims ... 20
Virtual page ... 1

## W

Water mark ... 9

# Table of Contents


**1 Introduction ..... 1**
- 1.1 Description ..... 1
- 1.2 Vocabulary ..... 1

**2 User Guide ..... 3**
- 2.1 Purpose ..... 3
- 2.2 How to print ..... 3
- 2.3 Important parameters ..... 4
- 2.4 Localizing ..... 4
- 2.5 Interfacing with other programs ..... 5
  - 2.5.1 elm ..... 5
  - 2.5.2 pine ..... 5

**3 Invocation ..... 6**
- 3.1 Command line options ..... 6
  - 3.1.1 Global behavior ..... 6
  - 3.1.2 Sheets ..... 7
  - 3.1.3 Pages ..... 8
  - 3.1.4 Headers ..... 9
  - 3.1.5 Input ..... 9
  - 3.1.6 Output ..... 11
  - 3.1.7 Pretty printing ..... 12
- 3.2 Meta sequences ..... 13
- 3.3 Page device options ..... 15
- 3.4 Statusdict options ..... 16
- 3.5 Library files ..... 16
- 3.6 Exit status ..... 16

**4 Pretty printing ..... 17**
- 4.1 Syntactic limits ..... 17
- 4.2 Automatic style ..... 17
- 4.3 Known languages ..... 18
- 4.4 Definition of the style sheets ..... 19
  - 4.4.1 Name and abbreviations ..... 19
  - 4.4.2 Alphabets ..... 19
  - 4.4.3 Keywords and regular symbols ..... 20
  - 4.4.4 Special symbols ..... 20
  - 4.4.5 Sequences ..... 20
  - 4.4.6 Escapes and verbatims ..... 20
  - 4.4.7 Case sensitivity ..... 20
- 4.5 Writing new style sheets ..... 20





**5 PreScript ..... 22**
- 5.1 Syntax ..... 22
- 5.2 `PreScript` commands ..... 22
  - 5.2.1 Structure commands ..... 22
  - 5.2.2 Printed symbols ..... 23
  - 5.2.3 Special characters ..... 23
  - 5.2.4 Quotation ..... 24
- 5.3 Differences with LaTeX ..... 24
  - 5.3.1 LaTeX commands ignored in `PreScript` ..... 24
  - 5.3.2 Commands available in PreScript but not in LaTeX ..... 24
- 5.4 Examples ..... 25

**6 Customizing a2ps ..... 26**
- 6.1 Configuration files ..... 26
  - 6.1.1 Your default options ..... 26
  - 6.1.2 Your guess rules ..... 26
  - 6.1.3 Your printers ..... 26
  - 6.1.4 Your shortcuts ..... 27
  - 6.1.5 Your PostScript magic number ..... 27
- 6.2 Designing prologues ..... 28
  - 6.2.1 a2ps faces ..... 28
  - 6.2.2 Definition of the faces ..... 28
  - 6.2.3 Colors in PostScript ..... 29
  - 6.2.4 A step by step example ..... 29

**7 Various ..... 32**
- 7.1 Non PostScript printers ..... 32
  - 7.1.1 From Clayton Weaver ..... 32
  - 7.1.2 From Pierre Juillot ..... 32
  - 7.1.3 These tips in a2ps ..... 33
- 7.2 Frequently asked questions ..... 33
  - 7.2.1 Printing goes beyond the frame of the paper ..... 33
  - 7.2.2 Helping on the translation ..... 33
  - 7.2.3 Why not having used `yacc` and such ..... 33
  - 7.2.4 Man pages ..... 34
  - 7.2.5 Can I change the base font ..... 34
- 7.3 What next? ..... 34
  - 7.3.1 EPSF ..... 34
  - 7.3.2 Spelling checking, syntax fixing. ..... 34

**Appendix A Genesis ..... 35**
- A.1 History ..... 35
- A.2 Thanks ..... 36

**Appendix B Copying ..... 37**

**Concept Index** ..... **38**