Philip's Music Writer

A Music Typesetting Program

by

Philip Hazel

This edition documents PMW 4.05
April 2004

Contents

1. Introduction

Philip's Music Writer (PMW) is a computer program for typesetting music. It is not a music processor or a sequencer. Its sole objective is the production of high quality printed sheet music.

This edition of the manual describes version 4.05. Version 4.00 was the first version for Unix and Unix-like systems. Previous versions are called *Philip's Music Scribe*, and have been running on Acorn's RISC OS operating system since the early 1990s.

PMW operates by reading an input file containing an encoded description of the music; such a file can be constructed using any text editor or wordprocessor. The output of this version of PMW is a PostScript file. This can be previewed on screen using *GhostScript* or similar software. If you have a PostScript printer, PMW output can be printed directly; otherwise, conversion software such as *GhostScript* is required.

As well as PostScript output, PMW can also write a MIDI file. This can be played by an application such as *Timidity*. It is not very sophisticated, and is intended for 'proof-hearing' purposes only.

Although a textual input method may not be considered as 'user-friendly' as pointing and dragging on the screen, it can be be a much faster way of inputting music, once the format of the input file has been learned. In addition, the usual facilities of a text editor, such as cutting and pasting, can be used to speed up entry, and PMW is also able to provide text-based features such as macros and included files.

PMW comes with a PostScript font called PMW-Music, which contains all the musical shapes (notes, rests, accidentals, bar lines, clefs, etc.) that PMW requires. I acknowledge with gratitude the help of Richard Hallas, who originally created some of the characters in this font and improved many others. He also contributed a second font called PMW-Alpha which contains additional characters that are useful when printing music (see chapter 49).

The PMW input encoding is designed to be easy for a musician to remember. It makes use of as many familiar musical notations as possible within the limitations of the computer's character set. Normally it will be input by a human using any available word processor or text editor. There is no reason, however, why PMW input should not form the *output* of some other computer program that captures (or generates) music in another fashion.

The next chapter is a short summary of the musical and other terminology used in this manual. The following chapters describe how PMW should be installed and operated.

Chapters 5 to 19 are an introduction to the PMW input encoding. They cover most of the more common requirements, with examples, in an introductory manner.

The bulk of the manual (from chapter 20 onwards) is reference material for the PMW input encoding; much of the information in earlier chapters is repeated, together with additional specialist information.

Finally, there are discussions of the PMW musical fonts, summaries of the syntax of input files, and an index.

2. Terminology

The word ‘default’ occurs frequently in this manual. It means some value or option setting that is used when the user doesn’t supply any alternative. For example, if no key signature is given for a piece, the default that is used is C major.

The word ‘parameter’ is used in discussions about PMW directives to refer to the various values that are given as part of the directive. For example, the directive to set the page length has to be followed by one number; this is called its parameter.

Some formal musical terminology is also used; it is summarized here for the benefit of readers who may not be fully familiar with it. I use the British names for notes: breve, semibreve, minim, crotchet, quaver, semiquaver, etc.

A *beam* is a thick line that is used to join together a number of quavers or shorter notes, instead of each note having its own separate flags.

A *brace* is a curly sign that is used to join together two staves that are to be played on a single instrument, for example the two staves of conventional keyboard music.

A *bracket* is another sign used for joining staves together. It is like a large square bracket and is used to join together staves of music for different instruments, for example, the four staves needed for a string quartet.

A *caesura* is a pause mark that appears between notes; it is normally printed as two short sloping lines through the top line of the stave.

A *fermata* is the pause mark that is placed over or under notes, consisting of a semicircle with a dot at its centre.

A *flag* is the name used for the additional marks added to the stem of a note to indicate that it is shorter than a crotchet. A quaver has one flag, a semiquaver has two, and so on.

Overlay is the word used to describe text that is printed above a stave in a vocal part. Usually, words are printed below the stave, and are called *underlay* (see below), but occasionally alternative words are printed above.

A *stave* is a single set of lines on which notes are printed. The normal stave contains five lines, but other numbers of lines are sometimes used, for example, a single horizontal line for percussion parts.

The *stem* of a note is the vertical line that is drawn either upwards or downwards from the notehead, for all notes shorter than a semibreve.

A *system* is a single line of music, comprising one or more staves, and usually joined at the left-hand edge in some way. For example, the systems of a string quartet each contain four staves.

Underlay is the word used to describe text that is printed under a stave in a vocal part, that is, the words which are to be sung. The less formal term ‘lyrics’ is often used for this in the context of popular songs.

In the PMW documentation the word ‘stave’ is used as the singular of ‘staves’. However, the program itself accepts ‘staff’ as a synonym of ‘stave’ under all circumstances.

3. Installing PMW

The reader is assumed to be familiar with using shell commands on Unix-like systems. PMW is installed in the same way as many other applications. First, download the tarball into a suitable directory. You should end up with a file with a name like this:

```
pmw-4.05.tar.gz
```

Uncompress the file with *gunzip* and then unpack the archive:

```
gunzip pmw-4.05.tar.gz
tar -xf pmw-4.05.tar
```

This creates a directory called **pmw-4.05**, containing a number of files and directories. Of interest for later are the **doc** directory, which contains documentation, and the **contrib** directory, which contains files that have been contributed by PMW users in the hope they may prove useful to others. Each of these contributed files has comments at the top, explaining what its contents are.

To build and install PMW, you can now issue these commands:

```
./configure
make
make test
make install
```

By default, this installs into the **/usr/local** directory. If you want to install PMW somewhere else, you can specify a different 'prefix' when configuring. For example:

```
./configure –prefix=/usr/local/pmw
```

The files that are installed in the prefix directory are as follows:

- **bin/pmw** is the PMW command.
- **man/man1/pmw.1** is a short 'man' page that describes the command options for PMW.
- **share/pmw/PSheader** is a PostScript header file for PMW output.
- **share/pmw/psfonts/PMW-Music.pfa** is the main PostScript music font. As of release 4.03 it is a Type 1 PostScript font – hence the `.pfa` extension.
- **share/pmw/psfonts/PMW-Alpha** is an auxiliary music font. This is still a Type 3 PostScript font (so no extension).
- **share/pmw/fontmetrics/** is a directory that contains font metric files giving character widths and kerning information for the standard PostScript fonts and the PMW music fonts.

Once you have installed PMW, you can use the command to generate PostScript from input files, as described in chapter 4 below. However, before you can print pages or view the output on the screen, you need to arrange for the PostScript music fonts to be available for your printer or viewer. Exactly what you have to do varies between systems. I hope the following instructions will give enough hints to cover most cases.

3.1 Viewing PMW output on the screen

The *GhostScript* application can be used to view PMW output on screen. As well as the basic **gs** command, there are some front-end applications with names such as **ghostview**, **gview**, or **gv**, which package the user interface to *GhostScript* in various more friendly ways. Make sure you have one of these installed.

Before *GhostScript* can display a PMW output file, it needs to be told where the PostScript Music fonts are. Typically, you need to install symbolic links from a suitable font directory to the two fonts that were installed in **share/psfonts**. You can find out which directories *GhostScript* searches for its fonts by running the following command:

```
gs -h
```

For example, for *GhostScript* running on Debian Linux, the links to the font files can be placed in **/var/lib/defoma/gs.d/dirs/fonts**, so you might use these commands:

```
ln -s /usr/local/share/pmw/psfonts/PMW-Music.pfa \
      /usr/lib/defoma/gs.c/dirs/fonts/PMW-Music.pfa
ln -s /usr/local/share/pmw/psfonts/PMW-Alpha \
      /usr/lib/defoma/gs.c/dirs/fonts/PMW-Alpha
```

On Redhat Linux, something like this is appropriate:

```
ln -s /usr/local/share/pmw/psfonts/PMW-Music.pfa \
      /usr/share/fonts/default/ghostscript/PMW-Music.pfa
ln -s /usr/local/share/pmw/psfonts/PMW-Alpha \
      /usr/share/fonts/default/ghostscript/PMW-Alpha
```

In addition to creating these links, you need to add this line to the file called **Fontmap** that is found in the *GhostScript* font directory:

```
/PMW-Music (PMW-Music.pfa) ;
```

This tells *GhostScript* that the font called 'PMW-Music' is to be found in the file called **PMW-Music.pfa**.

Once you have made these links and edited **Fontmap**, you can test whether *GhostScript* is working correctly by attempting to display the expected output from the test files. There are six such files in the **testdist/outfiles** directory of the PMW distribution. For example (assuming you have the **gv** command installed):

```
gv testdist/outfiles/Test01.ps
```

This is a page of a Mozart mass. If the fonts are not correctly installed, you should see an error message.

Hint for when you are creating your own input: the **gv** command has a useful option called 'watch file'; it causes the file to be re-displayed whenever it changes. If you set this and leave **gv** running, you can edit the input and reprocess it with PMW, and **gv** will notice the updated output file and re-display it at the same position.

3.2 Antialiasing and the screen display

When it is interpreting PostScript for display on the screen, *GhostScript* can be run with or without antialising, which is a technique for making text look better by adding pixels in various shades of grey round the edges of characters, to fool the eye into seeing less jagged outlines. Before the PMW-Music font was converted to a Type 1 font, this could give problems with some of the straight-edged shapes. With the Type 1 font, there should be no problem with antialiasing. However, the PMS-Alpha font is still a Type 3 font; if you make use of PMW-Alpha, the screen display of some characters may be odd.

Fortunately, this problem applies only to screen display. Printers have a much higher pixel resolution, and antialising would not be needed even if it were possible (which it definitely is not on black-and-white printers).

3.3 PDF files

You can use a command such as **ps2pdf** (which comes with *GhostScript*) to turn a PostScript output file from PMW into a PDF file. If you are using release 8 or later of *AFPL GhostScript*, characters from the PMW-Music font are included as outlines, which means that the PDF can be displayed nicely at any size on the screen. Earlier releases of *GhostScript* include the musical characters as bitmaps, which does not give such a good display. Characters from the PMW-Alpha font are still included as bitmaps, because it is still a Type 3 font.

3.4 Printing PMW output on a non-PostScript printer

If you do not have a PostScript printer, or one that can interpret PostScript directly, you have to use an application such as *GhostScript* to interpret the output of PMW and convert it for your printer.

3.5 Printing PMW output on a PostScript printer

The PostScript output that PMW generates is not totally freestanding. It expects the PMW PostScript music font to be loaded into the printer in advance. If this has not been done, an error will be generated.

If you have full control of the printer, you can load the Music font(s) into it once, and then send any number of music files to be printed. To do this, you need to know the printer's password. Then you must make a copy of the PMW music fonts with the password included, for sending to the printer. The fonts are distributed in the **psfonts** directory in the PMW distribution. Near the top of each font file you will find these lines:

```
%%BeginExitServer: 000000
%%serverdict begin 000000 exitserver
%%EndExitServer
```

The value 000000 is the default password which new PostScript printers have. If you haven't changed it, all you need to do is to remove the two percent signs (which are PostScript comment characters) at the start of the second line, so that it reads as follows:

```
serverdict begin 000000 exitserver
```

Then if you 'print' this file, the font will be permanently loaded into the printer (until it is powered off). **Note:** you must not make this change on the copy of the font that is to be used by *GhostScript*, because *GhostScript* does not cope with such lines.

When a font is loaded into a PostScript printer, it generates a warning message which may be returned to you. The message is

```
%%[ exitserver: permanent state may be changed ]%%
```

This is perfectly normal and can be ignored.

If you do not have full control over the printer, or do not want to load the fonts permanently, you should arrange to insert a copy of the PostScript font at the start of the file before printing (both fonts, if you are using PMW-Alpha). For example, you could use a command line like this:

```
cat /usr/local/share/pmw/psfonts/PMW-Music.pfa myscore.ps | lpr
```

This can be wrapped up into a script for convenience. You do not need to include the PMW-Alpha font unless you have used it in the PMW input file.

If you have several different PMW PostScript files to print, they can all be concatenated into one file for printing, with a single copy of the font(s) on the front.

4. The command for running PMW

The PMW command has the following form:

```
pmw [<options>] [<input file>]
```

where the items in square brackets are optional. If no file name is given, input is read from the standard input and by default the output is written to the standard output. When a file name is given, the default output file name is the input file with the extension *.ps* replacing any existing extension, or being added if there is no extension. The default output destination can be overridden in all cases by using the **-o** option.

Error messages and verification output are written to the standard error file, which can be re-directed in the usual way. Here are some examples of PMW commands:

```
pmw sonata 2>errors
pmw -p 3-4 mozartscore
pmw -format A5 -a5ona4 -pamphlet myscore
pmw -s 3 -o quartet.ps-viola quartet.pmw
```

The available options are as follows:

-a4ona3

The **sheetwidth** and **sheetdepth** parameters define the size of the page images PMW produces (see chapter 33). In the common case, this size is identical to the size of paper that is being used, in which case one image fits exactly on one piece of paper. However, PMW also supports *two-up* printing, in which two page images are printed next to each other on a larger piece of paper. This option specifies that the images are A4-sized, but are to be output two-up, assuming A3 paper.

-a5ona4

The pages are A5-sized; print them two-up, assuming A4 paper.

-a4sideways

The paper is A4, but the printer feeds it sideways, so rotate the page images before printing.

-c <*number*>

Set the number of copies to be printed as <*number*>. This number is honoured by PostScript printers. It may not be honoured by other programs that interpret PostScript.

-debug

Write debugging information to the standard error file (not currently very comprehensive).

-eps

Write the output as encapsulated PostScript (useful if this is an illustration that is going to be included in some other document).

-F <*directory*>

Use the given directory as the fontmetrics directory, instead of the default that was set up when PMW was built.

-f <*name*>

Specifies a format name, used when the input file is set up to output in several different formats. The format can be tested the ***if** directive in a PMW input file. This mechanism is used when different forms of the output are being generated from a single input file. For example, a piece might be arranged for either flutes or recorders. The user chooses words to describe each different format, and specifies the appropriate one here. Details of how to set up the input so that different headings etc. are printed when different staves or formats are selected are given in chapter 19.

-H <*file*>

Use the given file as the PostScript header file, instead of the default that was set up when PMW was built.

-help

Output a list of options, then stop. No file is read.

-MP <*file*>

Use the given file as the MIDIperc file, instead of the default that was set up when PMW was built. This file contains translations between names and MIDI 'pitches' for untuned percussion voices. Apart from comment lines (starting with #) and empty lines, each line in the file must begin with three digits, followed by a space, and the instrument name, without any trailing spaces. For example:

```
035 acoustic bass drum
036 bass drum 1
037 side stick
038 acoustic snare
```

-MV <*file*>

Use the given file as the MIDIvoices file, instead of the default that was set up when PMW was built. This file contains translations between names and MIDI voice numbers. Apart from comment lines (starting with #) and empty lines, each line in the file must begin with three digits, followed by a space, and then the instrument name, without any trailing spaces. The same number may appear more than once. For example:

```
001 piano
001 acoustic grand piano
002 hard piano
002 bright acoustic piano
003 studio piano
003 electric grand piano
```

-manualfeed

Set the 'manualfeed' option in the generated PostScript. Most PostScript printers interpret this to mean that the paper should be taken from an alternate input tray or slot. Some also require the user to push a button before each page is printed.

-midi <*file*>

This option specifies that MIDI output should be written to the given file. This is in addition to the PostScript output. Only a single movement can be output as MIDI; when the input file contains multiple movements, the **-midimovement** option (synonym **-mm**) can be used to select which one this is. The stave selection specified by **-s** applies, and the bars that are output can be selected by **-midibars** (synonym **-mb**). The page selection option does not apply to MIDI output.

A number of directives whose names all start with 'midi' are available for controlling the allocation of MIDI voices and channels to staves. The **midichannel** heading directive is used to specify the allocation of a MIDI voice and/or particular PMW staves to a MIDI channel, and the **[midichannel]**, **[midivoice]**, and **[midipitch]** directives are used to change the setup in the middle of a piece. For percussion staves, where the playing pitch selects different instruments, the **[printpitch]** directive can be used to force the printed notes to a single pitch.

If the input file contains no MIDI-specific directives, all notes are played through MIDI channel 1. The voice allocation on the channel is not changed, so whatever MIDI voice is assigned to the channel is used.

The 'velocity' parameter for each note corresponds to the volume setting, since 'velocity' controls the volume on many MIDI instruments. If relative volumes are set by means of the **playvolume** or **[playvolume]** directives, the overall volume is multiplied by the relative volume and then divided by 15 (the maximum relative volume). Thus, for example, if the overall volume is 64 and a stave has a relative volume of 10, its notes are played with a 'velocity' of 42.

-midibars <*start*>**-**<*end*>

Limits the bars that are written to a MIDI file to the specified range (**-mb** is a synonym). If this option is not given, the entire movement is included in MIDI output. The page selection option does not apply to MIDI output. If the end bar number is omitted, but the hyphen is present, output continues to the end of the movement. If just one number is given, just one bar is output.

-midimovement <*number*>

This option specifies which movement is to be output as MIDI (**-mm** is a synonym). Only one movement can be output in this manner. The default is the first movement in the file.

-norepeats

When generating a MIDI output file, do not repeat repeated sections of the music (**-nr** is a synonym).

-o <*file*>

Send the PostScript output to the given file, or, if a single hyphen is given as the file name, to the standard output.

-p <*list*>

Output only the specified pages. These can be individual page numbers, or pairs of numbers separated by a hyphen, to specify a range. Use commas to separate items in the list. For example,

```
  pmw -p 4,6,7-10,13
```

specifies that pages 4, 6, 7 to 10, and 13 are to be output. The page selection does not apply to MIDI output; use **midibars** and **midimovement** instead.

-pamphlet

The **pamphlet** page ordering option is useful when a two-up page output format is selected. In pamphlet mode, the piece is notionally extended with blank pages, if necessary, so that the number of the last page is a multiple of four. Page 1 is then paired with the last page, page 2 with the second last page, and so on. The odd-numbered page of the pair is always output on the right.

If the first page of the piece has a number greater than 1, then earlier pages are simply output as blanks, as are any internal missing pages (which can be created by using page increments other than one, or by explicitly skipping pages).

Outputting both sides at the same time in pamphlet mode is useful for producing master copies for reproduction elsewhere. If you want to produce a final two-sided copy directly, then use **-pamphlet** with **-printside** 1 to output all the first sides, and then use **-printside** 2 to output the second sides for printing on the backs of the same sheets.

When selecting individual pages to output with the pamphlet option, you should select only one member of each pair. The partner page is automatically added to each selected page, so selecting both pages will result in two copies being output.

For two-up printing, PMW automatically centres the page images in the half pages in which they appear, except that in pamphlet mode they are abutted together in the middle. This means that, when the sheetsize is smaller than half the paper size, any marks printed outside the sheetsize (crop marks, for example) are visible.

-printadjust <*x*> <*y*>

Experience has revealed that not all printing methods position the image in exactly the same position on the page. These two values specify a movement of the image on the page, in printers' points (1/72 of an inch). The movement is relative to the normal reading orientation of the page images (which may be rotated on the paper). The first value is left-to-right, and the second is up and down. Positive values move the image to the right and upwards, respectively, while negative values move it in the opposite directions.

-printscale <*n*>

Scale the output image by <*n*>.

-printside <*n*>

Output only odd or only even pages; <*n*> must either be 1 or 2. The side selection options make it easy to print on both sides of pages by feeding them through the printer twice, without having to set up an explicit page selection each time. When pamphlet mode is selected, it is the lower of the two page numbers that is tested. When a 2-up non-pamphlet mode is selected, this option is disabled, and all selected pages are always output.

-reverse

Output the pages in reverse order. The default order is in ascending sequence of page number if no pages are explicitly selected; otherwise the order is as selected by the user. Reverse order is precisely the opposite of this. It is useful for printers which stack face-up, and also in some two-sided printing operations.

-s <*list*>

Output only the specified staves. These can be individual stave numbers, or pair of numbers separated by a hyphen, to specify a range. Use commas to separate items in the list. For example:

```
pmw mozart -s 1,3-5,9-12
```

Setting values here is how you select one or more individual parts to be printed from a score. For example, in a work for choir and orchestra, to create a vocal score by printing only the voice parts, one might specify 11–14 if the vocal parts were on staves 11–14. More often, just a single number is given, in order to print out an individual instrumental part.

-t <*number*>

Specify a transposition, in semitones. A positive number specifies upwards transposition, and a negative one downwards transposition. A transposition of zero may also be entered; this is not the same as no transposition at all. For more details about transposition, see chapter 29.

-V

Output the PMW version number to the standard output, then stop. No file is read.

-v

Output verification information about the typesetting to the standard error file. The information is described in the next section.

4.1 Information about the piece

To understand all of this section, you need to be familiar with the way PMW handles pitches and dimensions. It is placed here because it follows on from the command line options, but it is best skipped on a first reading.

Here is an example of the information which is output when **-v** is selected:

```
Data store used = 76K (stave data 37K)

MOVEMENT 1

Stave  1: 51 bars; range   E′   to  A′′  average  A′
Stave  2: 51 bars; range  $B    to  D′′  average  E′
Stave  3: 51 bars; range   E′   to  F′′  average $B′
Stave  4: 51 bars; range   F‘   to  D′   average  D

PAGE LAYOUT

Page 1 bars: 1-4 5-8 (3) 9-12
  Space left on page = 131 Overrun = 61
Page 2 bars: 13-17 18-22 23-25 (10) 26-28
  Space left on page = 5
Page 3 bars: 29-31 32-34 35-38
  Space left on page = 159 Overrun = 33
Page 4 bars: 39-42 (15) 43-46 47-48 49-51
  Space left on page = 5
```

For each movement within the piece, PMW displays a bar count for each stave, the pitch range of notes on the stave, and the average pitch. The count includes only properly counted bars; if there are any uncounted bars, they are shown in brackets with a plus sign. For example, if a piece starts with an uncounted, incomplete bar, the bar count might be shown as ‘24(+1)’.

The pitches are specified at octave zero, that is, starting at the C below middle C. The average pitch of a vocal part is some kind of measure of the tessitura.

If there is more than one movement in a piece, the overall pitch ranges and average pitches for each stave are given at the end.

The ‘page layout’ section shows how PMW has laid out the music on the pages. In the example above, three systems have been put on page 1, containing bars 1–4, 5–8, and 9–12, respectively. If any system is too short to be stretched out to the full line length (or if stretching was not requested) an asterisk is printed after it.

After the range of bars for each system, the amount of horizontal overrun is given in brackets, provided it is less than 30 points. The overrun is the distance by which the linelength would be exceeded if another bar were added to the system.

The first line in the example above means that bars 5–9 were three points too long for the linelength, which is why the second system was ended after bar 8. This information can be useful when you are trying to alter the way the bars are allocated to systems.

‘Space left on page’ is the amount of vertical space left on the page. It is the amount by which stave or system spacings can be increased without causing the bottom system to be moved over to the next page.

‘Overrun’ is the amount of extra space that is needed to fit another system onto the page. It is the amount by which stave or system spacings would have to be reduced in order for the first system of the next page to be brought back onto the bottom of this page. It is not shown if the value is greater than 100 or if the page break was forced.

4.2 PMW input errors

When PMW detects an error in the input file, it writes an error message to the standard error file. In most cases it carries on processing the input file, so that as many errors as possible are detected in the run. As is the case in many programming languages, certain kinds of error can cause it to get confused and give misleading subsequent messages. If you don’t understand all the error messages, fix those that you do, and try again.

It is very easy to make simple typographic errors that leave a bar with the wrong number of notes in it. An example of the message that PMW outputs is as follows:

```
** Incorrect length for bar 1, stave 1 - too long by 1 quaver
** Near line 17:
rrf’-g |
        <
```

In this case a minus sign (indicating a quaver) had been omitted after the note g, which is therefore taken as a crotchet. The input line in which the error was detected is shown, and the character ‘<’ is output underneath the position where the error was detected. In this example, PMW has just reached the bar line.

The line number is given using the phrase ‘near line *n*’ because sometimes PMW has read on to the next line before detecting the error.

Most errors cause PMW to stop processing before it writes anything to the main output. However, there are a few errors which do not prevent output. An example is the detection of a bar that is too wide for the page; PMW diagnoses this, and then squashes it to fit.

5. Getting started with PMW encoding

In this and the next few chapters we will cover the basic facilities of the way PMW input is encoded, omitting some of the more exotic features in order to keep the explanations simple. Full information is given in the reference section, which starts at chapter 20.

We start with the first six bars of the British National Anthem. It is suggested that you try out these examples as you read this section. First, use your favourite text editor to create a file containing this example:

```
heading "|National Anthem"
breakbarlines
underlaysize 9.5
notespacing *1.1
key G
time 3/4

[stave 1 treble 1 text underlay]
"God save our gra-cious Queen,"
g g a | f. g- a |
"Long live our no-ble Queen,"
b b c’ | b. a- g |
"God save the Queen."
a g f | G. |
[endstave]

[stave 2 bass 0]
g` b` c | d. e- f | g e c | d. #d- e | c d d | G`. |
[endstave]
```

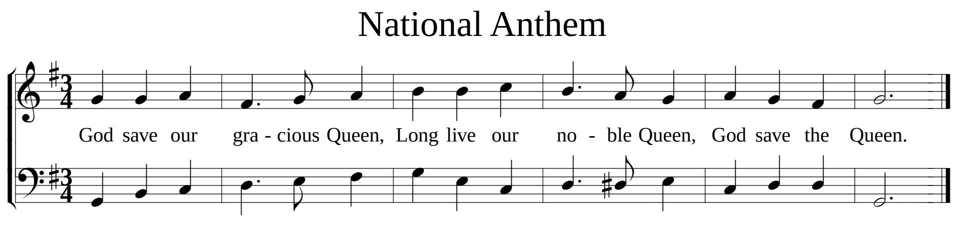
You may use any name you like for the file, and you should put it in any convenient directory. Let’s suppose it’s called *natanth*. Now run the file through PMW via this command:

```
pmw natanth
```

Assuming you have not made any typing mistakes, there will be no output on the screen, but a new file called *natanth.ps* will have been created. You can view this on screen by a command such as:

```
gv natanth.ps
```

(or by using any other PostScript viewer). The output should look like this:


National Anthem
God save our gra - cious Queen, Long live our no - ble Queen, God save the Queen.

If you have made a mistake, one or more error messages will be written to the standard error file, and should therefore appear on your screen. The messages should be self-explanatory. Correct the error(s), and try again.

If you did not make any typing mistakes, you might like now to deliberately introduce one or two, to gain familiarity with error handling. Omitting one of the | characters is a common mistake.

We now proceed to explain what the different parts of this input file mean to PMW. The data is in two parts: first there is heading information, such as the printed heading and key and time signatures for the piece, and then the music for each stave is given separately.

The heading in this example contains six *heading directives*. They have been put on separate lines for readability, but this is not a requirement; you can have several directives on one line if you like.

The first directive,

```
heading "|National Anthem"
```

gives a text heading for the piece, and the text itself must be supplied inside double quote marks. Heading lines normally consist of a left part, a centred part, and a right part. The division between these is marked by a vertical bar character in the text. This example prints nothing at the left (because there is nothing before the vertical bar), and nothing at the right (because there isn't a second vertical bar).

The second directive,

```
breakbarlines
```

causes PMW to make a break in the bar lines after each stave. Without this, the bar lines would be drawn continuously from the top of the first stave to the bottom of the second. It is conventional not to have bar lines between staves when there is vocal underlay (lyrics), as they can get in the way of the words.

In orchestral pieces you may want to have bar line breaks between different groups of instruments, and this can be achieved by listing the stave numbers after which you want the breaks, like this:

```
breakbarlines 4, 8, 12
```

The third directive,

```
underlaysize 9.5
```

sets the font size for the underlay text (i.e. the words). Font sizes are given in *points*, the traditional measure of type size used by printers. The default size for all text in PMW is 10 points; choosing a slightly smaller size for underlay is often helpful in fitting in the words.

Please note that the music shown above and in all the following examples in this manual is shown at 0.8 times its proper size, so the type sizes you see here are smaller than they will be if you print the example yourself.

The fourth directive,

```
notespacing *1.1
```

is an instruction to PMW to increase its normal horizontal note spacing by a factor of 1.1 (the asterisk is being used as a multiplication symbol). The standard note spacing is suitable for instrumental music. When vocal underlay is involved, it often improves the layout if the spacing is increased by 1.1 or 1.2.

PMW automatically increases the space between two notes in a bar if this is necessary to avoid two underlaid syllables colliding, but if this happens a lot, the spacing of the notes can look very strange. It is best to set the note spacing sufficiently wide that most of the layout is determined by the music, with only the occasional adjustment for the words.

The fifth directive,

```
key G
```

sets the key signature. If no key signature is given, C major is assumed. Minor keys are given by adding the letter 'm', for example, `Am`. Sharp and flat key signatures are given using the standard accidental notation in PMW. A sharp is represented by the character #, which is easily remembered. Unfortunately, there are no keys on the computer keyboard which resemble flats or naturals, so instead the two keys that are next to # on some keyboards were chosen: `$` for a flat and `%` for a natural. The key signatures C sharp minor and G flat major are coded as `C#m` and `G$` respectively.

The sixth directive,

```
time 3/4
```

sets the time signature. If no time signature is given, 4/4 is assumed. As well as the usual numeric time signatures, the letters `C` and `A` can be given, signifying 'common' and 'alla breve' time. These are printed as 𝄴 and 𝄵 respectively.

The stave data starts with the first line that begins with a square bracket,

```
[stave 1 treble 1 text underlay]
```

You will notice that a bit further down there is a line containing

```
[endstave]
```

This marks the end of the data for the first stave. Each stave's data is always contained between **[stave]** and **[endstave]** in this way.

The data itself consists of a mixture of encoded music, words, bar lines, and so on, and *stave directives*. To make it clear what is what, the stave directives are always enclosed in square brackets, and we will show them in brackets whenever they are mentioned in the text. Several stave directives can appear in succession within a single pair of brackets.

The number following the word 'stave' in the **[stave]** directive gives the number of the stave. The top stave of a system is numbered 1, the next one down is numbered 2, and so on. PMW can handle up to 63 staves.

Usually, a clef-setting directive comes next, as in both staves of this example, where the first stave uses the treble clef and the second stave the bass clef. The number which follows the clef name sets the *current octave* for the notes of the stave.

PMW octaves run from C up to B, and octave number 1 starts at middle C. It is usual, therefore, to set the current octave to 1 when using the treble clef, and to 0 when using the bass clef, as has been done here.

There is only one other stave directive in this example, and that is

```
  [text underlay]
```

What this is doing is to set the default type for any text strings encountered in the first stave. PMW supports several different kinds of text, as we shall see later, and one of them can be set as the default for a stave. Instances of strings of other types then have to be marked as such. When a stave has vocal underlay in it, it is usual to set the default as above, because by far the majority of the text will consist of the underlay.

So at last we come to the music and words of the first stave:

```
  "God save our gra-cious Queen,"
  g g a | f. g- a |
  "Long live our no-ble Queen,"
  b b c' | b. a- g |
  "God save the Queen."
  a g f | G. |
```

The vocal underlay is given as several text strings, each preceding the notes to which it relates. You can split up underlay into strings that are as long or as short as you like. PMW automatically distributes the syllables to the notes that follow. Single hyphens are used to separate the different syllables of the words, as in 'gra-cious' and 'no-ble', and PMW supplies as many printed hyphens as necessary to fill the space between them when they are printed.

The music itself is divided up into bars by the vertical bar character. PMW checks that the contents of a bar agree with the time signature, and complains if there are too many or too few notes.

The notes themselves are encoded using their familiar letter names. Because we set the current octave to be octave 1, the letter `g` in the first bar represents the G above middle C. The only note on this stave that does not lie in octave 1 is the last note of the third bar, the C above middle C. It is encoded as `c'` because each quote that follows a note letter raises the note by one octave.

The duration of a note is primarily determined by whether a capital (upper case) letter or small (lower case) letter is used. A lower case letter stands for a crotchet, while an upper case one is used for a minim, as in the last bar of this stave.

Further characters are used to adjust the duration: a minus sign (hyphen) after a lower case letter turns the crotchet into a quaver, the hyphen being mnemonically like the flag used to distinguish a printed quaver from a crotchet. A dotted note is coded simply by adding a full stop, as in the second, fourth, and last bars.

Turning now to the second stave,

```
  g` b` c | d. e- f | g e c | d. #d- e | c d d | G`. |
```

we see two new features. The first two notes, and the last one, are below the current octave for this stave, which was set as octave 0 (one below middle C). To lower a note by one octave, a grave accent is used, because it is a symbol which is the 'opposite' of an ordinary quote.

In bar four there is a note with an accidental. Accidentals are entered before note letters because they print before notes. The characters used for accidentals were described above when discussing key signatures, but to remind you,

`#`	is used for a sharp
`$`	is used for a flat
`%`	is used for a natural

Should you need double sharps or double flats, just type the character twice.

The spacing used in this example was chosen to make it easy to read. PMW does not require spaces to appear between notes or before bar lines, so the first two bars of the first stave could equally well appear as

```
gga|f.g-a|
```

However, spaces must not be used between any of the characters that make up the encoding for one note. For example, `# c` would not be recognized because of the space between the `#` and the `c`. Normally, you should put in spaces where it helps you to see the various items in a bar, and wherever one space is allowed, you may put as many as you like.

You may also start a new line in the input wherever a space is allowed, for example, between notes, or between text strings and notes. Most people try not to have a line break in the middle of the notes of a bar, as this makes the file easier to read.

When you start entering longer pieces, you may find it helpful to annotate the input file to make it easier to find your way around it. PMW recognizes the character `@` as a ‘comment character’ – anything on an input line that follows `@` is completely ignored. So, for example, you could have a line such as

```
@ This is the pedal part
```

at the start of a stave. It is also a good idea to put a bar number in the input at the end of each input line, like this:

```
g g a | f. g- a | @2
b b c′ | b. a- g | @4
a g f | G. | @6
```

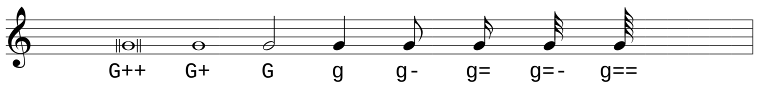
We have now covered everything in the National Anthem example. In the following chapters we will introduce other features of the PMW encoding, but without showing the complete file every time. It particular, the **[stave]** and **[endstave]** directives will normally be omitted.

6. More about notes

This chapter describes some more common facilities used when printing notes.

6.1 Note types

PMW can handle eight different kinds of note, from breves to hemi-demi-semiquavers. The encoding for crotchets, quavers, and minims was introduced in the previous chapter. For notes longer than a minim the `+` character is used to double the duration, and for those shorter than a quaver, the character = is used for ‘two flags’. The complete set is as follows:



6.2 Rests

Rests are specified in the same way as notes, but using the letter R instead of a note letter. The length of the rest is indicated by the case of the letter and following plus, minus, or equals characters, exactly as for notes.

There is one additional character that can follow the letter R, and that is an exclamation mark. This indicates that the rest is equal to the bar length, whatever the time signature may be.

6.3 Repeated rest bars

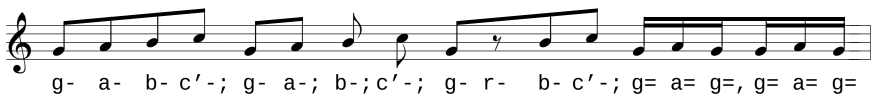
A whole bar rest can be repeated any number of times by putting a number in square brackets before the rest. For example,

```
[24] R! |
```

is the code for 24 bars’ rest. In fact, this kind of repetition is not confined to rest bars; it can be used to repeat any single bar.

6.4 Beams

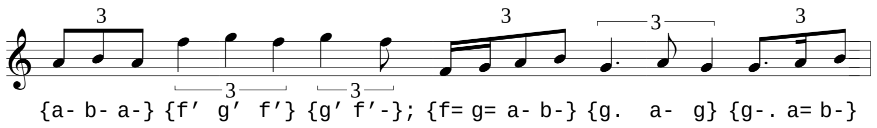
Notes that are shorter than a crotchet are automatically beamed together, unless they are separated by a beam breaking character. A semicolon breaks the beaming completely, while a comma breaks all but the outermost beam. Beams carry on across rests that are shorter than a crotchet, but they are always broken at the end of a bar, unless a continuation over the bar line is explicitly requested (see chapter 42.)


g- a- b- c′-; g- a-; b-;c′-; g- r- b- c′-; g= a= g=, g= a= g=

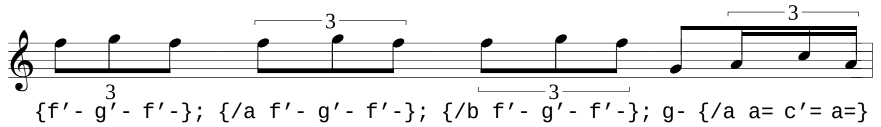
There must not be any space between the last note of a beam and the breaking character (semicolon or comma), but there can be spaces (and other items of data) between the notes themselves.

6.5 Triplets

Triplets are encoded by enclosing a set of notes in curly brackets. If the notes are beamed, just the number ‘3’ is printed, alongside the beam. Otherwise, a longways ‘bracket’ is drawn:


{a- b- a-} {f′ g′ f′} {g′ f′-}; {f= g= a- b-} {g. a- g} {g-. a= b-}

However, you can change this by putting options after the opening curly bracket. If you put `/a` the ‘3’ will be drawn above the notes, while `/b` forces it below the notes. In both cases the longways bracket will also be drawn.



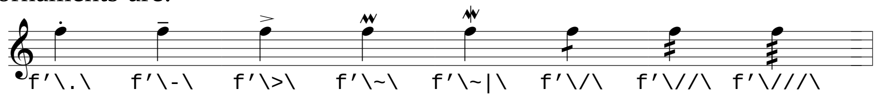
```
{f′- g′- f′-}; {/a f′- g′- f′-}; {/b f′- g′- f′-}; g- {/a a= c′= a=}
```

The last set of notes shows that triplets are beamed onto adjoining notes unless a beam breaking character is present.

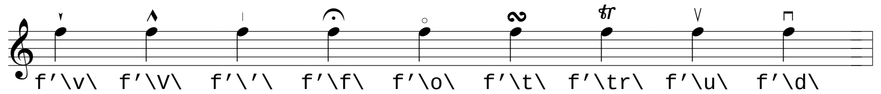
PMW supports other irregular note groupings as well as triplets, and has several more options for controlling the form and placing of the marking. Details are given in sections 38.125, 38.126, 41.19, and 47.102.

6.6 Accents and ornaments

The coding for accents and ornaments is always placed between two backslash characters immediately following a note. For example, a note with a staccato dot is followed by `\.\`. The most common accents and ornaments are:



```
f′\.\   f′\-\   f′\>\   f′\~\   f′\~|\   f′\/\   f′\//\  f′\///\
```



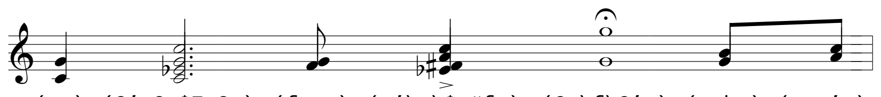
```
f′\v\  f′\V\  f′\'\  f′\f\  f′\o\  f′\t\  f′\tr\  f′\u\  f′\d\
```

The codes `\v\` and `\V\` are used for small and large ‘vertical wedge’ accents because of the similarity of shape, though the accents themselves may be the opposite way up to the coding letters, depending on whether they appear above or below the note. The other mark which looks like the letter V is a string ‘up bow’ marking, which is why `\u\` is used to represent it.

Other controlling options are also given between the same pair of backslash characters, for example, to force the stem of a note to point upwards or downwards, the encodings `\su\` or `\sd\` are used, respectively. When there is more than one item between the backslashes, spaces may be used to separate them. Details of all the various options are given in several sections from 41.13 onwards.

6.7 Chords

Chords in which all the notes are the same length are encoded by enclosing a number of separate notes in round brackets. If the chord has an accent, or any other special option, this must be given with the first note in the brackets. The notes can be given in any order.



```
(cg) (C′.G.$E.C.) (f-g-) (c′\>\$e#fa) (G+\f\G′+) (g-b-) (a-c′-)
```

7. Bar lengths and bar numbers

7.1 Bar lengths

PMW checks that the notes given for a bar match the current time signature, and generates an error message if they do not. However, there are times when this checking needs to be disabled.

For a piece that has variable-length bars without time signatures, or indeed for printing the kind of examples that appear in this manual, the checking can be entirely suppressed by using the heading directive **nocheck**. However, if there is more than one stave, each one must still contain the same total note length for each bar.

The length check can also be disabled for a single bar only. This is done by using the **[nocheck]** stave directive in the bar concerned, in each stave. The most common occurrence of this is at the start or end of a piece where there is an incomplete bar. For example,

```
time 3/4
[stave 1 treble 1]
[nocheck] g | c’fg |
```

7.2 Bar numbers

The **barnumbers** heading directive is used to request the printing of bar numbers above the top stave of systems. There are several options which control where these appear and in what form.

```
barnumbers line
```

causes PMW to print a bar number at the start of each line of music, except the first. Alternatively,

```
barnumbers 10
```

causes PMW to print a bar number every 10 bars. You can choose any number you like; specifying 1 causes a number to be printed on every bar. If the word ‘boxed’ appears after the directive name, the numbers are enclosed in a rectangular box. You may also specify the point size of the font after the word ‘line’ or the count. Thus,

```
barnumbers boxed line 9
```

requests bar numbers at the start of each line, in boxes, using a 9-point font. The default font size is 10 points. Finally, you can specify the font to be used for printing the number:

```
barnumbers 5 italic
```

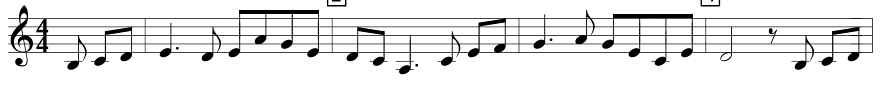
requests bar numbers in italic, every five bars, using the default font size of 10 points.

7.3 Bar counting

When a piece starts with an incomplete bar, it is conventional not to count it for bar-numbering purposes. Bar number 1 is normally the first complete bar of the piece. PMW does not do this automatically, but it does have the **[nocount]** stave directive, which causes a bar not to be counted for numbering purposes. This can be used anywhere in the piece, not only in the first bar. Chapter 23 explains how PMW identifies uncounted bars if it needs to refer to them, for example, in an error message.

This example shows the use of the directives that have been described in this chapter:

```
barnumbers boxed 2 italic
time 4/4
[stave 1 treble 1]
[nocount nocheck]
b`-; c-d- | e.d-; e-a-g-e- | d-c-a`.c-; e-f- | @2
g. a-; g-e-c-e- | Dr-; b`-; c-d- | @4
```



There are further options for forcing or suppressing individual bar numbers, and for moving them. For details, see the **[barnumber]** stave directive in the reference section.

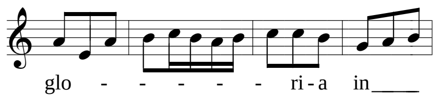
8. More about underlay (lyrics)

PMW supports overlay (words printed above the stave) as well as underlay, though this is much less common. To avoid too many repetitions of ‘underlay or overlay’, this manual mainly describes underlay, on the understanding that all the facilities are also available for overlay as well.

8.1 Multi-note syllables

In the National Anthem example in chapter 5, each syllable of the underlay was associated with just one note. When this is not the case, equals characters are used to continue a syllable over as many notes as necessary. For example,

```
"glo-========ri-a in=="
a-e-a- | b-c′=b=a=b= | c′-c′-b- | g-a-b- |
```



If the continued syllable is not the last one in a word, the equals characters are input following the hyphen. PMW prints a string of hyphens or an extender line, as appropriate, depending on whether the syllable is at the end of a word or not.

PMW does not treat tied notes specially when distributing underlaid syllables to notes, and so an equals character must be used when a syllable is associated with a tied note.

An underlay string must be followed by all the notes to which it relates. This includes continued notes that are indicated by equals characters. Consider the following example:

```
"the cat sat=" g- | gg_ |
"on the mat"  ge-f- | gr |
```

This example is not correct, because the first string provides words for four notes (three syllables plus a continuation), but only three notes follow before the next string. If, as in this example, you start another underlay string before the previous one is all used up, the second string is treated as a second verse.

8.2 Special characters and font changes

The computer keyboard does not contain all the characters that are needed for printing underlay, and there is often a requirement to use different fonts (for example, italic). To cope with both of these, PMW treats the backslash character specially if it is found in a string. (This in fact applies to all strings, not just underlay.) Backslash is known as the ‘escape character’ because it allows an escape from the string in order to give some control information.

Characters following a backslash are interpreted specially. There are a number of ‘escape sequences’ which allow you to specify characters that are not on the keyboard. For example,

`\a′`	prints á
`` \a` ``	prints à
`\a.`	prints ä
`\a^`	prints â
`\ss`	prints ß

See chapter 50 (*Syntax summary*) for a complete list. Changes of font are specified by giving a two-letter font code between a pair of backslashes:

`\it\`	change to *italic*
`\rm\`	change to roman
`\bf\`	change to **bold face**
`\bi\`	change to ***bold italic***

For example, the input string `"\it\sch\o.ner"` prints as *schöner*.

8.3 Spacing

Within a bar, PMW ensures that the syllables of the underlay text do not crash into each other, by spreading out the notes if necessary. Sometimes you may want to make additional adjustments to the spacing. The **[space]** directive is used to insert additional space between notes. The units used for space in PMW are *printers' points*, of which there are 72 to the inch. For example, the input

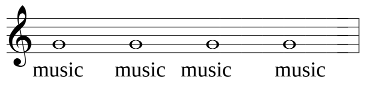
```
  a [space 7] b
```

ensures that the two notes are 7 points (about 0.1") further apart than they would otherwise be. Any underlay that is attached to the notes is also moved appropriately.

There are also two facilities for altering the position of an underlay syllable relative to its note. The character #, if it appears in an underlay string, prints as a space, but is treated as part of a syllable. Since syllables are centred on their notes, putting # characters at the start of a syllable moves it to the right, while putting them at the end moves it left.

Finally, if the character ^ appears in an underlay syllable, only those characters to the left of it are used for finding the centre of the string; the character itself does not print. This example shows the use of # and ^ affects the positioning of syllables:

```
  "music ##music music## mu^sic"
  G+ G+ G+ G+
```



9. Other kinds of text

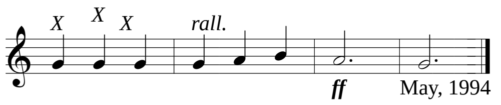
Text strings that are not part of the underlay are normally followed by one of the options `/a` or `/b`, indicating that the string is to be printed above or below the stave, respectively. (If **[text underlay]** has not been set for the stave, then unqualified strings are treated as if `/b` were present.)

Such strings are normally aligned so that they start at the position of the following note, or at the bar line if there are no subsequent notes in the bar. However, if the option `/e` is given, the string is aligned so as to end at the subsequent note or bar line.

The position of any string can be adjusted by following it by one or more of the options `/u` (up), `/d` (down), `/l` (left), or `/r` (right) and a number, which is a distance in printers' points.

The initial font for non-underlay strings is italic, but the escape sequences described above can be used to change it as necessary. Here are some examples:

```
"X"/a g "X"/a/u4 g "X"/a/l6 g |
"rall."/a gab | "\bi\ff"/b A. |
G. "\rm\May, 1994"/b/e |
```



Musical characters (such as notes) are available for use in strings, and there are a number of 'escape sequences' for the most common cases. For example,

`\*m\`	prints a minim
`\*c\`	prints a crotchet

These are most useful in strings of the form `"\*c\ \rm\= 45"`, which prints as

♩ = 45

Rehearsal marks are a special kind of string which are coded by placing the string in square brackets:

```
["A"]
```

PMW prints such strings in a fairly large font, enclosed in a rectangular box; there are options to control this if necessary.

10. Macros

A *macro* is a concept found in computer programming languages and in some kinds of wordprocessing systems. The idea is very simple: whenever there is a sequence of input characters that are going to be repeated several times in a file, they are given a name. Then simply giving the name calls up the required characters.

There are several advantages in using macros. Not only does it save typing, but it also guarantees that the same input string is used every time, thus ensuring consistency. In addition, if a change needs to be made to the string, it only has to be done once.

Simple macros are introduced here because they are frequently used for text strings that are repeated in a piece – typically strings such as *mf*, *ff*, etc. Consider the following input line:

```
  *define mf "\it\m\bi\f"/b
```

This is a directive which defines a macro whose name is **mf**. It is an example of a *preprocessing* directive, which is a third kind of directive, in addition to heading directives and stave directives.

Preprocessing directives may occur anywhere in a PMW input file. They always occupy a complete input line by themselves, and are identified by starting with an asterisk. The ***define** directive must be followed by the name of the macro being defined. The replacement text for the macro consists of the rest of the input line, which may be empty.

After the definition above has been processed, an occurrence of the characters `&mf` anywhere in the input is replaced by the text

```
  "\it\m\bi\f"/b
```

There must not be any space between the introductory & character and the name of the macro that is being inserted.

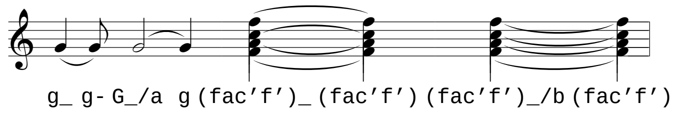
This particular example specifies a text item for the string *m**f***, where the *m* is printed in italic and the ***f*** in bold italic, as is commonly done. It also specifies that the string is to be printed below the stave. If other options are needed for instances of the string, they can be added after the macro call; in particular, adding `/a` will cause the text to be printed above the stave, as when both `/b` and `/a` appear, the rightmost one is used. Here are some examples of possible uses of this macro:

```
  &mf abc | &mf/a efg | cg &mf/d6 d |
```

Macros can be used for any string of input characters; their use is not confined to text items. A full description of all the macro facilities is given in chapter 22.

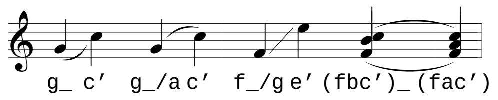
11. Ties, slurs, and glissandos

Single notes and chords are tied together by entering an underscore character at the end of the first note, or following the closing bracket of the first chord. For single notes, ties are normally drawn on the opposite side of the noteheads from the stems, but can be followed by `/a` or `/b` to force them above or below the noteheads. These options can also be used for chords.



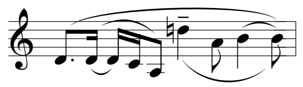
When two single notes of different pitches are connected by a slur, the same notation (an underscore) can be used. However, for chords, the **[slur]** directive (see below) is required to draw slurs, because if two chords are joined by an underscore, the notes in each that are of the same pitch are joined by a tie mark, any other notes being left alone.

The underscore is also used for glissandos between single notes; following it with `/g` causes a glissando line to be drawn instead of a short slur.



For slurs involving chords or more than two single notes, the **[slur]** and **[endslur]** (or **[es]**) directives are used. The notes which are to be covered by the slur appear between them. The slur is drawn above the notes unless `/b` is given.

```
[slur]
d-. [slur] d=_; d=c=a-; [es]
[slur/b] %d'\-\ a-\sd\ b_b- [es]
[endslur]
```

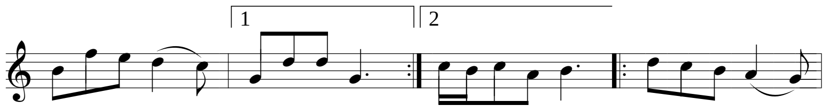


This example shows that slurs can be ‘nested’ inside one another if necessary, each **[endslur]** directive relating to the most recent **[slur]**. There are options for handling more complicated cases, and there are also options for adjusting the positions and shapes of slurs. See section 47.75 for details.

12. Repeats

Conventional musical repeat marks are encoded using the input strings `(:` and `:)` which may occur in the middle of a bar as well as at the start or end. When there is a first time and a second time ending, the directives **[1st]** and **[2nd]** are used to indicate it, with the directive **[all]** marking the bar where all the endings are complete.

```
b-f’-e’-; d’_c’- | [1st] g-d’-d’- g. :) |
[2nd] c’=b=c’-a- b. | [all] (: d’-c’-b- a_g- |
```



The **[all]** directive is not used when the second time bar is the final bar of a piece.

Instructions such as *Da capo* are given as text strings, and the music font contains the two conventional signs used in conjunction with *Dal segno*. They correspond to the letters c and d, and can be printed in text strings as follows:

`"\mf\c"`	prints 𝄌
`"\mf\d"`	prints 𝄋

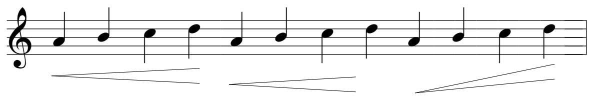
The escape sequence `\mf\` changes to the music font, full details of which are given in chapter 48.

13. Hairpins

Crescendo and decrescendo ‘hairpins’ are coded using the characters `<` and `>` in pairs. The hairpin starts at the note following `<` or `>` and ends at the note before the next one. Hairpins are drawn below the stave by default, but the directive **[hairpins above**] can be used to cause them to be drawn above.

Either end of a hairpin can be moved by following the angle bracket with `/u` (up), `/d` (down), `/l` (left), or `/r` (right), and a number, which gives a distance in points. Any up or down movements specified at the start of a hairpin apply to the whole hairpin, but any that are specified at the end apply only to the end – by this means, sloping hairpins can be drawn. For example,

```
  < abc’d’ < </d4 abc’d’ </l10 </r4/d8 abc’d’ </u10
```



If the beginning or ending character is followed by `/h`, then the corresponding end of the hairpin is moved to the right to be halfway between the note where it would otherwise be, and the next note, or the bar line if there are no more notes in the bar. Additional left and right movements can be specified, and are relative to this point.

There are some other options for changing the position and form of hairpins. The full description is in section 40.3.

14. Staves and systems

This chapter gives some introductory information about setting up staves and systems. The reference chapters describe additional facilities for use in complicated cases.

14.1 Stave spacing

The default spacing between staves is 44 points. This is the distance between the bottom line of one stave and the bottom line of the one below it. The **stavespacing** heading directive is used to alter this. It is followed by a list of stave numbers and spacings, each pair being separated by a slash. The spacings are the distances to the stave below. For example,

```
stavespacing 2/60 4/54
```

specifies that the spacing between staves 2 and 3 is to be 60 points, while that between 4 and 5 is to be 54 points. The remaining spacings will take the default value of 44 points.

PMW does not make any alterations to stave spacings by itself. However, there is commonly a requirement to make a change in the spacings for one particular system, usually when one stave has particularly high or low notes. This can be done by using the **[sshere]** directive. When this is encountered, it causes the spacing for the current stave to be changed, for the current system only.

A completely new value can be given, but if a number is given preceded by a plus or minus sign, it causes a change in the spacing of that amount. Thus

`[sshere +4]`	increases the spacing by 4 points
`[sshere -2]`	decreases the spacing by 2 points
`[sshere 48]`	sets the spacing to 48 points

14.2 System gap

The distance between systems is called the ‘system gap’, and is set by the **systemgap** heading directive. Again, the default is 44 points. However, since PMW normally puts additional space between systems so that the bottom stave is at the bottom of the page, the system gap value is really a minimum distance between systems. (See the **justify** directive if you want to stop PMW from doing this vertical justification.)

There is an **[sghere]** directive for changing the system gap for a single system, and it works in exactly the same way as **[sshere]**.

14.3 Brackets and braces

By default, PMW joins together the staves that comprise a system with a bracket, as can be seen in the National Anthem example. The other kind of joining sign (used most often for two staves for one instrument) is the brace, which is a large version of the { character.

There are two heading directives, **bracket** and **brace**, which specify which staves are to be joined with each of these signs. Each of these directives is followed by a list of stave ranges. For example,

```
bracket 1-4, 8-11
brace 5-6
```

causes the system to be divided into three sets of staves. Two of the groups, staves 1–4 and 8–11, are each joined by a bracket, while staves 5–6 are joined by a brace.

If you don’t want any staves at all to be bracketed, as might be the case when setting a keyboard piece, you need to include the directive

```
bracket
```

with nothing after it, in order to cancel the default setting, which is to bracket all the staves of the system.

14.4 Initial text

At the start of an instrumental piece it is common to print the names of the instruments. This is done in PMW input by giving a string in quotes as part of the **[stave]** directive, for example,

```
  [stave 1 "Clarinet" treble 1]
```

The text can be split up into several lines by including vertical bar characters; each vertical bar causes a line break. For example,

```
  [stave 5 "Horn|in F" treble 1]
```

Options are available for changing the form and layout of this text; for details, see section 47.82.

15. Keyboard staves

Keyboard music is one of the more complicated kinds of music to typeset, especially if it is a reduction of an instrumental score. It is usually a good idea to study the manuscript carefully to decide exactly how it is to be encoded before you start.

A brace is normally used to join the staves of keyboard music, and the name of the instrument is printed mid-way between the two staves. This can by done by adding `/m` to the relevant string.

15.1 Overprinted staves

There are two ways of tackling pieces which have two parts on one stave, with stems pointing in different directions. If most of the piece is like this, then the best approach is to use two different PMW staves, but specify a stave spacing of zero so that the staves print on top of each other. Use can be made of the directives **[stems up]** and **[stems down]** to force the stem directions of all notes. The directives **[ties above]** and **[ties below]** can also be used to force the default direction of all ties.

In the following example, two PMW staves have been used for each printing stave, and the stave spacings have been set accordingly. The spacing after stave 2 has been increased to avoid clashes of stems between the two staves.

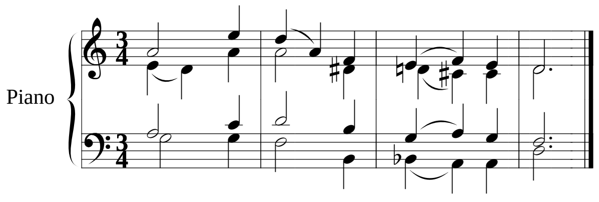
```
time 3/4
bracket
brace 1-4
stavespacing 1/0 2/48 3/0

[stave 1 "Piano"/m treble 1 stems up ties above]
Ae’ | d’_af | e_fe | D. |
[endstave]

[stave 2 treble 1 stems down ties below]
e_da | A#d | [smove 6] %<d_#cc | D. |
[endstave]

[stave 3 bass 0 stems up ties above]
Ac’ | D’b | g_ag | F. |
[endstave]

[stave 4 bass 0 stems down ties below]
[smove 6] Gg | Fb‘ | $b‘_a‘a‘ | D. |
[endstave]
```



In this example there are two additional items which have not yet been explained; both are connected with handling the case when a note on one stave would partially obscure a note on the overprinting stave.

PMW is not clever enough to detect that two notes are going to interfere in this way, so the input must contain explicit instructions to move one of the notes. Consider the very first pair of notes on the bottom stave; they are A and G and so would collide if printed at the same horizontal position. To prevent this, the directive **[smove 6]** has been used. This directive has two effects:

- It moves the following note (in this case, the G) 6 points to the right, without affecting the position of anything else.
- It inserts an additional 6 points of space *after* the next note. That is, everything after the next note is moved 6 points to the right.

Each of these effects can be realized independently, by means of the **[move]** and **[space]** directives; **[smove]** is a composite of the two, provided because they are so frequently required together in this situation.

The second additional item can be seen in the third bar of stave 2. The D natural has been moved to the right by means of **[smove]** but by itself this would have caused its accidental to collide with the E that we are trying to avoid. The **<** character after the percent sign (which is the code for a natural) has the effect of moving the accidental 5 points to the left. This is sufficent to get it clear.

15.2 The [reset] directive

If a printing stave can mostly be encoded using only a single PMW stave, but there are one or two bars where stems in opposite directions are required, the **[reset]** directive can be used. This has the effect of resetting to the beginning of the bar, so that a second set of notes can be specified for the bar. For example, the first bar of the right-hand part in the example above could be encoded on a single PMW stave like this:

```
[stems up] Ae′ [reset] [stems down] e_/b da |
```

This technique is not recommended except for the occasional bar or two.

15.3 Invisible rests

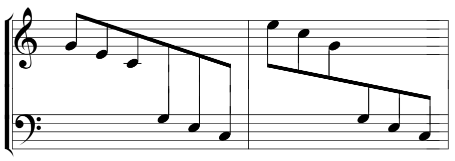
When using overprinting staves for keyboard pieces, it is frequently the case that one ‘part’ does not contain enough notes to fill the bar. The note letter Q can be used to make the bar up to its correct length. This letter (which can be thought of as standing for ‘quiet’) acts exactly like the rest letter R, except that it does not print anything at all. It is often referred to as an ‘invisible rest’.

15.4 Coupled staves

Keyboard music sometimes includes sets of beamed notes which extend over both staves. These can be printed using a technique known as ‘coupling’:

```
[stave 1 treble 1 couple down]
g-e-c-g`-e`-c`- | e′-\sd\c′-g-g`-\sw\e`-c`-
[endstave]

[stave 2 bass 0]
Q! | Q! |
[endstave]
```



The upper stave has been ‘coupled’ downwards to the lower one. When this is done, any note in the upper stave that is lower than middle C is printed on the lower stave. Notice the use of invisible rests on the lower stave to fill the bars without printing anything.

An alternative approach is to use **[couple up]** to get notes from the lower stave that are higher than middle C printed on the upper stave. Simultaneous coupling of two staves in both directions is permitted.

The second bar of this example shows how to get notes printed on both sides of a beam, a facility which is often needed when using coupled staves. The `\sd\` option on the first note of the beam forces its stem to be downwards. Normally, this would mean that all the other notes in the beam would also have downward pointing stems. However, the fourth note has the option `\sw\`, which has the effect of swapping the stem direction for the remaining notes.

For a stem direction swap to work, the two nearest notes have to be fairly far apart, as these two are. If stem swapping is tried in the middle of the first bar of this example, PMW generates an error, because there isn’t room to fit the beam in between the two sets of notes.

The coupling state can be changed as often as necessary in a piece: **[couple off]** turns it off altogether. It only works properly if the upper stave is using the treble clef and the lower one the bass clef, and the distance between them must be a multiple of four points.

16. Heads and feet

We introduced the **heading** directive in the discussion of the National Anthem example in chapter 5. You may have as many **heading** directives as you like at the start of a piece. By default, the first two will be printed in larger type than the remainder. However, you can specify an explicit font size by giving a number before the string:

```
  heading 13 "|Scherzo"
```

specifies a size of 13 points, for example. After printing a heading, the ‘current point’ is moved down the page by a distance equal to the font size, so a second heading after the one above would print 13 points below it. You can control this distance by giving a number after the string:

```
  heading 16 "|Mass" 24
```

specifies a type size of 16 points, and a subsequent space of 24 points.

One special case of this is to specify a distance of zero so that the next heading prints at the same level. This makes it possible to print in different sizes on the same line, for example:

```
  heading 16 "|Piece" 0
  heading 12 "Words: J. Smith||Music: A. Jones" 24
```

As the first heading consists of centred text, while the second has only left-hand and right-hand parts, with nothing in the middle, they won’t overlap.

The **footing** directive is of exactly the same form as **heading**; it specifies what is to be printed at the bottom of the first page. The escape sequence `\c)` is useful in footings; it prints as ©.

Both **heading** and **footing** apply to the first page of a piece only. To print heads and feet on other pages, you must use the **pageheading** and **pagefooting** directives. The **pageheading** directive applies to all pages except the first, while **pagefooting** applies to all pages, unless overridden for the first page by a **footing** directive.

The most common use of these directives is for printing page numbers, either at the top or the bottom of each page. There are three escape sequences for printing page numbers:

`\p\`	prints the current page number
`\pe\`	prints the current page number only if it is even
`\po\`	prints the current page number only if it is odd

A directive such as

```
  pagefooting "|\p\"
```

causes the page number to be printed centrally at the foot of each page (unless there is also a **footing** directive, which then controls what is printed on the first page), while

```
  pageheading "\pe\||\po\"
```

causes page numbers to be printed at the top of each page other than the first, alternately on the left and right. Even numbers are printed on the left, and odd ones on the right.

When heading or footing text contains left-hand and right-hand parts, these line up with the left and right edges of the musical staves. When printing page numbers it is sometimes desirable to have these print outside the normal margins.

The easiest way to do this is to make use of one of the special characters in the music font. These are characters that cause no marks to be made on the page, but which move the current printing position. They are provided for use by PMW when building up complicated shapes from simpler ones, but they can be used for other purposes as well.

Full details of the music font are given in chapter 48. The character of interest here is character number 123, which corresponds to the `{` character in text fonts. It causes a leftwards movement of 0.33 times the font’s size (e.g. 3.3 points for a 10-point font). Consider a heading line such as

```
  pageheading "\mf\{{{{\rm\\pe\||\po\\mf\{{{{"
```

The escape sequence `\mf\` changes to the music font. The string of four `{` characters causes a leftwards movement of the printing position, so that the even page number will be printed to the left of the normal margin (`\rm\` changes back to the roman font).

At the end of the line, the backwards spacing must follow the page number. At first sight it looks odd to end a string with spacing characters, but because this is a right-aligned string that must end at the right-hand margin, the backwards movement has the effect of causing the odd page numbers to print to the right of the normal margin, so that the subsequent leftwards movement brings the current printing point back to the margin.

Another common requirement is to print page numbers higher up the page than PMW normally starts printing. This can be achieved by using a **pageheading** directive with an empty text string and a negative downwards movement. For example,

```
  pageheading "" -10
```

has the effect of moving up the page by 10 points.

17. Page layout

The horizontal length of musical systems can be set by means of the **linelength** directive, and the vertical length of pages by the **pagelength** directive. The default values are equivalent to the directives

```
linelength 480
pagelength 720
```

These are suitable values for printing on A4 paper while leaving fairly generous margins, especially at the sides.

The linelength can be increased to as much as 520 for A4 paper without getting too near the edges. The music is printed centrally on the page, so changing the line length changes both margins symmetrically.

PMW assumes that you are printing on A4 paper, but it can support other paper sizes as well. The **sheetsize** directive can be used to set A3, while the **sheetwidth** and **sheetdepth** directives can be used to set the dimensions of the paper independently.

The value given for the page length sets the space used for headings and for printing musical systems. However, it does not include the space for footings, which are always printed starting 20 points below the page length distance down the page.

By default, PMW fills up each system with as many bars as it can within the given line length, and then fills up each page with as many systems as it can. Sometimes this means that the music takes up more or fewer pages than required, or does not end tidily at the end of a page.

If you know in advance what layout is required, you can use the **layout** heading directive to specify how many bars there are in each system and how many systems there are on each page. Otherwise, when using the default filling mechanism, the following stave directives can be used to influence the layout:

- The **[newline]** directive causes PMW to start a new line of music (a new system) with the bar in which it appears. It need appear only in one stave.
- The **[newpage]** directive causes PMW to start a new page with the system in which it appears. It need appear only in one stave.
- The **notespacing** directive can be used to spread out or to compress the music.

We introduced the **notespacing** directive in the National Anthem example; it causes the spacing between notes to be multiplied by a given factor. For example,

```
notespacing *0.92
```

reduces all the distances by a factor of 0.92. However much you reduce the notespacing, PMW will not allow notes to print on top of each other. Quite small changes of note spacing can sometimes make dramatic changes to the layout of a piece, by causing the assignment of bars to systems to change. At other times, for example when bars are very long, a large change might be needed to have any effect.

Occasionally it is helpful to change the notespacing for part of a piece only. This can be done by using the **[notespacing]** directive (abbreviation **[ns]**). This should always be given at the start of a bar; it then affects the current bar and subsequent ones. If it is given without a value, the spacing is reset to what it was at the start of the piece. Therefore, to reduce the spacing for one bar only, one might have:

```
[ns *0.8] g=a=b=g=; b=a=g=b= |
[ns] D |
```

This should be given in the first stave because PMW processes the staves in order, for each bar, and any previous staves would be processed using the old value. That is also why resetting the value should be done in the next bar; if **[ns]** were at the end of the first bar, the reset values would be used for the following staves.

Another way of fitting a piece onto a given number of pages is to change the magnification, as described in the next chapter.

18. Magnification

The standard size of music printed by PMW has a distance of 4 points between stave lines. The **magnification** heading directive can be used to cause it to print bigger or smaller staves. This should not be confused with the zoom factor, which is an additional magnification setting used when viewing music on the screen. The zoom factor applies only to the screen display, whereas the magnification applies to all forms of output. For example,

```
  magnification 1.5
```

has the effect of increasing the gap between stave lines to 6 points, while

```
  magnification 0.75
```

reduces it to 3 points. Everything else that is printed is magnified or reduced in proportion. There are no restrictions on the values that can be given for the magnification.

When a magnification other than 1.0 is set, the distances given in PMW directives are all magnified too. This means that if a vertical distance is specified as 4 points, it is always equal to the distance between stave lines. Thus, changing the magnification does not require changes to the musical data.

However, the values given for the **linelength** and **pagelength** directives are *not* magnified or reduced. They specify the real dimensions of the page, and so do not have to be changed if the magnification is changed.

19. Extracting parts from a score

When a score file has been created, individual parts can be extracted by using the **-s** command line option, as described in section 4. For example, if the input were a string quartet, selecting stave 2 would cause just the second violin part to be output.

Usually, you will want to make some changes when a part is printed. At the very least, the headings will probably be different, and you may want to print cue notes in parts but not in the score. You may also want to print parts at a larger magnification, and force page or line breaks at particular places.

This is where the PMW *conditional directives* come in. These are preprocessing directives that allow you to skip parts of the input file under certain conditions. For example, the heading portion of a file might contain

```
*if score
  magnification 0.9
*else
  magnification 1.3
*fi
```

Because they are preprocessing directives, ***if**, ***else**, and ***fi** must appear each on a line by itself. In the example above, ***if** tests to see whether a full score is being printed, and if so, sets the magnification to 0.9. Otherwise it sets it to 1.3. PMW considers that a score is being printed if no staves are selected by the **-s** command line option.

The ***if** directive can also test for individual stave selections, and this is the way to print appropriate headings:

```
*if stave 1
  heading "Violin I"
*fi
*if stave 2
  heading "Violin II"
*fi
*if stave 3
  heading "Viola"
*fi
*if stave 4
  heading "Violoncello"
*fi
```

The ‘stave’ test succeeds if the given stave, and only the given stave, is selected, but it is possible to give a list or range of staves (and to use the plural ‘staves’):

```
*if staves 1-2
  heading "Violins"
*fi
```

Finally, the ***if** directive can be used to test for an arbitrary *format name* defined by the user. You specify the format using the **-f** option in the PMW command line. It can be any word that you like. For example, if you wanted to print out the string parts from a score, instead of explicitly specifying the stave numbers each time, you could specify ‘strings’ as the format, and use input such as

```
*if strings
  selectstaves 4-9
*fi
```

(The **selectstaves** directive has the same effect as selecting staves in the **Staves** options item, provided it precedes any tests on the stave selection.) This facility can be put to many other uses for varying the format of the output.

It is not necessary to indent the directives that appear between ***if** and ***fi**, but it helps make the input more readable.

Because these conditional directives are preprocessing directives, they can appear anywhere in a PMW file, not just in the heading portion. Here is an example that shows how to print rest bars in a score, but cue bars in a part:

```
[stave 6 "Trumpet" treble 1]
[20] R! |
*if score
[2] R! |
*else
"(flute)"/a [cue] g’f’e’ | [cue] C’. |
*fi
```

The **[cue]** directive specifies that the remaining notes in the bar are to be printed at the cue note size.

20. Reference section

The preceding chapters describe the basic features of the PMW music encoding in an introductory manner, in an order suitable for this purpose. Using only the material therein, you should be able to typeset a wide variety of music.

However, there are many special-purpose features which have not yet been covered. The remainder of this book is written in the form of a reference manual. It gives a complete description of PMW input files, in an order which is more suitable for looking up individual features than for straight reading.

When describing the syntax of directives, use is often made of one or more italic words in angle brackets, for example,

```
  tripletfont <fontsize> <name>
```

What this means is that the items in angle brackets must be replaced by some specific instance of what they describe (in this case, values for the font size and the font name), when the directive is used. An example of the use of **tripletfont** is

```
  tripletfont 8 italic
```

Frequently, when the required value is a single number, <*n*> or some other single letter is used. In the example above, <*fontsize*> was replaced by a single number; however, more complicated ways of specifying the size of a font are possible, as described in chapter 32 (*Font sizes, aspect ratios and shearing*).

The next two chapters describe the form of PMW input files, and the preprocessing directives. Then follow a number of chapters giving general information, with references to particular directives. Complete descriptions of the directives themselves are not given here; they may be found, in alphabetical order, in the *Heading directives* and *Stave directives* (chapters 38 and 47). The chapters in between, starting with *Stave data* (chapter 39) contain the specification of all items other than directives that may appear as part of a stave's data.

21. Format of PMW files

A file containing input for PMW is an ordinary text file that can be constructed using any available text editor or wordprocessor.

There is only one circumstance in which the use of a space is necessary, and that is to delimit items on a line when there would otherwise be ambiguity, for example, when a word is followed by another word. However, spaces are allowed between items, and can be profitably used to increase the readability of the file. Other than in quoted strings, a sequence of spaces is equivalent to one space.

The character `@` is a comment character; if it appears outside a quoted string the rest of the input line is ignored. This provides a way of annotating PMW input files. The first line of a file is very often something like

```
@ Created by Christopher Columbus, October 1492
```

Within the file, line breaks are equivalent to spaces, except in three cases:

- When a line contains a comment (see above), the comment continues to the end of the line.
- Preprocessing directives always take up a complete line of their own (see the next chapter) and may not continue onto subsequent lines.
- When a directive takes a number of numerical arguments, these can be separated by commas and/or spaces. However, if the list of numbers continues onto the next line, the final one on the first line must be followed by a comma.

The character `&` is an insert character and is recognized at any point in the file. It must be followed by the name of a previously-defined macro, the contents of which are inserted at that point – for details, see the description of the ***define** preprocessing directive in section 22.2. If a literal `&` character is actually required in the input, it must be entered as `&&`.

PMW is case-sensitive. That is, it distinguishes between capital (upper case) letters and small (lower case) letters. The only places where case does not matter are

- The names of directives (`KEY` is equivalent to `key`),
- The names of key signatures (`E$M` is equivalent to `e$m`),
- The 'common' and 'cut' (alla breve) time signatures (`C` and `A` are equivalent to `c` and `a`).
- Format words used to specify alternative forms of output.
- Words following the ***if** preprocessing directive.

A PMW file starts off with a number of items collectively known as the *heading*. These give information which applies to the whole piece of music, such as its title, and they may also change default parameters which control the final layout on the page (for example, the line length). The heading is terminated by the first opening square bracket in the file, and may be completely empty.

Following the heading there is information for each stave, beginning with an item of the form

`[stave <`*n*`> <`*additional data*`>]`

and ending with an item of the form

```
[endstave]
```

A description of the **[stave]** directive is given in section 47.82. There may be up to 63 staves, numbered from 1 to 63. They are output in numerical order down the page. If a stave numbered *n* is present, then all the staves with numbers lower than *n* are automatically supplied as empty staves if they do not appear in the input. For example, if only staves 2 and 4 are given, empty staves 1 and 3 are manufactured.

In addition to staves numbered 1–63, data for stave 0 may also be specified. This is treated specially and is not a normal stave. It is always selected for printing, and by default it overprints the topmost stave of each system. Any notes, key signatures, time signatures and clefs specified are not printed, but text and other items are. Stave 0 can be used to print tempo descriptions and other marks above the topmost stave, whichever staves are selected for printing.

It is also possible, by means of the **copyzero** directive, to specify that stave zero should overprint more than one stave in each system. This can be useful in large scores where it is helpful to print things like tempo directions and rehearsal letters at several levels. Further details of stave 0 are given in chapter 35.

A PMW input file need not contain any stave data; in this circumstance the only output will be the headings and footings, on a single page. This is a slightly eccentric way of printing concert posters. As the heading section is also optional, it follows that a completely empty file is also valid; its output is one blank page.

If the headings fill up a lot of the page, there may be insufficient room for the first system of music, which is therefore printed on the next page. This gives a way of producing a title page followed by pages of music, all from a single input file.

A PMW file may contain more than one musical movement, that is, the piece may be split up into several independent sections, each with its own title. It is worth doing this if there is some possibility of not having to start a new page for each movement, which is sometimes the case when instrumental parts are being printed. If you know that each movement will always start on a new page, it is usually best to keep each movement in a separate file, unless there are only a few, short movements.

The start of a new movement is indicated by an item of the form

```
[newmovement]
```

following any stave's information. After this there may appear a new set of heading items, followed by the staves for the new movement.

The general format of a complete PMW input file is therefore as follows:

<*Heading information*>
<*First stave of first movement*>
<*Second stave of first movement*>
`...`
<*Last stave of first movement*>
`[newmovement]`
<*Supplementary heading information*>
<*First stave of second movement*>
<*Second stave of second movement*>
`...`
etc.

The contents of headings and stave data are separately described in detail in subsequent chapters.

22. Preprocessing directives

Preprocessing directives may occur at any point in an input file; in the heading, in the middle of a stave's data, or between staves. Most of them have the effect of modifying the subsequent input text in some way. They are called preprocessing directives because they take effect before any other processing of the input lines.

A preprocessing directive must be at the start of a line, preceded by an asterisk (spaces before the asterisk are permitted), and it occupies the whole line.

22.1 *Comment

This directive causes the remainder of the input line to be written to the PMW verification output (the standard error stream). It may be useful for outputting reminders to the user.

22.2 *Define

The ***define** directive is used to define *macros*. A macro is a name for a string of characters; usually the name is much shorter and easier to type than the string it represents. The format of ***define** for a simple macro is

```
*define <name> <rest of line>
```

The rest of the input line, starting from the first non-space after the name, is remembered and associated with <*name*>, which must consist of a sequence of letters and digits. It may start with a letter or a digit, so names such as '8va' can be used, and upper and lower case letters are considered different in macro names. The rest of the line may consist of no characters at all, in which case <*name*> is associated with an empty string.

If there is a comment character @ on the input line, outside double quote marks, it terminates the string which is being defined. That is, comments are permitted on ***define** directives, provided they contain either no quotes, or matched pairs of quotes. If you use macros to generate partial strings, with unmatched quotes in the defining lines, then the use of the @ character should be avoided.

The character & is used as a flag character to trigger the substitution of the remembered text. Wherever it appears in the input (except when it follows the @ comment character), it must be followed by a name that has previously been set up by ***define**. The sequence &<*name*> in the input is replaced by the remembered text. If a genuine ampersand is required in the input, it must be input as &&.

To avoid ambiguity, a semicolon character can optionally be used to terminate the name in a substitution, for example, if the immediately following character is a letter or digit. It is removed from the text when the substitution takes place. If a semicolon is required in the input following a substitution, then two semicolons should be entered.

If an undefined name is encountered following &, PMW issues an error message, and substitutes an empty string. It is possible to test whether a name has been defined or not (see below).

An example of the use of a simple macro is given in chapter 10.

22.3 Macros with arguments

There are times when it is useful to be able to vary the text which is inserted by a macro. The word *argument* is used in mathematics and computer programming to describe values that are passed to functions and macros on each call, and that term is adopted here.

The use of arguments is best explained by an example. Suppose a piece of music has many 'hanging ties', that is, ties which extend to the right of a note but which end in mid-air rather than on the next note. The input to achieve this (for a given note) could be

```
[slur/rr15] g' [es]
```

A macro with an argument can be defined in order to shorten the input, as follows:

```
*define hang()  [slur/rr15] &&1 [es]
```

The brackets after the macro name tell PMW that this macro has one or more arguments, and the characters `&&1` in the replacement text indicate the place where the first argument is to be inserted. This macro can be used for many different notes, for example:

```
&hang(g') &hang(B++) &hang(e'-)
```

In each case, the text that forms the argument is substituted into the replacement text where `&&1` appears. The argument is supplied immediately after the macro name, enclosed in round brackets. Any number of arguments may be used. The example macro could be extended to make use of a second argument as follows:

```
*define hang()  [slur/rr15&&2] &&1 [es]
```

Now it is possible to use a second argument to specify that the tie is to be below the note, for example:

```
&hang(g,/b)
```

As this example shows, arguments are separated from each other by commas. All the characters between the brackets and commas form part of the argument; if, for example, there is a space after a comma, it forms part of the next argument. Arguments may contain no characters; this is not an error.

An argument can be inserted many times in the replacement text. If the following character is a digit, the argument number must be followed by a semicolon as a terminator. This means that if the following character is a semicolon, two semicolons are required.

There are times when it is necessary to include commas and brackets as part of an argument. The following rules make this possible:

- No special action is necessary if an argument contains matched brackets. Within the brackets, commas are not recognized as terminating the argument. For example:

  ```
  &hang((fac'))
  ```

- To include an unmatched bracket or a comma that is not within brackets, the character & is used as an escape character. For example, if a note with a bracketted accidental is used with the `hang` macro, the input is

  ```
  &hang(#&)c')
  ```

  Without the & preceding it, the accidental bracket would be interpreted as terminating the argument list.

- If an argument contains matched double quote characters, commas and brackets (matched or unmatched) within the quotes are not treated specially. An unmatched double quote character can be included by escaping it with &.

In fact, the appearance of & before a non-alphanumeric character anywhere in a macro argument always causes the next character to be taken literally, whatever it is. To include an & character itself within the text of an argument, it must be specified as &&.

Macro arguments may contain references to other macros, to any arbitrary depth. An & followed by an alphanumeric character in an argument is interpreted as a nested macro reference. It is also possible to have macro substitions in the definition of another macro.

If a macro which is defined with argument substitutions is called without arguments, or with an insufficient number, nothing is substituted for those that are not supplied, unless defaults have been provided.

Default values for arguments are supplied as an argument list in the macro definition. For example:

```
*define hang(g',/a)  [slur/rr15&&2] &&1 [es]
```

The rules for the default argument list are the same as for argument lists when calling macros, except that, if & is required to escape a character, it must be written twice. This is necessary because macro definition lines are themselves subject to scanning for macro substitution before they are interpreted. For example:

```
*define hang(#&&)g') [slur/rr15] &&1 [es]
```

It follows that, if an & character is actually required in a default argument, &&&& must be typed.

22.4 *Include

This directive can be used to include one file within another. For example, the same standard heading file could be used with a number of different pieces or movements that require the same style. The name of the included file is given in quotes, for example,

```
*include "std-setup"
```

If the name does not start with a slash, it is taken relative to the directory in which the main PMW input file is stored if PMW was called with a file name to specify the input. If the standard input is being used, a non-absolute path name is taken relative to the current directory. Included files may be nested. That is, an included file may contain further ***include** directives.

22.5 Conditional preprocessing directives

The conditional preprocessing directives are ***if**, ***else**, and ***fi**. Their purpose is to arrange for certain sections of the input file to be included or omitted under certain circumstances.

The ***if** directive is followed by a condition, which consists of a word, possibly followed by more data. It the condition is met, then subsequent lines of the input, up to ***else** or ***fi**, are processed. If the condition is not met, these lines are skipped. When ***else** is used to terminate the block of lines after ***if**, then the lines between it and a subsequent ***fi** are obeyed or skipped depending on whether the first block of lines was skipped or obeyed. An example will make this clearer:

```
*if score
  magnification 0.9
*else
  magnification 1.2
*fi
```

Each ***if** must have a matching ***fi**, but there need not be an ***else** between them. It is permitted to nest conditional directives, that is, a complete sequence of ***if** → ***fi** may occur within another. This provides a way of testing that a number of conditions are all true.

The word ‘or’ can be used to test whether either one of two (or more) conditions is true, for example:

```
*if staves 1-3 or stave 7
*if violin or viola
```

If a condition is preceded by the word ‘not’, then the sense of the condition is negated. For example,

```
*if not score
  magnification 1.2
*fi
```

We now describe the various conditions that can be tested using ***if**:

If the word that follows ***if** or ***if not** is ‘score’, then the condition is true only if no stave selection option is specified on the PMW command line, and the **selectstave** directive has not been used earlier in the file.

If the word is ‘part’ then the condition is true if and only if a stave selection option is given on the command line, or via the **selectstave** directive earlier in the file.

If the word is ‘stave’ (or ‘staff’ or ‘staves’), it must be followed by a list of staves. In this case, the condition is true if the listed staves, *and no others*, are selected. The intended use is for varying the headings of the piece when different combinations of staves are selected for printing.

If the word is ‘undef’ then this must be followed by a name, and the condition is true only if the given name has not yet been defined as a macro using the ***define** directive.

If the word is ‘format’, the condition is true if the **-f** command line option has been used to specify a named format, and false otherwise.

If the word following ***if** is not one of the above, then the condition is false, unless the **-f** command line option was used to specify the same word that follows ***if** or ***if not** as a format name. The comparison of the words is done in a case-independent manner.

Some examples of the use of the conditional preprocessing directives are as follows:

```
*if score              @ print full score reduced
  magnification 0.8
*else                  @ print part(s) magnified
  magnification 1.1
  systemgap 60
*fi

*if stave  1
  heading "Flute"
*fi
*if staves 2-3
  heading "Violins"
*fi

*if undef topspace
  *define topspace 20
*fi

*if large
  magnification 1.5
*fi
```

The last example would be triggered by including

```
-f large
```

in the PMW command line. Only one format word can be set at a time in this way. It must begin with a letter and consist of letters and digits only.

23. Identification and counting of bars

PMW identifies bars in its messages using the same number as would be printed as a bar number on the music. This applies both to error messages and to the bar numbers which are used to verify the layout of systems on the page.

This change makes it easy to associate messages with the actual bars of the music, but it requires some special notation for identifying bars containing the **[nocount]** directive.

If the first bar of a stave contains a **[nocount]** directive (which is the most common use of **[nocount]**) it is identified as bar number zero, provided that the **bar** directive has not been used. If there is more than one such bar at the start of a stave, they are identified as ‘0’, ‘0.1’, ‘0.2’, etc.

Bars other than at the start of a stave which contain **[nocount]** directives are identified by the number of the previous counted bar, followed by ‘.1’, ‘.2’, etc. as needed. This also applies to bars at the start of a stave if **[bar]** has been used to set an initial bar number other than one.

For example, the following input contains five bars that would be identified in messages as ‘0’, ‘1’, ‘2’, ‘2.1’, and ‘3’:

```
[stave 1 treble 1]
[nocount] a | gggg | cd [nocheck] :) |
[nocount nocheck] ef | gggg |
```

The number of bars in each stave is included as part of the information that appears as a result of specifying **-v** on the PMW command line. (See section 4.1 for further information on this.) The count is given as the number of bars that do not contain **[nocount]**, followed by the count of bars that do contain **[nocount]**, if any, enclosed in brackets and preceded by a plus sign. The count for the example above would be ‘3(+2)’.

24. Movements

Several different musical movements may be typeset in a single PMW run by using **[newmovement]** to separate them, as described in chapter 21. The term ‘movement’ is something of a misnomer. All it means to PMW is that another piece of music is to follow, possibly on the same page as the previous one. A ‘movement’ may be as short as a few bars of a musical example for insertion in a book.

PMW starts a new page for a new movement unless there is enough room on the page for the headings and the first system of the new movement, or, if the first system contains only one stave, two such systems. This can be overridden by options on the **[newmovement]** directive (see section 47.49).

In general, most parameters which can be set by heading directives persist from movement to movement, but **key**, **layout**, **notime**, **startbracketbar**, **startnotime**, **suspend**, **time**, **transpose**, and **unfinished** apply only to the movement for which they are specified. **Notespacing** persists in one of its forms, but not the other.

If **notespacing** is used to set absolute note spacings at the start of a movement, for example,

```
  notespacing 33 30 24 18 14 12 10 10
```

these spacings are reset as the defaults at the start of subsequent movements. However, if **notespacing** is used to multiply the note spacings by a factor, for example

```
  notespacing *1.2
```

this change does not persist into the next movement.

Of the parameters whose values persist, most may be changed by heading directives at the start of the new movement. However, the following directives may appear at the start of the first movement only: **landscape**, **magnification**, **maxvertjustify**, **musicfont**, **nokerning**, **page**, **pagelength**, **pssetup**, **sheetdepth**, **sheetsize**, **sheetwidth**, and **textfont**.

24.1 Headings and footings

Page headings and footings persist from movement to movement, but new ones can be specified if required. New page headings and footings completely replace those of the previous movement, and are used at the first page break of the new movement.

For all movements, if no **footing** is given, but there is a **pagefooting** (either given for the movement or carried on from the previous one) then the page footing is printed at the bottom of the first page as well as on all subsequent pages.

One exception to the above is when a new movement continues on the same page as one or more previous movements. If a **footing** was specified for a previous movement but has not yet been printed (i.e. this is still the first page of that movement) and the subsequent movements do not themselves have overriding **footing** directives, then that footing is printed on the page.

If, for example, a copyright footing is defined at the start of the first movement, it will be printed at the bottom of the first page, even if the second movement starts on that page, provided the second movement does not itself contain any **footing** directives.

If the start of a new movement coincides with the top of a new page, the page heading is printed, followed by the heading for the new movement. This means that, for example, if page numbers are specified in the first movement by a **pageheading** directive, they will be printed by default on all subsequent pages.

Sometimes it is required to suppress page headings at the start of a new movement, for example if they are being used to print the name of the movement at the top of each page. This can be done by adding the keyword ‘nopageheading’ to the **[newmovement]** directive. For example:

```
  [newmovement nopageheading]
```

This option can be used with or without the ‘newpage’ option; it takes effect only if the new movement actually starts at the top of a page.

When a new movement starts at the top of a page there is sometimes a requirement for a special footing to be printed on the last page of the preceding movement. This can be requested by the use of

```
  [newmovement uselastfooting]
```

which then uses the **lastfooting** setting for this purpose. It can then be reset for the new movement if necessary.

25. Dimensions

The unit of length used by PMW is the printer’s *point*. As defined by the PostScript language this is equal to 1/72 of an inch (the true printer’s point is slightly smaller). Thus one millimetre is 2.835 points. Whenever a dimension is required in a PMW directive, its units are always points. For example,

```
  linelength 720
```

specifies a line length of 720 points, that is, 10 inches. PMW works internally in millipoints (that is, thousandths of a point), and any dimension can be given with a decimal point and a fractional part, though any digits after the third decimal place are ignored. For example,

```
  barlinespace 3.5
```

specifies that the space after bar lines should be 3.5 points.

When the output is being magnified (or reduced), dimensions specified by the user refer to the unmagnified (or unreduced) units, with the exception of the line length, page length, sheet depth, and sheet width, which are always in absolute units. Thus, for example, if the line length is set to 480 points, it remains 480 points at a magnification of 1.5, but if the distance between staves is set to 50 points, the staves are actually printed 75 points apart at this magnification. This means that a change of magnification does not require other dimensions in the input to be changed.

The following dimension information (in points) is given to help users who want to position items manually on the page:

distance between stave lines	4
width of note heads	6
default text baseline level below stave	10
default text baseline level above stave	4

The solid vertical line of the bracket that is used to join the staves of a system together is 2 points wide. This is another useful reference when trying to make dimensional judgements.

26. Horizontal and vertical justification

The word ‘justification’ is used in a typesetting context to describe the way in which a line of text is arranged within its boundaries. ‘Left justified’ means that the line begins hard up against the left-hand edge; ‘right justified’ means it is hard up against the right-hand edge. If both left and right justification are required, then the line must be stretched out so that it fits exactly between the boundaries.

There is also a concept of ‘vertical justification’, in which the lines of a page are spread out so that the page is exactly filled, instead of leaving blank space at the bottom.

In typesetting music, similar considerations apply, with musical systems taking the place of lines. Normally, systems are stretched to fill out the entire width required, but there are occasions when this is not required, or would look silly because the line is very short. Similarly, it is often necessary to spread systems vertically so that the bottom stave is at the same level on each page.

PMW supports both horizontal and vertical justification. By default, both are enabled, but the **justify** and **[justify]** directives allow the user to control the justification of each page and each system if required. The **topmargin** and **bottommargin** directives offer some further flexibility in the page layout.

27. Time signatures

Time signatures are specified to PMW by separating the two numbers with a slash. For example, 3/4 specifies waltz time. There are two special time signatures that are specified as letters:

- The letter `C` specifies 'common time' – equivalent to 4/4 but printed using the conventional character 𝄴.
- The letter `A` specifies 'alla breve' – equivalent to 2/2 but printed using the conventional 'cut time' character 𝄵.

A time signature can be preceded by a number and an asterisk. This has the effect of multiplying the number of notes in the bar for the purposes of checking bar lengths. However, the time signature is printed as given. Thus, for example, the time signature `2*C` prints as 𝄴, but expects there to be four minims rather than four crotchets in a bar, while `2*3/4` prints as 3/4 but expects three minims in a bar.

There are options for suppressing the printing of time signatures at various places, and there is a directive (**printtime**) which can be used to specify exactly how certain time signatures are to be printed. For example, 8/8 can be printed as 3+3+2/8, or only a single, large number can be printed.

By default, numerical time signatures are printed using the bold font. However, the **timefont** heading directive can be used to specify an alternative. In addition, if **printtime** is used, the normal font-changing escape sequences can be used in the strings that are specified.

PMW imposes no limitations on the values of the numbers used in time signatures.

It is possible to print music where different staves have different time signatures. For compatible cases (e.g. 3/4 *vs* 6/8) no special action is necessary. For other cases (e.g. 2/4 *vs* 6/8) the **[time]** stave directive has to be used to specify the conversion.

28. Key signatures

Key signatures are specified by key letter, followed by a PMW accidental character if necessary, and possibly the letter `m` to indicate a minor key. PMW uses the sharp character (`#`) to indicate a sharp, but because there is nothing resembling a flat on a computer keyboard, the key that is adjacent to sharp on some keyboards, the dollar sign (`$`) is used. Thus:

`a`	means A major
`c#m`	means C sharp minor
`B$`	means B flat major
`CM`	means C minor

All the standard key signatures are supported. See the next chapter for a discussion of key signatures after transposition.

29. Transposition

Octave transposition can be specified for each stave, to simplify the input notation. See the **[octave]** and the various clef directives (**[treble]**, etc).

In addition, general transposition can be specified for the whole piece or for individual staves. PMW can transpose up or down by an arbitrary number of semitones.

A transposition for the whole piece can be specified externally, via the **-t** command line option, or within the input file by the **transpose** heading directive. Transposition for individual staves is specified with **[transpose]**. If more than one transposition is present, the effect is cumulative.

PMW transposes key signatures as well as notes. Thus a piece which is to be transposed should be input with its original key signature(s) specified in the normal way. When **[transpose]** is used to transpose a single stave, only those key signatures which follow the directive in the input are transposed.

The key signature of F♯ major is used in transposed output only if specially requested via the **transposedkey** directive, G♭ being used by default. A number of other keys are also not used by default but can be specially requested. The complete list is as follows:

B	major	instead of	C♭	major
D♭	major	"	C♯	major
G♭	major	"	F♯	major
G♯	minor	"	A♭	minor
B♭	minor	"	A♯	minor
E♭	minor	"	D♯	minor

The **transposedkey** directive also has uses when transposing music where the key signature has fewer accidentals than the tonality.

If a note is specified with an accidental, then an accidental will always be present by default after transposition, whether or not it is strictly necessary. This ensures that ‘cautionary accidentals’ are preserved over transposition. There is an option to suppress this for individual notes, and the **transposedacc** directive can be used to suppress it throughout a piece.

29.1 Transposition of chord names

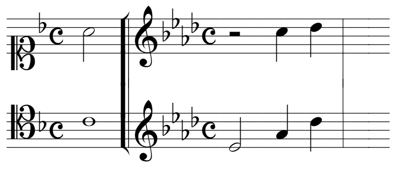
PMW can automatically transpose the names of chords in text strings. This is achieved by means of a special escape sequence `\t`. For example, in the string

```
"Sonata in \tE$"
```

the sequence `\tE$` is replaced by `E` and a flat sign when no transposition is taking place and by `F` with a transposition of +2. Full details of string escape sequences, including chord name transpostion, are given in chapter 34.

30. Incipits

The word *incipit* is the name given to stave notation that appears before the first bar of a piece, as commonly seen in scholarly editions. This notation is often used to show the original clef and other information about the piece. Here is a typical example:



This example was produced by using the **startbracketbar** directive to 'indent' the joining bracket. The input was as follows:

```
startbracketbar 1

[stave 1 soprano 1 key F time C nocheck]
A | [treble 1 key a$ time c] Rc’d’ |
[endstave]

[stave 2 tenor 1 key F time C nocheck]
C\M+\ | [treble 1 key a$ time c] Ead’ |
```

If an incipit is required on one stave only, for example, to print a single voice introduction at the start of a liturgical item, then the other staves can be completely suppressed by making use of the **[omitempty]** directive.

Another style of incipit leaves blank space between the incipit stave and the start of the piece proper. With a little bit of trickery, PMW can cope with this as well. The incipit and the rest of the piece must be input as separate 'movements', separated by

```
[newmovement thisline]
```

The incipit movement must be specified as left justified, and the start of the next movement as right justified, switching to left and right justification on the second system. If necessary, **[newline]** can be used to control the number of bars that are printed in the first system.

31. Text fonts

PMW supports the use of a number of different fonts, or typefaces, for use when printing text. As well as the standard four (roman, italic, bold face, and bold-italic), the use of a symbol font and of the music font in text is supported. In addition, up to twelve other fonts can be defined by the user.

The different kinds of text (e.g. underlay or bar numbers) each have a default font, and there are directives to change these. The fonts are referred to by the following names:

`roman`	the roman font
`italic`	the italic font
`bold`	the bold face font
`bolditalic`	the bold-italic font
`symbol`	the symbol font
`music`	the music font, at 0.9 size
`bigmusic`	the music font, at full size
`extra <`*n*`>`	the the <*n*>th extra font

The **textfont** heading directive is used to define exactly which fonts correspond to these names. By default, the *Times* series of fonts are used for text, and the *Symbol* font for symbols. PMW needs access to the ‘fontmetrics’ file of every text font that it uses. Fontmetrics files for the standardly available PostScript fonts are supplied with PMW. If you want to use other fonts, you will need to obtain the appropriate fontmetrics files and install them in PMW’s **fontmetrics** directory.

The music font is available at two different relative sizes, because the musical characters look too large if printed alongside text at the same point size, for example, when printing tempo markings.

The next chapter describes how font sizes are specified, and the following one explains how text strings are coded, including how the font may be changed within any string.

32. Font sizes, aspect ratios, and shearing

Many PMW directives allow you to specify a size for a font. For example, when defining a heading by a line such as

```
  heading 15 "|Sonatina" 30
```

the first number (15) specifies that the text is to be printed using a 15-point font. There are further parameters that you can specify to control the size and shape of any text font. These are coded as two additional numbers, separated from the main size value by slashes. For example:

```
  heading 15/1.3/10 "|Sonatina" 30
```

The first additional parameter is a horizontal stretching factor which alters the aspect ratio of the font. If it is greater than one, the resulting font appears short and fat; if it is less than one, the appearance is tall and thin. Stretching a font horizontally makes it look larger without using up any more vertical space.

This 10-point font is neither stretched nor compressed.
This 10-point font is stretched horizontally by 1.2.
This 10-point font is compressed horizontally by 0.8.

The second additional parameter is a shearing angle, measured in degrees. It specifies the angle between the true vertical and what were originally vertical lines in the font. A positive shear angle causes the font to slope to the right. Sheared roman fonts are sometimes used instead of italic fonts. For example,

```
  heading 14/1/20 "Slanted text"
```

prints the heading with a 20° shear.

This 10-point font is sheared by 20 degrees.

The stretching and shearing parameters can be specified in all the places where a text font size can be specified.

33. Paper size

PMW assumes by default that printing is to take place on A4 paper and so its default is to create a page image of that size. If a different size is required, the **sheetwidth** and **sheetdepth** directives can be used to specify what its dimensions are.

For standard page sizes, it is not normally necessary to use **sheetwidth** and **sheetdepth**, because the **sheetsize** directive, which takes as its argument the name of a page size, can be used instead. All the **sheet...** directives may appear only in the first movement of a file.

Sheetsize takes as its argument one of the words ‘A3’, ‘A4’, ‘A5’, or ‘B5’, and it has the effect of setting the sheet width and depth to the correct values for the given size. It also sets the page length and line length parameters to appropriate default values for the page size, but these can be changed by subsequent appearances of the **linelength** or **pagelength** directives if necessary.

Sheetsize should therefore be given at the top of the file before any use of **linelength** or **pagelength**, and also before any use of the **landscape** directive.

In the most common case, the page image size is the same as the actual size of paper being used, but PMW does also support *two-up* printing, in which two page images are printed next to each other on a larger piece of paper. Details of this may be found in chapter 4.

34. Text strings

Text strings (often just called ‘strings’) are used in a number of different places in PMW to define text that appears on the page with the music. They must always be enclosed in double-quote characters. The double-quote character itself cannot appear in a string (but can be printed using a character number, if necessary). There is no limit to the length of a string.

Three characters are treated specially in all strings:

- The quote character `'` and the grave accent character `` ` `` are converted into closing and opening quote characters, respectively, except when using the music font.
- The backslash character `\` is an *escape character* (see below).

In vocal underlay strings, which are described in detail in chapter 45, a number of additional characters are treated specially.

In headings and footings, the vertical bar `|` serves to separate the left-hand, middle and right-hand parts of the text. In text that appears at the start of a stave, it serves to delimit individual lines.

34.1 Escaped characters

The backslash character is used as a means of including characters that are not in the normal computer character set, for specifying changes of font, and for some other special effects. In all character strings, the following sequences are available to represent the commonly accented characters in European languages:

`\a'`	prints á
`` \a` ``	prints à
`\a^`	prints â
`\a.`	prints ä

The same accents are available for the other four vowels, and can be used with both upper case and lower case letters.

`\c,`	prints as ç
`\c)`	prints as ©
`\c]`	prints as © (but see below)
`\fi`	prints as the ligature ﬁ
`\fl`	prints as the ligature ﬂ
`\n~`	prints as ñ
`\ss`	prints as ß
`\?`	prints as ¿
`\\`	prints as \
`\'`	prints as ′
`` \` ``	prints as `
`\--`	prints as –
`\---`	prints as —

The normal way to print a copyright symbol is to use `\c)` because this prints it in the current font. However, some older PostScript printers do not have a copyright symbol in every font. The alternative escape sequence `\c]` is provided to print a copyright symbol from the PostScript *Symbol* font in this circumstance.

Other characters that are not on the keyboard can be included in strings by giving the decimal character number enclosed between two backslashes. For example, `\183\` prints a bullet character. The encoding for text fonts is the standard Latin1 encoding.

Characters that are treated specially in text strings can also be printed by this means. For example, `\34\` prints a double-quote character.

The interpretation of string escape sequences happens after a string has been split up into different parts for headings or for underlay text. Therefore it is possible to print the splitting characters, should they ever be wanted, by specifying their character number. For example, the sequence `\124\` can be used to print a vertical bar in a heading line.

Characters from the PostScript *Symbol* font are also available for use in text strings. This font contains some large brackets which are sometimes useful. To include a single character from this font, specify its decimal character number preceded by `s` and enclosed in backslashes. For example, `\s212\` prints character 212, which is the ™ trademark symbol.

34.2 Page numbers

There are three escape sequences that are different to the others in that they do not generate a particular fixed character:

`\p\`	prints the current page number
`\po\`	prints the current page number if it is odd
`\pe\`	prints the current page number if it is even

If the page number is even, `\po\` prints nothing, and if it is odd, `\pe\` prints nothing. These are made available for use in heading and footing lines, to enable page numbers to be printed on the right or left as appropriate.

There is an additional facility for skipping parts of the string depending on the value of the page number. Any characters between two occurrences of the substring `\so\` are skipped if the page number is odd, and similarly for `\se\` if the page number is even. This makes it possible to specify page headings of the form

```
pageheading "\so\page \p\\so\||\se\page \p\\se\"
```

which will print 'page <*n*>' on the left or right of the page, depending on the value of the page number.

34.3 Comments within strings

There is a facility for in-string comments. Any characters between the string `\@` and the next backslash are ignored. This can be useful when an entire piece's underlay is being input as a single, very long string. However, if such a comment in an underlay string is surrounded by spaces, it acts as an empty syllable.

34.4 Transposing chord names

A special escape sequence is provided to define the names of chords that should be changed if the piece (or stave, for strings associated with a stave) is being transposed. This makes it straightforward to transpose pieces that show chord names above a line of music.

The escape sequence is `\t`, and it must be followed by one of the letters A–G, in upper case. This may optionally be followed by one of the characters `#`, `$`, or (for completeness) `%`. Such a sequence has two effects; firstly, the chord name is transposed in the same way as its base note would be transposed, and secondly, if the new chord name involves a sharp or a flat, the correct sign is used, with appropriate spacing adjustment. Thus, even without transposition, this notation is a convenient way of specifying chord names that involve accidentals.

Natural signs are never used on transposed note names. The rules for transposing real notes can yield a new note with double sharp or a double flat. When this happens for a chord name, the enharmonic name is substituted. For example, G is used for F double-sharp.

When a string that involves a transposable chord name appears in a header or footer line, only **transpose** heading directives that are earlier in file are applied to it, because the transposition is performed when the string is read.

34.5 Musical characters

The backslash character can also be used to include single musical 'characters' in textual output. Typical uses of this are for indicating tempo by printing a note followed by an equals sign and its metronome marking, or for printing sharps and flats in the names of instruments. To include a note from the musical font, the following special sequences can be used:

`\*b\`	prints a breve
`\*s\`	prints a semibreve
`\*m\`	prints a minim
`\*c\`	prints a crotchet
`\*Q\`	prints a quaver
`\*q\`	prints a semiquaver

Any of the above can include a dot after the note letter to print the dotted form of the note, e.g. `\*c.\`. The accidental characters are available as follows:

`\*#\`	prints a sharp
`\*$\`	prints a flat
`\*%\`	prints a natural

A typical example of a tempo marking which uses this facility might be

```
"Maestoso \*c\ = 60"   @ 60 crotchets per minute
```

which prints as

Maestoso ♩ = 60

Musical characters included in character strings with a single asterisk in this way are printed using a musical font which is 9/10 the nominal size of the surrounding text characters. This is an appropriate size for items such as tempo markings. Thus, if a 10-point text font is being used, a 9-point music font is used with it.

The music font which is used to print the actual music being typeset is a 10-point font, and it is sometimes useful to be able to print musical characters at full size. If two asterisks are present in an escape sequence for a musical character, the character is taken from a music font which is the same size as the text font. Since the default text fonts are the same size as the standard music font, this gives music characters at the same size as those being set by PMW.

There are a number of ‘characters’ in the music font that do not actually cause any marks to be made on the page. However, ‘printing’ these characters has the effect of moving the current printing position, thus affecting the placing of the following characters. The following sequences (which may also be used with two asterisks) access some of these special characters:

`\*u\`	moves up by 0.2 times the font’s size
`\*d\`	moves down by 0.2 times the font’s size
`\*l\`	moves left by 0.33 times the font’s size
`\*r\`	moves right by 0.55 times the font’s size
`\*<\`	moves left by 0.1 times the font’s size
`\*>\`	moves right by 0.1 times the font’s size

Thus, for example, in a 9-point music font, `\*u\` moves up by 1.8 points. This is half the distance between stave lines for a 9-point music font.

If more than one ‘escaped sequence’ starting with an asterisk is required in succession, they can all appear between a single pair of backslashes, for example, `\*#*c\`. However, you cannot mix single and double asterisks between the same pair of backslashes.

It is possible to print *any* single character from the musical font by specifying its decimal character number, preceded by one or two asterisks, between backslashes. A list of the available characters is given in chapter 48. For example, the sequence `\*45\` prints a crotchet rest.

34.6 Font changes

Roman, italic, bold and bold-italic fonts are available for all text printed by PMW. By default, these use the Times series of fonts but can be changed by the **textfont** heading directive. In addition, the user may define up to twelve additional fonts via **textfont**. If any of these is used without being defined, the roman font is substituted.

The initial font setting at the start of each character string is roman for all text that is not part of any stave’s data. Within a stave, the default depends on whether the text is underlay, overlay, figured bass, or other text. For underlay, overlay, and figured bass the default is roman, while for other text it is italic. These defaults can be changed on a per-stave basis (see the **[underlayfont]**, **[overlayfont]**, **[fbfont]**, and **[textfont]** stave directives).

Within a text string, the following special character sequences are used to change font:

`\rm\`	change to roman
`\it\`	change to italic
`\bf\`	change to bold face
`\bi\`	change to bold-italic
`\sc\`	change to small caps
`\sy\`	change to the Symbol font
`\mu\`	change to the music font at 0.9 size
`\mf\`	change to the music font at full size
`\x1\`	change to the first extra font
`...`	
`\x12\`	change to the twelfth extra font

For example:

```
"\rm\this is roman \it\this is italic \bf\this is bold"
```

Note that the letters involved are always in lower case. A change of typeface does not persist beyond the end of the text string in which it appears.

The default of italic for ordinary text within staves is appropriate for dynamic markings such as *ff*, though sometimes a bold-italic type is used for this kind of mark. Tempo markings at the start of pieces are normally printed in bold face, as in the following example:

```
  "\bf\Adagio"
```

Changing to SMALL CAPS does not in fact change the typeface, nor does it force subsequent letters to be capitals; it just changes to a smaller font of the same typeface as the current font. The effect lasts until the next font change.

The relative size of small caps can be set by the **smallcapsize** heading directive, whose argument should be a number between 0 and 1. The default value is 0.7, because this makes small caps whose height is equal to the x-height of the normal font in the Times series of fonts, and this is the usual typographic convention.

34.7 Sizes of text strings

The heading directives that specify page headings and footings allow arbitrary sizes to be given for those texts. Text within a stave is by default printed using 10-point fonts, but various facilities are provided for changing this.

Underlay, overlay, figured bass, and other text each have their own separate default sizes, which are set up by heading directives. In addition the user may specify up to eleven additional sizes which can be requested for any particular item of text. Details are given in section 38.114 and chapter 44.

Whenever the size of a text font is specified, an associated aspect ratio and/or shearing angle may also be specified. See chapter 32 for details.

Stave text strings that are not underlay or overlay can be rotated so that they print at an angle. Details are given in chapter 44 (*Text strings in stave data*). Text at the start of a stave can be rotated so as to print vertically instead of horizontally – see the description of **[stave]** in section 47.82.

34.8 Kerning

Kerning is the word used to describe the practice of moving certain pairs of letters closer together or (more rarely) further apart, in order to improve the appearance of text. Compare, for example, ‘Yorkshire’ (kerned) with ‘Yorkshire’ (unkerned).

PMW makes use of the kerning information in fontmetrics files automatically. This action can be disabled by including the directive **nokerning** in the heading of the first movement. To prevent kerning between a particular pair of characters, a redundant font change can be used. For example,

```
  "\rm\Y\rm\orkshire"
```

is printed without the o being moved nearer to the Y.

35. Stave 0

The normal staves of a piece are numbered from 1 to 63. In addition, data for a special stave, numbered 0, can be supplied. This stave is by default overprinted on the topmost stave of each system; the **-s** stave selection option on the command line does not affect it.

No stave lines, clefs, key signatures or time signatures are printed for stave 0, and any notes that are specified are treated as ‘invisible’. However, text items are printed.

The intended use of stave 0 is for setting up text items that are to be printed above the topmost stave, whatever combination of staves is selected for printing from the full score. This saves having to input the text items with each part. Dummy notes can be supplied to ensure that the text items are horizontally aligned where they are required. A typical example might be:

```
[stave 0]
"Allegro"Q+ | [15]Q! | Q "rit." Q | [23]Q! |
"with feeling" Q+ |
[endstave]
```

Overprinting stave zero on the top stave of each system is the default action of PMW. In fact, the **copyzero** heading directive makes it possible to have copies of stave zero printed over any number of staves. It is followed by a list of stave numbers, each of which may be optionally followed by a slash and a dimension. The dimension is a vertical adjustment to the level of stave zero for the given stave. For example:

```
copyzero 1 7/10 11/-2
```

All the staves over which stave zero is to be printed must be specified, including the top stave. Different versions of **copyzero** can be used for different movements; if not given, it is copied from the previous movement.

If a stave over which stave zero is being printed is suspended, then stave zero is printed over the next following non-suspended stave, if there is one. However, if that stave itself is listed in the **copyzero** directive, then its spacing parameter is used. In general, if, as a result of suspension or overprinting, stave zero is requested to be multiply printed at any given level, then the spacing parameter for the highest numbered stave is used.

Selection of a subset of staves for printing is equivalent to the suspension of all others. Thus the default setting of

```
copyzero 1
```

has the desired effect of printing over individual staves that are extracted as parts. If it is necessary to adjust the overall level for a particular part, constructions such as the following can be used:

```
*if stave 9
copyzero 9/4
*fi
```

There is also a **[copyzero]** stave directive, which takes a dimension as an argument, and adjusts the vertical level of any stave zero material in the current bar when stave zero is printed at the level of the current stave. For example,

```
[copyzero 4]
```

raises the stave zero material in the current bar by 4 points.

It is not necessary for there to be an instance of the **copyzero** heading directive specifying the current stave for **[copyzero]** to take effect. In the default case, **[copyzero]** takes effect whenever the stave in which it appears is the top stave of a system.

When first and second time bar markings are specified in stave zero, and there is a need to adjust their height for certain staves, it should be noted that the markings are drawn when the bar in which their end point is determined is processed. Consequently, it is that bar in which **[copyzero]** should appear. The same applies to slurs and lines.

36. Temporarily suspending staves

When a part is silent for a long period of time, it is conventional in full scores to suppress its stave from the relevant systems. The term ‘suspended’ is used to describe a stave that is not currently being printed. PMW does not suspend staves automatically, but requires an instruction in the input to tell it to do so (see **[suspend]** in section 47.86).

Staves can be suspended only if they contain no notes or text items, though other items such as time and key signature changes may be present. Resumption of printing is automatic, though there is also a **[resume]** directive for forcing it to happen.

It is conventional to print all the staves in the first system of a piece, even if some of them contain only rest bars. However, there is a heading directive called **suspend** that enables PMW to suspend individual staves right from the start (see section 38.111).

Normally, a stave which is not suspended will be printed right across the system, with rest bars as appropriate. However, a stave can be tagged with the **[omitempty]** directive, in which case completely empty bars are not printed at all. This can be useful for printing *ossia* passages, for example. A completely empty bar has no data at all specified for it; a bar containing a rest is not a completely empty bar.

When a single part is being printed, suspension normally has no effect, since multiple rest bars are packed up into a single bar with a count printed above, and so systems containing only rest bars do not occur. However if `S!` is used for rest bars, it prevents the amalgamation of adjacent bars and may lead to suspendable systems. In these cases, the **[suspend]** directive should be skipped (using the ***if** preprocessing directive) when printing the part.

37. Drawing facilities

PMW contains a facility for drawing simple shapes, defined by the user, positioned relative to notes, bar lines, headings, stave names, or gaps in slurs and slur-like lines. This makes it possible to print musical notation that is not provided explicitly by PMW.

For example, the facility can be used to draw piano pedal markings, boxes round notes, vertical brackets between notes, and to print unusual markings above or below the stave. It can be used with headings or footings to rule lines across the page or to print crop marks.

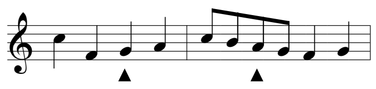
A simple programming language is used to describe drawings. Readers unfamiliar with computer programming may find this chapter hard going and may prefer to skip it on a first reading.

Before describing the facility in detail, we consider a short example. Suppose there is a requirement to draw a solid black triangle, with its point upwards, 4 points below the stave. The first thing to do is to define this shape. This is done using the **draw** heading directive as follows:

```
draw triangle
  3 -4 moveto       @ move to apex
  -3 -6 rlineto     @ line to bottom left
  6 0 rlineto       @ horizontal line to bottom right
  -3 6 rlineto      @ line back to apex
  fill              @ fill it in (solid triangle)
enddraw
```

This example of **draw** defines a drawing called ‘triangle’. The lines between **draw** and **enddraw** are drawing instructions in a form which is described below. Whenever the triangle shape is wanted, the stave directive **[draw triangle]** is specified before the relevant note. For example:

```
c’f [draw triangle] g a |
c’-b’- [draw triangle] a’-g’- fg |
```



Obviously, if lots of triangles are required, it would be a good idea to use ***define** to set up a macro for **[draw triangle]** to save typing.

The ‘language’ used to describe drawings is based on the notion of a *stack*. This will be familiar to you if you have any experience of the computer programming languages Forth or PostScript. For those readers who are not familiar with stacks, we now explain how they work.

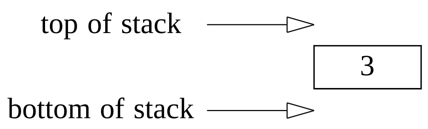
37.1 Stack-based operations

A stack is a means of storing items of data such that the last item which is put on the stack is the first item to be taken off it. An analogy is often drawn with the storage arrangements for trays in self-service restaurants, where a pile of trays is on a spring-loaded support. Trays are added to the stack on the top, thereby pushing it down; when a new tray is required, it is taken from the top of the stack, and the remainder of the trays ‘pop up’.

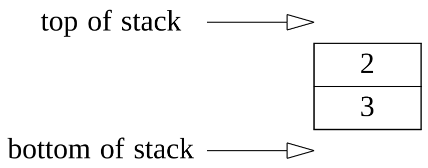
PMW’s drawing stack contains numbers and references to text strings rather than trays. (Discussion of strings is postponed till section 37.13.) When PMW is obeying a set of drawing instructions, if it encounters a number in its input, the number is ‘pushed’ onto the top of the stack. Consider the following fragment of a drawing program:

```
3 2 add
```

In this example, the first item is the number 3, so the effect of reading it is to put the stack into this state:

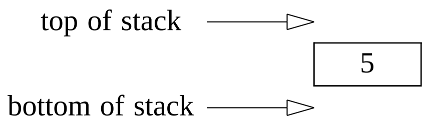


The second item is also a number, so after it is read, the stack is as follows:



The third item in this fragment is the word ‘add’. This is not a number – it is an *operator*. The operators used in PMW drawings are based on those found in the PostScript language.

When an operator is encountered, it causes PMW to perform an operation on the numbers which are already on the stack. In the case of the **add** operator, the two topmost numbers are ‘popped’ off the stack, added together, and the result is pushed back onto the stack. So in this case, after ‘add’ has been obeyed, the stack is like this:



The stack mechanism is very simple, and operates quickly. However, it does make it possible to write very obscure programs which are hard to understand. Use PMW’s comment facility to help you keep track of what is going on.

PMW does not clear the drawing stack between one invocation of **[draw]** and the next. This provides one way of passing data between two drawing function calls, and there is no problem if the related drawing functions are called in the same bar of the same stave, because they will be always obeyed in the order in which they appear in the input.

However, you must not rely on the order in which PMW processes bars and staves, other than that bar *n* will be processed before bar *n*+1 on any particular stave, but not necessarily immediately before it (a bar on another stave may intervene). Apart from this, the order of processing, and therefore the order of obeying **[draw]** directives on several staves, is not defined, and may change between releases of PMW.

Therefore, if you need to pass data between drawing functions in different bars, and use this facility on more than one stave, the stack cannot be used. *User variables* (described in section 37.12) must be used instead.

If an operator is encountered which requires more numbers on the stack than there are present, *stack underflow* is said to occur. PMW generates an error message and abandons its attempt to display or print the page.

37.2 Drawings with arguments

Whenever a drawing function is called (by the **[draw]** directive or as part of some other directive), its name may be preceded by a list of numbers or text strings (see section 37.13), separated by spaces. These are pushed onto the drawing stack immediately before the function is obeyed, and therefore act as arguments for the function. For example,

```
heading draw 44 logo
[draw 3 -5.6 thing]
[linegap/draw 8.2 blip]
[slurgap/draw "A"/c annotate]
```

There is no explicit facility for default values, but these can be provided by using a macro with arguments to call the drawing function (see section 22.3).

37.3 Arithmetic operators

The following arithmetic operators are provided for use in drawing descriptions:

- **add**: Add the two top numbers on the stack, leaving the result on the stack.
- **div**: Divide the second topmost number on the stack by the number on the top of the stack, leaving the result on the stack.
- **mul**: Multiply the two top numbers on the stack, leaving the result on the stack.
- **neg**: Negate the topmost number on the stack, leaving the result on the stack.
- **sub**: Subtract the topmost number on the stack from the second topmost number, leaving the result on the stack.

Evaluation of the expression ((3+4) × 5 + 6)/7 could be coded as follows:

```
3 4 add
5 mul
6 add
7 div
```

37.4 Truth values

The operators **false** and **true** push the values 0 and 1 onto the stack, respectively. These are the same values that are returned by the comparison operators, and can be tested by the conditional operators.

37.5 Comparison operators

The following operators operate on the top two values on the stack and leave their result on the stack. The values must be numbers – if they are not, the result is undefined. Otherwise the result is 1 for *true* and 0 for *false*.

eq	test equality
ne	test inequality
ge	test first greater than or equal to second
gt	test first greater than second
le	test first less than or equal to second
lt	test first less than second

For example,

```
10 10 eq
```

would leave the value 1 (*true*) on the stack, while

```
25 4 lt
```

would yield 0 (*false*). The conditional operators can be used to test these values.

37.6 Bitwise and logical operators

The following operators perform bitwise operations on the integer parts of the top two values on the stack. The result always has a zero fractional part.

and	bitwise and
or	bitwise or
xor	bitwise exclusive or

The **not** operator performs bitwise negation on the top number on the stack. These bitwise operators act as logical operators when applied to the results of the comparison operators. For example,

```
5 6 ne 13 7 gt and
```

leaves 1 (*true*) on the stack, because 5 is not equal to 6 and 13 is greater than 7.

37.7 Stack manipulation operators

There are a number of operators which perform manipulations of the numbers on the stack:

- **copy**: Remove the top number from the stack, then duplicate that number of items on the stack by pushing a second copy of the sequence of items onto the stack. For example, if the stack contained the numbers 23 and 53, then after

  ```
  2 copy
  ```

  it would contain 23, 53, 23, 53.

- **dup**: Duplicate the number on the top of the stack. This is equivalent to **copy** with an argument of 1.
- **exch**: Exchange the two top numbers on the stack.
- **pop**: Remove the topmost number on the stack, and discard it.
- **roll**: This operator performs a circular shift of items on the stack. When it is encountered, there must be three or more items on the stack. The topmost item on the stack is a count of the number of positions by which items are to be shifted. The second topmost item is the number of items involved, and there must be at least this many additional items on the stack. PMW first removes the two control numbers from the stack. Then it shifts the given number of items by the given amount.

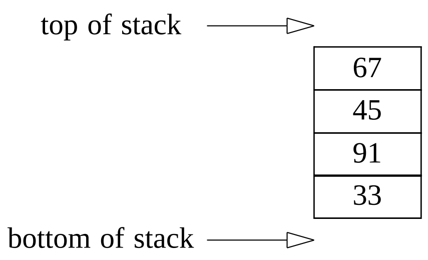
  If the amount of shift is positive, each shift consists of removing an element from the top of the stack, and inserting it below the last element involved in this operation. This is an ‘upwards’ roll of the stack.

  If the amount of shift is negative, each shift consists of removing the lowest element involved in the operation, and pushing it onto the top of the stack. This is a ‘downwards’ roll of the stack.

Here is an example of the use of **roll**. After obeying the input

```
33 45 67 91 3 1 roll
```

the stack will be as follows:



The three elements 45, 67, 91 have been rolled upwards one place.

37.8 Coordinate systems

The coordinate system used in PMW drawings is a traditional X-Y system, with all distances specified in points. The initial position of the origin of the coordinates depends on the item with which the drawing is associated. PMW drawings can be associated with four kinds of item:

(1) They can be associated with a note (or chord) on a stave, or with the end of a bar if no notes follow the **[draw]** directive. In this case, the origin is on the bottom line of the stave, either at the left-hand edge of the associated note, or at the bar line.

The left-hand edge of a note or chord with a downwards pointing stem is the edge of the stem. This applies even when a chord has some noteheads moved to the other side of the stem. For breves and semibreves the behaviour is as if there were a stem.

Noteheads are 6 points wide, so the horizontal coordinate of the centre of a note is 3. That is where the 3s in the triangle example come from.

(2) Drawings can be associated with heads or feet. The origin of the coordinate system is at the left-hand side, at the level of the next heading or footing line. See the **heading** directive for more details.

(3) Drawings can be associated with a gap in a line that is defined by the **[line]** directive or with a slur. For details see the **[linegap]** and **[slurgap]** directives in sections 47.41 and 47.76, respectively.

(4) Drawings can be associated with the text that is printed at the start of a stave. The origin of the coordinate system is at the left-hand side of the page, at the level of the bottom line of the stave. For details, see the **[stave]** directive in section 47.82.

37.9 Moving the origin

There is an operator called **translate** which moves the origin of the coordinate system to the point specified by the two numbers on the top of the stack, relative to the old origin.

37.10 Graphic operators

PMW follows the PostScript model in the way drawn marks on the page are specified. There are operators that set up a description of a *path* (i.e. an outline) on the page, and this outline is then either filled in completely, using the **fill** operator, or a line of given thickness is drawn along the path, using the **setlinewidth** and **stroke** operators.

A path may consist of several segments, which may be lines or curves. There can be gaps in the path. A single path can consist of a number of disconnected segments.

A path definition must always start with a **moveto** operation, in order to establish an initial current point. Thereafter, a mixture of moving and drawing operators may be specified.

Distances are, as always, expressed in points. They are subject to the overall magnification in the same way as other dimensions. They are not, however, subject to the relative magnification of individual staves, but there is a variable which contains the magnification of the current stave (when the drawing is associated with a stave), so that adjustments can be made explicitly where required.

Whenever a pair of coordinates is required to be on the stack, it is always the x-coordinate which must be pushed first, and the y-coordinate second.

The graphic operators are as follows:

- **currentgray**: Push onto the stack the current value of the gray setting – see **setgray** below.
- **currentlinewidth**: Push onto the stack the current value of the line width setting – see **setlinewidth** below.
- **currentpoint**: Push the coordinates of the current point onto the stack. The x-coordinate is pushed first.
- **curveto**: This operator draws a Bezier curve. There must be six numbers on the stack when it is called; they are treated as three pairs of coordinates. The final pair are the end point of the curve, which starts from the existing current point. The two middle pairs of coordinates give the Bezier curve control points.

  If you are not familiar with Bezier curves, you will need to discover a bit about them before you can fully understand this operator. They are described in many books on computer graphics. Very roughly, the curve starts out from its starting point towards the first control point, and ends up at the finishing point coming from the direction of the second control point. The greater the distance of the control points from the end points, the more the curve goes towards the control points before turning back to the end point. It does not, however, pass through the control points.
- **fill**: This operator causes the interior of the previously defined path to be completely filled in in black. The path is then deleted, and a new one can be started.
- **fillretain**: This command behaves like **fill**, except that the path is retained and can be used again. See **setgray** below for an example.
- **lineto**: The path is extended by a line segment from the current point to the absolute position given by the two top items on the stack.
- **moveto**: If there is no current path, this must be the first graphic operator encountered. It establishes the initial current point. Otherwise, the path is extended by a move (i.e. invisible) segment from the current point to the absolute position given by the two top items on the stack.
- **rcurveto**: This operator acts like **curveto**, except that the three pairs of coordinates are taken as relative to the existing current point.

- **rlineto**: This operator acts like **lineto**, except that the coordinates are taken as relative to the existing current point.
- **rmoveto**: This operator acts like **moveto**, except that the coordinates are taken as relative to the existing current point.
- **setgray**: This operator is used to specify a gray shade for drawn items resulting from the use of **stroke** or **fill**. It does not apply to text. It must be preceded by a number between 0 (black) and 1 (white), for example:

  ```
  0.5 setgray
  ```

  In particular, this can be used to fill an area with white and thereby 'rub out' any previous marks on the page. For example, to blank out an area with white and draw a black line round its edge one could define the path and then use:

  ```
  1 setgray fillretain 0 setgray stroke
  ```

  If you do something like this on a stave of music, you should invoke the drawing with **[overdraw]** rather than **[draw]** as that ensures that it is output after everything else on the stave.
- **setlinewidth**: A single number is required on the stack to specify the width of lines to be drawn by the **stroke** operator. The default line width is 0.5 points. The value persists from one call of **[draw]** to the next.
- **show**: This operator prints a text string. Details are given in section 37.13 below.
- **stroke**: This operator causes a line to be drawn along the previously defined path, omitting any segments that were defined with **moveto** or **rmoveto**. Afterwards, the path is deleted, and a new one can be defined.

37.11 System variables

In order to set up drawings that are positioned according to the following note or chord (for example, to draw a box around it) it is necessary to have access to some data about it. There are a number of *system variables* which provide this, as well as other variable data values. When encountered in a drawing description, the name of a variable has the effect of pushing the relevant data value onto the stack.

The system variables are listed below. Those that relate to notes should be used only when the drawing function is called immediately before a note, and those that relate to staves and systems should not be used in drawings that are called as headings or footings.

- **accleft**: The distance from the left-hand edge of the leftmost notehead to the left-hand edge of the accidental which is furthest to the left. This is given as a positive number. If there are no accidentals, the value given is zero.
- **barnumber**: When used in a bar, this contains the bar number; when used at the start of a system, it contains the number of the first bar in the system. If used in headings or footings, it contains zero.
- **gaptype**: Contains +1 or -1 when a drawing function is being obeyed as part of a **[linegap]** directive; the value is positive for a line above the stave, and negative for a line below the stave. Otherwise the variable contains zero.
- **headbottom**: The y-coordinate of the bottom of the notehead; if the drawing function precedes a chord, this refers to the lowest notehead.
- **headleft**: For a chord with a downwards stem in which there is a notehead on the 'wrong' side of the stem, the width of this notehead is given as a positive number; otherwise zero is given.
- **headright**: The width of the notehead; for a chord with an upwards pointing stem in which there is a notehead on the 'wrong' side of the stem, twice the notehead width is given.
- **headtop**: The y-coordinate of the top of the notehead; if the drawing function precedes a chord, this refers to the highest notehead.
- **leftbarx**: The x-coordinate of the previous bar line, except in the first bar of a system, in which case it is the x-coordinate of a point slightly to the left of the first note in the bar.

- **linebottom**: This gives a value of 2 points (scaled to the stave size) if the bottom of the lowest notehead is on a stave (or ledger) line (i.e. the notehead itself is positioned in a space); otherwise it gives zero.
- **linegapx** and **linegapy**: When a drawing is being obeyed as part of the **[linegap]** directive, these variables contain the position of the start of the next part of the line. Otherwise they contain zero.
- **linelength**: This variable contains the current line length, as set by the **linelength** heading directive.
- **linetop**: This gives a value of 2 points (scaled to the stave size) if the top of the highest notehead is on a stave (or ledger) line (i.e. the notehead itself is positioned in a space); otherwise it gives zero.
- **magnification**: The value of the magnification setting.
- **originx** and **originy**: These variables are for use when more than one note position is participating in a drawing. They place on the stack the *absolute* x-coordinate and y-coordinate of the local coordinate system's origin, respectively.

  This makes it possible to leave absolute coordinate values in user variables or on the stack at the end of a call to **[draw]**. A subsequent **[draw]** program can relate these values to its own local coordinate system by its own use of **originx** and/or **originy**. An example of this is given below.

  The variable **origin** is equivalent to **originx**. It is provided for compatibility with previous versions of PMW when only the x-coordinate origin was available. It should not be used in new input files.
- **pagelength**: This variable contains the current page length, as set by the **pagelength** heading directive.
- **pagenumber**: The current page number.
- **stavesize**: The relative magnification for the current stave, as specified by the **stavesize** heading directive.
- **stavespace**: The current stave spacing for the current stave, that is, the vertical distance between this stave and the next one.
- **stavestart**: The x-coordinate of the left-hand end of the current system, relative to the current origin.
- **stembottom**: The y-coordinate of the bottom of the stem of the note or chord. If there is no stem, or if the stem points upwards, this gives the same value as **headbottom**.
- **stemtop**: The y-coordinate of the top of the stem of the note or chord. If there is no stem, or if the stem points downwards, this gives the same value as **headtop**.
- **systemdepth**: The distance from the bottom of the top stave of the current system to the bottom of the bottom stave.
- **topleft**: The coordinates of the position at which PMW starts writing on a page, relative to the current origin. This is normally one inch down from the top of the paper, and indented according to the sheet width and line length. This operator can be used with **translate** to move the origin to a fixed point on the page in order to draw such things as crop marks.

37.12 User variables

Up to 20 user variables are available for use in drawing functions. These variables are *global* in that they are shared between all drawing functions. When you want to pass values from one drawing function call to another, using a variable is often more convenient than leaving data on the stack.

The names of the variables are chosen by the user, but they must not be the same as any of the built-in variables or operators. To set a value into a variable, the following construction is used:

/<*name*> <*value*> `def`

The appearance of `/` indicates that the name of the variable is required, rather than its value. For example, to put the value 10 into a variable called `abc`:

```
/abc 10 def
```

Once a variable has been defined, its value is retrieved by simply quoting its name; this causes the value to be copied onto the stack.

A variable's name must be defined (that is, appear with a slash) before its value is used. If a variable is set in one drawing function and used in another, the definition of the one in which it is set must come first in the PMW input file. This is not always possible. For example, when the defining function calls the other function, the called function must come first. In such a case, a dummy drawing function which is never obeyed can be used just for the purpose of defining user variable names to PMW.

37.13 Text strings in drawings

Text strings can be printed from within the drawing mechanism. The appearance of a string in quotes inside a drawing definition or as an argument to a drawing function causes an item representing the string to be pushed onto the stack. Such an item can be copied or moved around the stack in the normal way, but the only operation that can be carried out on it is the **show** operation, which causes the string to be printed at the current point. For example,

```
draw string
  0 -12 moveto "text" show
enddraw
```

The string may contain font changes and other escape sequences. The default font is roman. In addition, it may be followed by the following options:

`/box`	enclose the string in a rectangular box
`/c`	centre the string at the current point
`/e`	align the end of the string with the current point
`/ring`	enclose the string in a ring (circular if a short string)
`/rot<`*n*`>`	rotate the string by <*n*> degrees
`/s<`*n*`>`	print the string at size <*n*>

where the string sizes are defined by the **textsizes** directive in the normal way. For a discussion of rotated text, see chapter 44.

When the string is centred or end-aligned, the printing of the string does not change the current point; in the other case, the current point is moved to the end of the string. A line may be begun before printing a string and continued afterwards.

As an example of the use of the text facility, consider music printed in the sixteenth and seventeenth century style where, instead of using ties that cross bar lines, augmentation dots without notes are printed on the far side of the bar lines.

```
*define bd() [notes off draw dot] &&1-; [notes on]

draw dot
  0 headbottom 2 linebottom sub add moveto
  "\mf\?" show
enddraw

time 2/4
[stave 1 treble 1]
ra | &bd(a) r-g |
```

In this example, the macro **bd** ('bar dot') is defined, in order to shorten the input for each dot. Its argument is the note pitch for which a dot is required. The input could be shortened even further by including the previous note and the bar line inside the macro expansion.

The 'tied' note is not actually printed because of the use of **[notes off]** but its pitch is available to the drawing function, which uses it to print a dot character from the music font at the appropriate level on the stave.

37.14 String operators

The **cvs** operator converts a number to a string, typically so that it can be printed. There must be two arguments on the stack: the number to be converted, and an empty string which provides a place to store the converted number. The string may be followed by any of the usual string options. For example, to print the current barnumber at text size 2, centred at the current position:

```
barnumber ""/c/s2 cvs show
```

The two arguments are removed from the stack and the string containing a representation of the number is then pushed back.

The operator **stringwidth** replaces a string on the stack with the horizontal and vertical distances by which the current point would move if the string were printed. Note that the second value is *not* the height of the string, and in most cases is zero. There is an example of the use of **stringwidth** in the section on looping operators below.

The operator **fontsize** replaces a string on the stack with the font size associated with it.

37.15 Drawing subroutines

One drawing program can be called as a subroutine from within another by means of the **draw** operator. The drawing stack and the current graphics state (current position and current path) are passed over to the called routine. For example, to draw two crosses below the stave on either side of a note's position:

```
draw cross
  -4  0 rmoveto 8 0 rlineto
  -4 -4 rmoveto 0 8 rlineto
  stroke
enddraw

draw crosses
  -10 -6 moveto draw cross
   10 -6 moveto draw cross
enddraw

[stave 1 treble 1]
[draw crosses] g
```

The subroutine must be defined before the definition of any drawing functions in which it is called. Subroutines cannot be called recursively (i.e. it is not possible to call a routine from within itself).

37.16 Blocks

A *block* is a portion of a drawing function enclosed in curly brackets. It is used by the conditional and looping operators. When a block is encountered during drawing, its contents are not obeyed immediately. Instead, a reference to them is placed on the stack, for use by a subsequent operator. Blocks can be nested inside each other.

37.17 Conditional operators

The operator **if** is used to obey a portion of the drawing function conditionally. It uses the top two items on the stack. The first must be a number, and the second a reference to a block. Both are removed from the stack, and if the value of the number is zero, nothing else happens. Otherwise, the contents of the block are obeyed. For example, to print the bar number if it is greater than 5:

```
barnumber 5 gt { barnumber "" cvs show } if
```

The bar number and then the number 5 are pushed on the stack; the comparison operator **gt** replaces them with 1 if the barnumber is greater than 5, or 0 otherwise. Then a reference to the block is pushed onto the stack and the **if** operator causes it to be obeyed if the number is non-zero.

The **ifelse** operator is similar to **if**, except that it require two blocks on the stack. The first is obeyed if the condition is true, the second if it is false.

37.18 Looping operators

The **repeat** operator expects a number and a block on the stack. It removes them, and then obeys the block that number of times. If the number has a fractional part, it is ignored. For example, to print a row of asterisks from the start of the bar to just before the current note or bar line, the following function could be used:

```
draw astline
  leftbarx -15 moveto
  leftbarx neg "*" stringwidth pop div
  0.5 add { "*" show } repeat
enddraw
```

Adding 0.5 ensures that the count is rounded to the nearest integer.

The **loop** operator expects only a block on the stack, and it obeys it repeatedly until the **exit** operator is encountered. To guard against accidents, a limit of 1,000 times round the loop is imposed. Another way of printing the asterisks is

```
draw astline
  leftbarx -15 moveto
  { "*" show currentpoint pop 0 ge {exit} if } loop
enddraw
```

The **exit** operator can also be used to stop a **repeat** loop prematurely. If encountered outside a loop, it causes an exit from the current drawing function.

37.19 Drawing in headings and footings

Drawing functions can be obeyed in headings and footings. For example, crop marks and borders on title pages can be drawn by this method. For details, see the description of the **heading** directive in section 38.45.

37.20 Drawing at stave starts

Drawing functions can be obeyed at the start of a stave, as well as, or instead of printing text. For details see the description of the **[stave]** directive in section 47.82.

37.21 Testing drawing code

When a drawing does not turn out the way you expect it to, it can sometimes be difficult to track down exactly what is wrong. Being able to examine the contents of the stack at particular points is sometimes helpful. The operator **pstack** causes the contents of the stack to be written to the standard error stream.

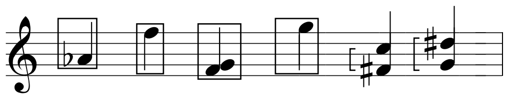
37.22 Example of use of system variables

This example illustrates the use of the variables which contain the dimensions of the note that follows the **[draw]** directive.

```
draw box
  -2 headleft sub accleft sub stembottom 1.3 sub moveto
  stemtop stembottom sub 2.6 add dup 0 exch rlineto
  headleft headright add accleft add 4 add dup 0 rlineto exch
  0 exch neg rlineto
  neg 0 rlineto
  stroke
enddraw

draw bracket
  -2 headleft sub accleft sub headbottom linebottom add moveto
  -2 0 rlineto
  -4 headleft sub accleft sub headtop linetop sub lineto
   2 0 rlineto
   stroke
enddraw

[stave 1 treble 1]
[draw box] $a [draw box] f’ [draw box] (fg)
[space 10] [draw box] (f’g’)
[space 6] [draw bracket] (#fc’) [draw bracket] (g#d’)
[endstave]
```



The definitions look a bit daunting at first sight, but are not difficult to understand when broken down into their constituent parts. If you find the explanation hard to follow, try using pencil and paper to keep track of the values as they are pushed onto and popped off the stack. This is also a good way of developing your own drawings.

We consider first the ‘box’ drawing, which encloses the following note or chord in a rectangular box.

The first line establishes the start of the drawing path at the bottom left-hand corner of the box:

```
-2 headleft sub accleft sub stembottom 1.3 sub moveto
```

It starts by pushing the value -2 onto the stack, then subtracting from it the **headleft** and **accleft** variables. This gives a value for the x-coordinate which is two points to the left of the leftmost accidental, taking into account any notehead which is positioned to the left of the stem. The y-coordinate is computed as the value of the **stembottom** variable less 1.3 points. The **moveto** operator then establishes the start of the drawing path, using the two coordinate values that are on the stack, and leaving the stack empty.

The second line of the drawing instructions,

```
stemtop stembottom sub 2.6 add dup 0 exch rlineto
```

computes the length of the vertical sides of the rectangle. It does this by subtracting the value of **stembottom** from the value of **stemtop** and then adding 2.6 to the result. This is to allow 1.3 points of clear space at the top and the bottom. As this value is going to be needed twice, once for each side, the **dup** operator is called to duplicate it. To draw the left-hand vertical, a relative x-coordinate of zero is pushed on the stack, and then **exch** is used to get the coordinates in the correct order on the stack before calling **rlineto**.

The current point is now at the top left-hand corner of the rectangle, and the stack contains the duplicated value of the vertical sides’ length. The third line,

```
headleft headright add accleft add 4 add dup 0 rlineto exch
```

does a computation for the rectangle’s width, which is computed as the sum of the contents of the **headleft**, **headright**, and **accleft** variables, plus four (allowing two points clear on either side). Once again, **dup** is used to leave a copy of the value on the stack, and this time a zero relative y-coordinate is used, in order to draw a horizontal line. The two remembered lengths that are left on the stack are now exchanged, so that the vertical length becomes the topmost value.

The remaining lines use these stacked values to complete the rectangle:

```
0 exch neg rlineto
neg 0 rlineto
stroke
```

The first line pushes a zero relative x-coordinate, ensures that the order on the stack is correct by means of **exch** (bringing the vertical side length to the top), and negates the y-coordinate so that the line is drawn downwards.

The second line negates the one remaining value on the stack, which is the width of the rectangle, pushes a zero relative y-coordinate, and draws the final horizontal line to the left. Finally, **stroke** causes a line to be drawn along the path which has just been defined.

The ‘bracket’ drawing draws a left-hand bracket whose size is adjusted for the notes of a chord, and which also takes into account the position of the noteheads on stave lines or in spaces.

```
-2 headleft sub accleft sub headbottom linebottom add moveto
-2 0 rlineto
-4 headleft sub accleft sub headtop linetop sub lineto
 2 0 rlineto
 stroke
```

The first line computes the position of the start of the path, which is the right-hand end of the bottom ‘jog’. The x-coordinate is 2 points to the left of the left-most accidental, while the y-coordinate is the bottom of the lowest notehead if this position is not on a stave line (in which case **linebottom** is zero) or two points above if it is.

The second line draws the lower horizontal ‘jog’ to the left as a relative line. The third line computes the absolute coordinates of the top left-hand corner, taking into account whether the top notehead is on a line or not. An alternative to this would have been to save the initial x-coordinate on the stack instead of recomputing it from scratch. Finally, the top ‘jog’ is drawn to the right, and the path is stroked.

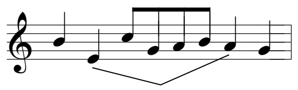
37.23 Example of inter-note drawing

This example illustrates the use of the **originx** variable for connecting up two different notes:

```
draw save
  headbottom originx
enddraw

draw connect
  originx sub 3 add dup 3 add 2 div
  3 1 roll exch 2 sub moveto
  -12 lineto
  3 headbottom 2 sub lineto
  stroke
enddraw

[stave 1 treble 1]
b [draw save] e c'-g-a-b-
[draw connect] a g |
[endstave]
```



The ‘save’ drawing doesn’t actually do any drawing at all. It simply saves on the stack the coordinate of the bottom of the next note, and the absolute coordinate of its left-hand edge. Using the stack to pass data between two drawing functions is a simple method that works well when both functions are called in the same bar on the same stave. An alternative method is to use user variables (see section 37.12); this must be used if the drawing functions appear on several different staves and the related functions are not called in the same bar.

The first thing the ‘connect’ drawing program does is to push *its* origin onto the stack, and subtract it from the saved origin. The result of this computation is the x-coordinate of the first note (the one immediately following **[draw save]**), relative to the current local coordinate system, which is, of course, based on the note following **[draw connect]**.

A value of 3 is added to this, giving the horizontal position of the middle of the first note. The **dup** operator saves a copy of this value on the stack for later use, while another 3 is added to the top value, giving the coordinate of the right-hand edge of the first note.

The next bit of computation is to find the mid-point between the two notes. The left-hand edge of the second note has an x-coordinate of zero in the local coordinate system, so simply dividing the coordinate of the right-hand edge of the first note by 2 gives us the mid-point. There are now three values on the stack:

the x-coordinate of the halfway point
the x-coordinate of the mid-point of the first note
the y-coordinate of the bottom of the first note

The operation ‘3 1 **roll**’ changes this to

the x-coordinate of the mid-point of the first note
the y-coordinate of the bottom of the first note
the x-coordinate of the halfway point

and the subsequent **exch** changes it to

the y-coordinate of the bottom of the first note
the x-coordinate of the mid-point of the first note
the x-coordinate of the halfway point

A value of 2 is subtracted from the y-coordinate of the first note, and the **moveto** operator is called to start the drawing path, which therefore begins two points below the first note, and halfway along its notehead. Now only the x-coordinate of the halfway point between the two notes remains on the stack.

The operation ‘-12 **lineto**’ draws a line from the initial position to the halfway point, twelve points below the bottom of the stave. The stack is now empty. The final lines of the drawing program continue the path to a position two points below the end note, and at the mid-point of its notehead, and then cause it to be stroked.

38. Heading directives

The information given in the heading section of the input to PMW is in the form of *heading directives*. Each takes the form of a keyword, and some of them must be followed by numerical or other data. Each keyword may appear on a new line, but this is not necessary.

In general, they may appear in any order, but there are some obvious cases where the order matters. For example, a multiplicative change to the note spacing must follow a setting of absolute values, the definition of a drawing or macro must precede its use, and a stave selection must precede any tests based on which staves are selected.

A number of heading directives are followed by a list of numbers. In all cases, such numbers can be separated by commas or spaces, or commas and spaces, except when such a list is continued onto a subsequent input line, when the final number on the first line must be terminated by a comma.

The heading section is terminated by the first square bracket character in the file (excluding any in text strings or comments).

None of the heading information is mandatory, as there are default values for all the parameters; the heading section of the file may be completely empty.

Most of the heading directives may appear at the start of a new movement as well as at the start of the input; a few may only appear at the very start of the file. See chapter 24 for details.

The heading directives are described in this chapter in alphabetical order. Those that are restricted to appearing at the start of a PMW file only are marked with an asterisk. The remainder may also appear at the start of a new movement within a PMW file. Those directives that are marked with a dagger (†) set values which are reset at the start of a new movement. The effects of the others continue into subsequent movements.

38.1 Accadjusts

Accidentals are normally printed about four points to the left of the notes to which they apply (the exact distance depends on the accidental). The **accadjusts** directive can be used to vary this positioning, on a note-type basis. It does not affect the spacing of the notes themselves, simply having the effect of moving the accidentals right or left. The directive is followed by up to eight numbers, which apply to each of the eight note types, starting with breves. The numbers can be positive (move to the right, i.e. nearer the note) or negative (move to the left, i.e. further from the note). For example,

```
accadjusts 1.8
```

has the effect of moving any accidental which precedes a breve 1.8 points to the right.

38.2 Accspacing

The **accspacing** directive must be followed by five numbers. These give the printing widths of the accidental characters in the order double sharp, flat, double flat, natural, and sharp. The default values are equivalent to

```
accspacing 5.25 4.5 8.0 4.25 5.0
```

It should not be necessary to change these widths unless a non-standard music font is being used.

38.3 Bar

This directive must be followed by a number, and it causes the numbering of bars to begin from a number other than one. This facility is useful for printing parts of pieces, or continuing a bar number sequence through several movements.

38.4 Barcount

By default, PMW allocates its internal tables in such a way as to make room for 500 bars of music per stave per movement. If the music in any movement is longer than this, the **barcount** directive must be used to increase the size of these tables. Its parameter is the maximum number of bars in the current and subsequent movements.

38.5 Barlinesize

When a a system contains staves of differing sizes (as set by **stavesizes**) it is usually the case that bar lines are split where the stave size changes, so the use of barlines of differing thicknesses for the different staves looks reasonable. To cope with the rare case when barlines must cover staves of different sizes, the **barlinesize** directive exists, because PMW cannot decide for itself what the size should be – the default use of different sizes leads to jagged joins.

Barlinesize takes a single number which specifies the relative size of bar line to be used throughout. A value of zero requests the default action of using different sizes. The directive

```
  barlinesize 1
```

requests that full size barlines be used throughout; they will look somewhat fat on any staves whose size is less than 1.

38.6 Barlinespace

This directive allows control over the amount of space left after bar lines. It must be followed by a single dimension, which may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value.

The default is related to the space following a minim, with a minimum of 3 points. However, if an explicit space is specified, no minimum is enforced. A value of zero may be given – this is useful when printing a piece with no bar lines, where 'invisible bar lines' can be used to tell PMW where lines can be broken, but no space must be inserted.

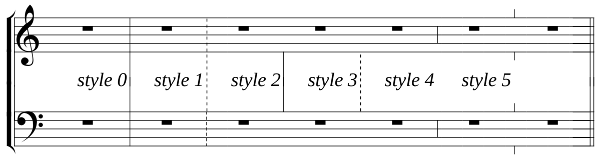
Any value set by **barlinespace** persists over movement changes. The default value can be reset by following the directive with an asterisk.

38.7 Barlinestyle

This directive specifies the way in which bar lines are to be printed on all staves. It takes a numerical parameter, with a value in the range 0–5. There is also a **[barlinestyle]** stave directive which sets the style separately for an individual stave. The styles are as follows:

- Style 0 is the normal style, using solid bar lines.
- Style 1 prints with dashed bar lines.
- Styles 2 and 3 cause solid or dashed bar lines (respectively) to be drawn *between* staves only; if the bar line is broken at a stave where either of these styles applies, nothing at all is printed. These styles work only when the stave spacing is 32 points or greater (which is normally the case).
- Style 4 causes a half-height bar line to be printed in the middle of the stave. It implies a bar line break at any stave where it is used.
- Style 5 causes two very short stub lines to be drawn, above and below the stave. It implies a bar line break at any stave where it is used.

Specifying a double bar by inputting `||` overrides the stave or movement bar line style, which can also be overridden by inputting a digit immediately after the vertical bar character. The following example shows the six available styles:



Note that the **breakbarlines** directive can be used to specify breaks in bar lines at particular staves when style 0 or 1 is used, and an individual bar line may be broken by using **[breakbarline]**.

38.8 Barnumberlevel

The heading directive **barnumberlevel** is provided to adjust the level of all the bar numbers in a piece. It must be followed by a plus or a minus sign and a dimension. For example,

```
barnumberlevel +4
```

prints all bar numbers four points higher up the page than they would otherwise have been.

38.9 Barnumbers

This directive specifies that the bars of the piece are to be numbered. Several different options are available, the general form of the directive being

`barnumbers` <*enclosure*> <*interval*> <*fontsize*> <*fontname*>

The first parameter, which is optional, must be one of the words 'boxed' or 'ringed'. These specify that barnumbers are to printed inside rectangular boxes, or roughly circular rings, respectively. If neither word is given, the numbers are printed without any special identification.

The second parameter must be present, and is either the word 'line' or a number. If 'line' is given, it causes a bar number to be printed over the first bar of every line of music except the first; if a number is given, it specifies the interval between bar numbers. Thus, for example,

```
barnumbers boxed 10
```

causes a bar number, enclosed in a box, to be printed every 10 bars.

The third parameter is optional; it specifies the size of the font in which the numbers are printed. The default size is 10 points. For example,

```
barnumbers line 8.5
```

numbers the bars at the start of each system, using a font of size 8.5 points.

The final parameter, which is optional, specifies the font (typeface) for printing the bar numbers. The default is roman, but any of the standard font names can be used. For example,

```
barnumbers 5 9/1.1 italic
```

prints bar numbers every 5 bars in a 9-point italic font, horizontally stretched by 1.1.

The automatic bar numbering that is set up by this heading directive can be overridden for individual bars by means of the stave directive **[barnumber]**, which is described in section 47.9.

In certain circumstances it may be necessary to prevent a bar in the middle of the piece from being counted, for example, when using an 'invisible bar line' to make PMW split a bar over two lines on the page. There is a stave directive **[nocount]** to request this. If a piece starts with an incomplete bar, PMW counts it for the purposes of bar numbering, unless it contains the **[nocount]** stave directive, which should therefore normally be used in 'upbeat' bars.

38.10 Beamendrests

This directive, which has no parameters, requests PMW to include rests at the ends of beams within the beams. For details, see section 42.3.

38.11 Beamflaglength

The length of short, straight note flags that are used with beams (e.g. for a semiquaver beamed to a dotted quaver) can be set by this directive. The default is 5 points; it scales with the relative stave magnification.

38.12 Beamthickness

This directive takes a single dimension as its parameter; it sets the thickness of the lines drawn for beaming notes together. The default thickness is 1.8 points. On some printers and at some magnifications a better effect can be obtained by changing the thickness (normally by making it smaller). The thickness should not be set greater than 2.5 points, as otherwise beams will not be correctly printed.

38.13 Bottommargin and topmargin

The **bottommargin** and **topmargin** directives make it possible to reserve white space at the top or bottom of a page, provided that there is room on the page to do this after the systems have been fitted onto it. Effectively, these directives give some additional control over the vertical justification action described in the section 38.49 below, once the pagination of a piece is determined.

The parameter for each of these directives is a dimension. They set default values for the vertical margins for each page. For example,

```
topmargin 20
bottommargin 5
```

The values can be changed for an individual page by means of the **[topmargin]** and **[bottommargin]** directives. The default values for the margins are zero for the bottom margin and 10 points for the top margin.

The use made of the supplied values depends on the justification mode for the page. The phrase 'the contents of the page' below excludes any text that is defined as a footing or as a page heading, but includes start-of-piece headings.

- If the justify mode is 'top' only, then the contents of the page are moved down by the top margin, provided there is enough room on the page to do this. If not, the contents of the page are moved down as far as possible. The bottom margin value is ignored.
- If the justify mode is 'bottom' only, then the contents of the page are moved up by the bottom margin, provided there is enough room on the page to do this. If not, the contents of the page are moved up as far as possible. The top margin value is ignored.
- If the justify mode is both 'top' and 'bottom' then the amount of space available for spreading the systems vertically is decreased by the sum of the top margin and the bottom margin, and the contents of the page are moved down by the top margin, provided there is enough spreading space available. If there is insufficient spreading space, it is divided *pro rata* between the top margin and the bottom margin, the systems are not spread at all, and the contents of the page are moved down by the adjusted top margin value.
- If the justify mode is neither 'top' nor 'bottom' then both values are ignored.

The effect of using these directives is to allow more of the page to be used when necessary, but to keep the systems nearer the centre of the page when there is a lot of space left over.

38.14 Brace and Bracket

These two directives specify which staves in the system are to be joined by brackets and/or braces. A bracket is traditionally used for groups of independent instruments or voices, while a brace is reserved for pairs of staves that apply to a single instrument, frequently a keyboard. (See also the **thinbracket** directive, which specifies another kind of bracket.)

Each of these directives must be followed by a list of pairs of stave numbers, the members of each pair being separated by a minus sign, with the pairs themselves separated by spaces and/or commas. For example,

```
bracket 1-4,5-7
brace 8-9
```

specifies that staves 1–4 and 5–7 are to be joined by brackets, while staves 8 and 9 are to be joined by a brace. In addition to these marks, the entire system is by default joined by a single vertical line at the left-hand side. (See the **join** and **joindotted** directives for ways of changing this.)

Occasionally a bracket is required for a single stave within a system; this may be specified by giving just one stave number. The effect can also occur if all but one of a bracketed group of staves is suspended. By contrast, a brace is never printed for just one stave.

If only a single stave is selected for printing, for example, when a part is being extracted from a full score, these directives are ignored; no marks precede the clef on a single stave in this case.

The default action of PMW is to join all the staves with a single bracket. If no brackets of any kind are required, it is necessary to suppress this by including a **bracket** directive with no parameters.

38.15 Bracestyle

The default brace shape curves a lot at the ends and almost touches the staves. An alternate form which does not curve so much at the ends can be selected by specifying

```
  bracestyle 1
```

38.16 Breakbarlines

By default, PMW draws bar lines through all the staves of a system without a break. The **breakbarlines** directive specifies after which staves there is to be a break in the bar lines. It is followed by a list of stave numbers. For example,

```
  breakbarlines 3 6 8
```

specifies that there is to be a vertical break in the bar lines after staves 3, 6 and 8. Two numbers separated by a minus sign can be used to specify breaks for a sequence of staves. For example,

```
  breakbarlines 1-4
```

Breakbarlines can also appear with no numbers after it at all; in this case there is a break after every stave.

If **breakbarlines** is specified at the start of a new movement, it must list all the staves at which a break is required. If it is not given, breaks carry over from the previous movement.

The stave directives **[breakbarline]** and **[unbreakbarline]** can be used to override the setting for individual barlines on a given stave.

38.17 Breakbarlinesx

The **breakbarlinesx** directive acts exactly as **breakbarlines**, except that the final bar line of each system on the page is *not* broken, but is drawn solid right through the system.

38.18 Breveledgerextra

This directive specifies the number of points of extra length that ledger lines for breves have at either end. The default value is 2.3.

38.19 Breverests

By default, PMW prints a semibreve rest sign for a complete bar's rest, whatever the time signature. This heading directive changes its behaviour so that the notation used for a whole bar rest depends on the number of crotchets in the bar:

- If there are 8 crotchets (4/2 or 2/1 or 2*C etc.) then a breve rest sign is used.
- If there are 12 crotchets (6/2 or 12/4 or 2*3/2 etc.) then a dotted breve rest sign is used.
- If there are 6 crotchets (3/2 or 2*3/4 etc.) then a dotted semibreve rest sign is used.
- Otherwise a semibreve rest is used.

38.20 Caesurastyle

The default caesura 'character' is two slanting lines through the top line of the stave. This directive can be used to specify an alternative. It is followed by one of the following numbers:

0	default style
1	one slanting line only

38.21 Check

This directive, which has no parameters, can be used to override an occurrence of **nocheck** in a previous movement.

38.22 Checkdoublebars

This directive, which has no parameters, can be used to override an occurrence of **nocheckdoublebars** in a previous movement.

38.23 Clefsize

By default, new clefs that appear in the middle of lines of music are printed at the same size as clefs at the left-hand side. This directive is used to specify a different relative size for such clefs. For example,

```
  Clefsize 0.7
```

specifies that such intermediate clefs are to be printed at 0.7 times the normal size.

38.24 Clefstyle

Some early editions use for the F-clefs and 𝄡 for the C-clefs. The **clefstyle** directive makes it possible to reproduce this usage. It takes a single numerical argument, with the following values:

0	all modern clefs
1	old-fashioned F clefs
2	old-fashioned C clefs
3	old-fashioned F and C clefs

The graphic is wider than the modern 𝄢 shape. Printing has been arranged so that two dots appear in the same place in both cases. This means that the old-fashioned clef extends further to the left than the modern one, and with PMW's default settings, it runs into stave joining lines and brackets. Therefore, when using old-fashioned F clefs, the **startlinespacing** directive should be used to insert at least 2 points of space before the clefs.

38.25 Clefwidths

When it is laying out a system, PMW inspects the clefs of all the staves, and positions the key signature immediately to the right of the widest clef. When the clefs change between systems, it can happen that the key signatures do not all line up vertically on the page, and some people want that to happen. Unfortunately, it is not easy to arrange for PMW to do this automatically, because it does the layout in a single pass through the input, and so does not know what clef arrangements lie ahead. However, the **clefwidths** directive is provided to enable this to be done manually. **Clefwidths** specifies the widths to be used for each type of clef when computing where to put key signatures. The directive is followed by up to five numbers, which specify the widths of the G-clef, F-clef, C-clef, H-clef, and no clef, respectively. The default settings are equivalent to

```
  clefwidths 13 16 15 15 0
```

The values given must be whole numbers (no fractions are allowed). For example, in a piece which has treble and bass clefs in some systems and only treble clefs in others, a setting such as

```
  clefwidths 16 16
```

would ensure that all the key signatures line up.

38.26 Copyzero

This directive makes it possible to have copies of stave zero printed over any number of staves. It is followed by a list of stave numbers, each of which may be optionally followed by a slash and a dimension. Details of the use of **copyzero** are given in chapter 35 (*Stave 0*).

38.27 Cuegracesize

This directive, which takes a single number as a parameter, specifies the font size to be used when printing grace notes in bars containing cue notes. See the **[cue]** directive for further details.

38.28 Cuesize

This directive, which takes a single number as a parameter, specifies the font size to be used when printing cue notes. See the **[cue]** directive for further details.

38.29 Dotspacefactor

This directive specifies the factor by which the horizontal space after a dotted note is extended. The default value is 1.2. If, for example,

```
dotspacefactor 1.5
```

is specified, then the amount of space which follows a dotted note is 1.5 times the amount that would follow an undotted note of the same type. (Of course, when several staves are involved, the value is a minimum, as the notes on the other staves may cause additional spacing.)

When a note is double-dotted, half as much space again is added. Thus in the default case a double-dotted note occupies 1.3 times the space of an undotted note.

38.30 Doublenotes

This directive, which applies to the current movement only, causes the length of each note to be doubled. It also affects time signatures as follows:

- `C` and `A` are turned into `2*C` and `2*A`, that is, they are printed as before, but the bar length is doubled.
- Other time signatures are changed by halving the denominator, unless the denominator is 1, in which case the numerator is doubled instead.

For example, 4/4 becomes 4/2, but 4/1 becomes 8/1. See also **halvenotes**.

38.31 Draw

The **draw** directive is used for defining simple drawn shapes that are to be printed with music on staves. A full description of this facility is given in chapter 37.

38.32 Endlinesluradjust and endlinetieadjust

When a slur or a tie is continued onto the next line, the first part is normally drawn right up to the end of the first line. Some editors prefer it to stop a little short of this; **endlinesluradjust** and **endlinetieadjust** specify a dimension that is added to the right-hand end of such slurs and ties, respectively. Normally the value given is a small negative dimension. The value for ties also applies to glissandos.

38.33 Endlineslurstyle and endlinetiestyle

Each part of a continued slur or tie is normally drawn as a complete slur, that is, with both ends tapering to a point, which is the most commonly found style. Some editors, however, prefer each portion to have the appearance of half a normal slur. **Endlineslurstyle** and **endlinetiestyle** specify this behaviour when style 1 is selected. The default is style 0.

38.34 Extenderlevel

The vertical level of extender lines which are drawn when the last syllable of an underlaid or overlaid word extends over several notes can be altered by this directive, which takes a positive or negative number as its parameter. This specifies a number of points, positive numbers moving the lines up, and negative ones down. Extender lines are output by printing underscore characters, and the default level is just below the baseline of the text. Specifying

```
extenderlevel 1
```

moves the extender lines up to near the baseline, while larger values can be used to place them nearer the middle of the text letters.

38.35 Fbsize

By default, text which is specified as being part of a figured bass is printed at the same size as other textual items (10 points). This directive enables a different point size to be chosen for the figured bass font. For example,

```
fbsize 8.2
```

Individual figured bass text strings may have an explicit size specified (see chapter 44).

38.36 Footing

This directive has the same parameters as **heading**, and they have the same meaning. See **heading** below for a full description. **Footing** sets up text to be printed at the foot of the first page only, for example

```
footing "Copyright \c) 1992 J.S. Bach"
```

Note the use of the escape sequence `\c)` to obtain a copyright symbol. If a type size parameter is not given, 8-point type is used.

Several footing lines may be specified. They are printed below the bottom of the page, as specified by the **pagelength** directive, the first one being 20 points below. This is an absolute distance that does not change if the magnification is altered. However, the distance between footings and the sizes of fonts used are subject to magnification.

As is the case with headings, if the left-hand part of a footing (the text before the first | character) is longer than the line length, it is split up into as many lines as necessary, and all but the last are fully justified.

As the first footing line on a page is always at the same fixed vertical position, the drawing facility (as described for **heading**) can be used for drawing fixed marks on the page. For example, crop marks and borders on title pages can be drawn by this method.

Setting any footing line for a new movement automatically cancels all the footings that were in force for the previous movement.

See the **pagefooting** and **lastfooting** directives for a means of setting up footings for pages other than the first.

If no **footing** directive is present, then the text specified by **pagefooting** is printed on the first page as well as on subsequent pages. If the piece is only one page long, **footing** overrides **lastfooting**, but **lastfooting** overrides **pagefooting**.

38.37 Footnotesep

This directive specifies the amount of vertical white space to leave between multiple footnotes on the same page. The default is 4 points. See the **[footnote]** directive for a full description of footnotes, which should not be confused with footings.

38.38 Footnotesize

This directive sets the type size used for printing footnotes. The default size is 9 points.

38.39 Gracesize

The default size of the musical font used for printing grace notes is 7 points. This directive allows a different size to be chosen. It must be followed by a number specifying a point size for the font.

38.40 Gracespacing

By default, a grace note is printed 6 points to the left of the note which follows it. This directive allows this value to be changed. The parameter is a dimension which may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value.

38.41 Gracestyle

When two or more staves are being printed, and a note on one stave is preceded by one or more grace notes, then the notes on the other staves which fall at the same point in the bar are printed directly above or below the main note, leaving the grace notes sticking out to the left. This is, of course, the conventional practice in modern music. The **gracestyle** directive can be used to make PMW behave differently. It is followed by a number:

`0`	default style
`1`	align with first grace note

When the style is set to 1, the notes which are not preceded by grace notes are aligned with the first grace note on other staves. In addition, if underlaid text is present, it is aligned to start at the first grace note instead of being centred on the main note.

This facility can be used, in combination with setting the grace note size equal to the main note size, and using notes with no stems (see **[noteheads]**), to print some forms of plainsong music.

38.42 Hairpinlinewidth

This directive specifies the width of line used to draw hairpins. Its parameter is a width in points. The default width of hairpin lines is 0.2 points.

The number may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value.

Making hairpin lines thicker may help alleviate jagged effects on long hairpins printed on high resolution printers.

38.43 Hairpinwidth

This directive specifies the vertical width of the open end of crescendo and decrescendo ‘hairpins’. Its parameter is the number of points. For example,

```
hairpinwidth 5.6
```

The number may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value. The default value for this parameter is 7 points.

38.44 Halvenotes

This directive, which applies to the current movement only, causes the length of each note to be halved. It also affects time signatures as follows:

- `C` and `A` cannot be halved. The signatures `2*C` and `2*A` can be halved, and turn into `C` and `A` respectively.
- Other time signatures are changed by doubling the denominator.

For example, 4/4 becomes 4/8. See also **doublenotes**.

38.45 Heading

The **heading** keyword defines a line of text to be printed as a heading to the piece or movement. Any number of occurrences of this keyword may be given. There are two forms which the **heading** directive can take:

In the more common form, the keyword may be followed by up to three parameters:

`heading` <*fontsize*> `"`<*text*>`"` <*depth*>

The first parameter is a number, and is optional. If present, it defines the size for this heading, in printer’s points. As for all font size sizes, an aspect ratio and/or shear angle may be specified as well as the basic size. If this parameter is omitted, default sizes are used.

For headings at the start of the piece the default sizes are 17 points for the first heading line, 12 points for the second, 10 points for the third, and 8 points for the fourth and subsequent heading lines.

For headings at the start of a new movement the default sizes are 12 points for the first heading line, 10 points for the second, and 8 points for the third and subsequent heading lines.

The second parameter is a string in double-quotes, and must be present. It defines the contents of the heading. The vertical bar character has a special meaning in this text – it splits it up into left-hand, centred and right-hand parts. Characters to the left of the first vertical bar are printed at the left of the page; characters between the first and second vertical bars are centred on the page; the rest of the text is printed at the right of the page.

If the left-hand part of the text is longer than the line length, it is split up into as many lines as necessary. All but the last line are fully justified, by expanding any spaces they contain. The last line is also justified if it is nearly as long as the line length. Justification does not take place when there are no spaces in the text.

This facility makes it possible to print paragraphs of introductory text on title pages or at the start of pieces or movements. Note, however, that PMW does not set out to be a fully-fledged wordprocessor. Any special characters required in the text (for example, the ‘ﬁ’ ligature) have to be coded explicitly (see chapter 34); they are not provided automatically.

The paragraph mechanism should not be used with text that contains variable data such as the escape sequence for the current page number, because the splitting and justification happens only once, when the heading directive is read in.

Note that heading strings do not need to be input on a single line; line breaks in the string are treated as spaces.

The third parameter of **heading** is a number and is optional. If present, it specifies the number of points of vertical space to leave after the heading. It may be zero; this can be useful for printing headings of different sizes on different parts of the line. It may also be negative; this can be used with an empty text string to make PMW start printing higher up the page than it normally does. If the parameter is omitted, the amount of space left after the heading line is equal to the point size of the text.

For the last heading line, the space specified (or defaulted) is the space between the base line of the heading text and the top of the first stave of music.

When a heading string is split up by PMW into several lines, the spacing value given (or defaulted) is used for the space after each line in the paragraph. To leave space between paragraphs, a heading containing an empty string can be used.

Here are some examples of this form of the **heading** directive:

```
Heading "|Partita"
Heading 11 "Moderato||J.S. Bach" 14
Heading "" 20
```

The third example prints nothing, but leaves 20 points of space. If no headings are given, no space is left at the top of the first page.

The second form of the **heading** directive causes a drawing subroutine to be obeyed at the next heading position (see chapter 37 for more details). The syntax is:

`heading draw` <*argument(s)*> <*name*> <*optional space*>

Arguments are optional. The definition of the drawing must precede such a heading line in the input file. If no space is given, no vertical movement is made following the drawing. The origin of the coordinate system is set at the left-hand side, at the level of the next heading line. For example, to draw a line right across the page after a heading:

```
draw rule
  0 0 moveto
  0 linelength rlineto
  1 setlinewidth stroke
enddraw

heading "|Some Text" 0
heading draw rule 20
```

As the first heading or footing line on a page is always at the same fixed vertical position, it can be used for drawing fixed marks on the page. For example, crop marks and borders on title pages can be drawn by this method. See the file **cropmarks** in the **Contrib** directory on the PMW distribution disc for an example of this.

See the **pageheading** directive for a means of setting up headings for pages other than the first.

38.46 Hyphenstring

When PMW is outputting rows of hyphens between underlaid syllables, it normally uses single hyphen characters. This directive can be used to change this. The parameter is specified as a string for generality, but normally only a single character would be used. For example, longer lines than the default can be obtained by the use of en-dash characters instead of hyphens. These are specified in strings by a backslash escape and two successive minus signs:

```
hyphenstring "\--"
```

See section 45.5 for other facilities for controlling exactly what is printed between underlaid syllables.

38.47 Hyphenthreshold

PMW automatically inserts hyphens between syllables of underlaid text. When the distance between the syllables is less than the 'hyphen threshold', a single hyphen is printed, centred in the space (unless the syllables are so very close together that there is no room for even one hyphen). If the space is greater than the threshold, a number of hyphens are printed, the distance between them being the threshold value divided by three. The default value for the hyphen threshold is 50 points.

The number following this directive may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value.

38.48 Join and joindotted

The syntax of these directives is the same as for **bracket** and **brace**. They control the joining line at the left-hand edge of systems. By default a solid line is drawn through all staves; these directives can be used to cause breaks in the line and/or to print a dotted line. For example,

```
join 1-2, 3-4
```

causes solid lines to be drawn joining staves 1 and 2, and 3 and 4, leaving a gap between staves 2 and 3, while

```
join 1,2,3,4
```

causes solid lines to be drawn at the ends of each stave, but gaps to be left between the staves. **Join** and **joindotted** can be used together:

```
joindotted 1-2
join 1,2
```

causes a dotted line to be used to join staves 1 and 2, and solid lines to overprint this at the ends of each stave, leaving the dotted line between them.

38.49 Justify

Justify sets the horizontal and vertical justification parameters. It can be followed by one or more of the words 'left', 'right', 'top', 'bottom', or 'all'. Each occurrence of the **justify** heading directive completely re-specifies the justification parameters, in contrast to the stave directive **[justify]**. An appearance of **justify** which is not followed by one of the above words requests no justification in any direction. The default value for justification is 'all', that is, complete vertical and horizontal justification.

The effect of the horizontal parameters is as follows:

- When 'left' is specified without 'right', each musical system starts at the left-hand side of the page, but is *not* stretched out to the right-hand side. This is not normal for performing music, but is useful when printing examples for inclusion in other documents.
- When 'right' is specified without 'left', each musical system ends at the right-hand side, but is *not* stretched to start at the left.
- When both 'left' and 'right' are specified (the default), each musical system begins at the left-hand side of the page, and is stretched so that it ends at the right-hand side, unless it is less than 1/2 of the linelength, in which case the behaviour is as if 'right' were not specified.
- When neither 'left' nor 'right' is specified, each musical system is centred horizontally on the page. The width of page used for this purpose can be adjusted by the **sheetwidth** directive. This is another style of printing that is useful for examples and illustrations.

The effect of the vertical justification parameters exactly parallels that of the horizontal ones.

- When 'top' is specified without 'bottom', systems are printed starting at the top of the page, using the system gaps specified in the input.

- When ‘bottom’ is specified without ‘top’ the systems are again printed with the gaps specified, but this time they are so arranged that the final stave on the page is exactly at the page depth. This form is useful when generating PostScript files for inclusion in other PostScript documents.
- When both ‘top’ and ‘bottom’ are specified, the first system is printed at the top of the page. If there is more than one system on the page, and if more than 1/2 of the page depth has been used, then additional space is inserted between the systems so that the final stave is exactly at the page depth, except that there is a maximum amount of space that PMW is prepared to insert between any two systems.
- When neither ‘top’ nor ‘bottom’ is specified, then the systems are printed with the gaps specified, but the set of systems is centred vertically on the page.

The maximum amount of vertical space that PMW is prepared to insert between any two systems is controlled by a heading directive called **maxvertjustify**. The default value is 60 points which means that if the default system gap is in force, the furthest apart any two systems can be is 104 points. To ensure that the bottom stave is always at the bottom of the page under all circumstances, specify a large value for **maxvertjustify**.

In its information window, after it has listed the layout of bars on a page, PMW outputs the amount of space left. When vertical justification is happening, it is this amount of space which is inserted between systems or at the top of the page. When space is being inserted between systems, the same amount is inserted between each pair of systems.

Page headings, footings, and page footings are not affected by vertical justification. However, if ‘top’ is not specified, start of piece headings are moved down the page.

If a new movement starts in the middle of a page which is stretched vertically, additional space is inserted before the start of the movement, that is, before its headings (if any), but not between its headings and its first stave.

The justification parameters persist from one movement to the next, but may be reset by the appearance of **justify** at the start of a new movement. They can also be changed by the appearance of the stave directive **[justify]**. Unlike the heading directive, it specifies *changes* in the justification parameters only, and its effect lasts only until the end of the current movement.

See also the **topmargin** and **bottommargin** directives for further parameters that control the layout of pages.

38.50 Key †

This directive sets a key signature for the piece. Naturally, it is also possible to set keys on individual staves and to change them in the middle of the music. Setting the key in the heading is a convenient shorthand. The single parameter to the directive is a key signature in the format described in chapter 28. For example,

```
key A$
key BM
```

If no key signature is given, the default is C major.

38.51 Keysinglebar and keydoublebar

PMW prints a double bar line before a change of key by default. The **keysinglebar** directive can be used to request a single bar line instead; **keydoublebar** can be used to reset the default for a new movement.

38.52 Keywarn

This directive can be used at the start of a new movement to cancel the effect of **nokeywarn** in the previous movement.

38.53 Landscape *

This directive causes printing to take place in ‘landscape’ format, that is, with the long side of the paper horizontal. It causes the value of **linelength** to be interchanged with the value of **pagelength**, and likewise the value of **sheetwidth** to be interchanged with **sheetdepth**. Subsequent directives affect the new values.

38.54 Lastfooting

This directive specifies footing material for the last page of music, replacing any text specified with **footing** or **pagefooting** for that page. The parameters are exactly as for **heading** or **footing**, but long strings are not automatically split up into multiple lines.

Lastfooting can also be used to specify a special footing for the last page of an individual movement in a piece that has several movements. For details, seen the **[newmovement]** directive.

38.55 Layout

The **[newline]** and **[newpage]** directives are useful for occasional overriding of PMW's default layout choices. However, if a particular layout for the entire piece is required, achieving it with **[newline]** and **[newpage]** is a bit tedious.

The **layout** directive makes it possible to specify exactly how many bars are to be placed in each system, and how many systems are to fit on each page. It its simplest form, it is followed by a list of numbers, separated by commas or white space. If the list is long, it can be split into several lines of input.

Each number in the list specifies the number of bars in a system. If there are more systems than numbers, the list is restarted from the beginning. Thus, the directive

```
layout 4
```

specifies that every system in the piece (except possibly the last one) is to contain exactly four bars.

If page breaks are required at particular points in the piece, they are specified by a semicolon in the layout list. For example,

```
layout 4, 3; 5, 4, 4;
```

specifies two pages, the first containing two systems and the second three systems. If too many systems are specified for a page, PMW inserts a page break of its own.

If the width of the bars specified for a system is greater than the linelength, they are simply compressed until they fit; if too many are specified this can result in very cramped output.

If the same item or sequence of items is repeated in a layout list, it can be enclosed in round brackets and preceded by a repeat count. Brackets can be nested. For example,

```
layout 3(4, 5(6);)
```

defines three pages, each consisting of one system of four bars followed by five systems of six bars. Note the difference between the following two examples:

```
layout 2(4,3;)
layout 2(4,3);
```

The first specifies two pages; the second specifies one page containing four systems.

The **layout** directive applies only to the movement in which it appears. If a subsequent movement contains no **layout** directive, then PMW fills its systems according to the width of the bars.

38.56 Leftmargin

Normally, PMW centres music horizontally on the page. The width of page used for this purpose can be specified by **sheetwidth**. This directive can be used to specify a particular left-hand margin instead.

38.57 Linelength and pagelength

The **linelength** and **pagelength** directives specify the size of the output on the paper; each takes a single parameter which is a dimension in points. The number may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value.

The default line length is 480 points and the page length is 720 points. These values leave generous margins all round on an A4 page.

These two dimensions, together with **sheetwidth** and **sheetdepth**, are the only ones that are not affected by magnification. They define that part of the page on which printing is to occur, in absolute terms.

38.58 Longrestfont

The font used for the number that is printed above a multi-bar rest sign can be set by means of the this directive. Its arguments are an optional size followed by a font name. For example,

```
  longrestfont 13 bolditalic
```

The default size is 10 points; if this directive is not used, the numbers are printed in the roman font.

38.59 Magnification *

This directive causes all the output to be magnified or reduced. For example,

```
  magnification 1.5
```

causes the output to be one and a half times as large as the default, while

```
  magnification 0.5
```

makes it only half the size. All the dimensions in the PMW system are subject to the magnification factor, *except* the line length, page length, sheet width, and sheet depth, which are absolute values, and which are therefore not affected by magnification.

Magnification or reduction can be useful in fitting a piece onto the paper. For example, if in the default state the last page contains only two bars, then a small reduction may enable the whole piece to fit onto the previous page(s). On the other hand, if the final page is not being fully used, setting a magnification greater than 1.0 can cause it to be filled out. (An alternative way of achieving these effects is to change the note spacing.)

38.60 Maxbeamslope

This directive can be used to limit the maximum slope of beams. It takes two numbers as arguments. These are the maximum slopes of beams containing two notes and beams containing more than two notes, respectively. The default setting is equivalent to

```
  maxbeamslope 0.31 0.33
```

The **[beamslope]** directive, which sets an explicit slope for a given beam, is not limited by these maxima.

38.61 Maxvertjustify *

This directive controls the maximum amount of vertical space that PMW is prepared to insert between any two systems when vertically justifying a page. The default value is 60 points, which means that if the default system gap is in force, the furthest apart any two systems can be is 104 points. To ensure that the bottom stave is always at the bottom of the page under all circumstances, specify a large value for **maxvertjustify**.

38.62 Midichannel

This directive is used to specify the allocation of a MIDI voice and/or one or more PMW staves to a MIDI channel, and to set a relative volume for the channel. This information is used only when PMW writes a MIDI output file.

The values set by **midichannel** are carried over from one movement to the next, but it can appear in any movement to alter the settings if necessary. It can have up to four arguments, of which only the first, the channel number, is mandatory. There is also a **[midichannel]** stave directive which can be used to change the settings part-way through a movement.

To allocate a particular MIDI voice (also known as a ‘program’ or a ‘patch’ in MIDI-speak) to a MIDI channel, the voice number or name is given in quotes after the channel number. For example,

```
  midichannel 1 "#57"
  midichannel 2 "church organ"
```

If a channel's voice is specified more than once, the last specification overrides the earlier ones. There are sixteen MIDI channels, numbered 1–16 (but see below for the special properties of channel 10).

There are 128 possible MIDI voices; the first form of the directive, where the string starts with #, specifies the voice by a number in the range 1–128 (but note that it must still be supplied as a string in quotes). This numbering is in accordance with the *General MIDI* specification, which a number of manufacturers are following. Some MIDI instruments use the numbers 0–127 when setting voices manually; for these the manually set number of any given instrument is one less than the corresponding *General MIDI* number.

The second form of voice identification uses an index file to translate a name to a voice number. The file is installed in the PMW **share** directory and is called **MIDIvoices**, and you can edit it to change or add names. The version supplied contains voice names taken from the *General MIDI* specification. However, there seems to be some variation in some of the names, so it is permitted to have more than one name for any given voice number, and there are some duplicates in the supplied file.

In some MIDI multi-timbral instruments, the different voices are not balanced with regard to volume, so if the same values are used in the **playvolume** or **[playvolume]** directives for different voices, the resulting volumes do not correspond. To help balance voices, an volume value in the range 0–15 may be given after the voice name, preceded by a slash. For example,

```
  midichannel 1 "trumpet"/12 9
```

has the effect of reducing the volume of notes played via channel 1 by 12/15. This applies to all staves playing via that channel. The actual volume used for any MIDI note is 127 multiplied by the channel volume and the stave volume and divided by 225.

All staves are initially allocated to MIDI channel 1. This channel allocation can be changed by giving a list of staves to the **midichannel** command, with or without a voice name. For example,

```
  midichannel 2  1,3,4-7
  midichannel 4  "piano" 8-11
```

If no voice name is given, but a voice was set in a previous movement, that voice will be allocated when the current movement is played. If no voice is given in the first movement, no voice allocation setting is transmitted on the channel, which allows the voicing to be set manually on the instrument (if it has that ability). It is possible, having set a voice in one movement, to request 'no voice setting' in a subsequent movement by specifying an empty quoted string.

Before describing the final argument of the **midichannel** directive, it is necessary to discuss MIDI's handling of untuned percussion. A single 'voice' can handle a large number of different untuned percussion instruments, by using the 'pitch' of each note to determine which instrument should sound. For example, C might sound a bass drum and D a snare drum. Electronic keyboards often have a 'keyboard percussion' mode in which the keys correspond to percussion sounds in this way.

For some reason, this multiple instrument has not been defined as one of the 128 *General MIDI* instruments. Instead, the *General MIDI* specification states that MIDI channel 10 is to be used for this kind of percussion. On MIDI instruments that implement this, it is not possible to allocate any other voice to channel 10.

The final (optional) argument of the **midichannel** command is used to select an untuned percussion instrument. It must follow a list of staves (typically just one stave) and consists of a string in quotes which specifies either the MIDI pitch number, or the instrument name. For example,

```
  midichannel 10 5 "#60"
  midichannel 10 6 "triangle"
```

specify the use of pitch 60 for stave 5 and the pitch corresponding to a triangle for stave 6, both on channel 10.

As for MIDI voices, if the string starts with #, it specifies the pitch by number; otherwise the file **MIDIperc** inside the PMW **share** directory is searched to translate the name to a number. The supplied file contains the name allocation that appears in the *General MIDI* specification.

The effect of supplying this argument is to force all notes on the stave to be played at the same pitch, independent of the pitch that is given for printing purposes. A percussion stave could therefore be set up thus:

```
midichannel 10 4 "cowbell"
[stave 4 "Cowbell" hclef 1 stavelines 1]
b r b r | etc.
```

The notes are specified as Bs so that they print on the stave line, but they are played at the pitch that activates the cowbell sound (provided channel 10 is a *General Midi* percussion channel). For an alternative way of handling untuned percussion, see the **[printpitch]** directive.

38.63 Midkeyspacing

When a mid-line bar starts with a key signature, the **startlinespacing** data is used for any time signature that follows, but not for the key signature itself. Instead, **midkeyspacing** controls the position of such key signatures. It takes a single dimension as its argument; a positive value moves the signature further away from the preceding bar line.

38.64 Midtimespacing

When a mid-line bar starts with a time signature, its position can be controlled by the **midtimespacing** directive, which takes a single dimension as its parameter. A positive value moves the signature further away from the preceding bar line.

38.65 Musicfont *

This directive specifies, as a string in quotes, the name of the music font to be used by PMW; its parameter is a text string. The facility is intended for accessing new or alternative versions of the font. The default music font is PMW-Music.

38.66 Nobeamendrests

This directive, which has no parameters, can be used to cancel the effect of **beamendrests** in a previous movement.

38.67 Nocheck

This directive, which has no parameters, instructs PMW not to check that the notes in each bar agree with the time signature. It is also possible to suppress this check for individual bars (see section 47.51).

38.68 Nocheckdoublebars

This directive, which has no parameters, instructs PMW not to check that the notes in bars which begin or end with a double bar line agree with the time signature.

38.69 Nokerning *

This directive disables the use of kerning for text strings. For details of kerning, see section 34.8.

38.70 Nokeywarn

By default, when there is a key signature change at the start of a new system, PMW prints the new key signature at the end of the previous system, as is conventional in most music. The heading directive **nokeywarn** suppresses these warning key signatures for the entire piece. Individual occurrences can be suppressed by an option on the stave directive for the change of key signature.

If there is a change of both time and key signature at the start of a system, the warning at the end of the previous line is suppressed only if both are being suppressed. If suppression of only one of the time or key signatures has been requested, both are nevertheless printed in these circumstances.

38.71 Nosluroverwarnings

This directive, which has no parameters, can be used to cancel the effect of **sluroverwarnings** in a previous movement.

38.72 Nospreadunderlay

By default, PMW inserts additional space between notes if underlaid or overlaid syllables would otherwise overprint each other. This directive disables this facility for both underlaid and overlaid text.

38.73 Notespacing

PMW contains a table which defines the minimum amount of horizontal space to follow each kind of note; so much for a breve, so much for a semibreve, so much for a minim, and so on. Systems are made up using these spacings, until a bar is encountered which would make the system longer than the specified line length. The previous bars are then stretched to fill the line if horizontal justification is enabled.

The **notespacing** directive allows the table to be altered. It must be followed by eight numbers which define the space (in points) which must follow breves, semibreves, minims, crotchets, quavers, semi-quavers, demi-semiquavers and hemi-demi-semiquavers respectively. The values in the default table are those of the following example:

```
notespacing 30 30 22 16 12 10 10 10
```

Internally, note spacings are held to an accuracy of 0.001 points.

An alternative form of this directive which specifies a multiplicative factor for each value in the table is also available. This is requested by following the directive by an asterisk and a single number, or by two numbers separated by a slash. For example,

```
notespacing *1.5
notespacing *3/2
```

both request that each value in the note spacing table be multiplied by 1.5.

If more than one multiplicative **notespacing** is present, their effect is cumulative, but a multiplicative **notespacing** will be overridden if it is followed by an absolute setting.

At the start of a new movement, the absolute values that were current at the start of the previous movement, before any multiplications, are re-instated.

It is possible to make temporary changes to the note spacing table for certain bars of the music only (see section 47.56).

38.74 Notime †

This directive, which has no parameters, specifies that time signatures are not to be printed for the current movement. It does not of itself stop PMW from checking the notes in a bar and complaining if there are too many or too few. See also **startnotime**.

38.75 Notimebase

This directive requests that only the ‘numerator’ (that is, the upper number) in time signatures be printed, in the middle of the stave. For example, in 3/4 time, only the 3 would be printed. Both numbers are required to be given when specifying time signatures, however. This directive has no effect on time signatures specified as C or A. See also the **printtime** directive for another way of customizing time signatures.

38.76 Notimewarn

By default, when there is a time signature change at the start of a new system, PMW prints the new time signature at the end of the previous line, as is conventional in most music. The heading directive **notimewarn** suppresses these warning time signatures for the entire piece. Individual occurrences can be suppressed by an option on the stave directive for the change of time signature.

If there is a change of both time and key signature at the start of a system, the warning at the end of the previous line is suppressed only if both are being suppressed. If suppression of only one of the time or key signatures has been requested, both are nevertheless printed in these circumstances.

38.77 Nounderlayextenders

This directive suppresses the printing of extender lines at the ends of underlay words whose last syllable extends over more than one note. In a subsequent movement **underlayextenders** can be used to restore them.

38.78 Overlaydepth

If two or more character strings, all designated as overlay, are attached to the same note, then they are automatically printed one above the other. The distance between the baselines of the strings can be set by this directive. The default depth is 11 points. The overlay depth and the underlay depth are separate parameters.

38.79 Overlaysize

By default, text which is specified as being vocal overlay is printed using a 10-point font. This directive enables a different point size to be chosen for overlaid text. For example,

```
  overlaysize 9.7
```

Individual items of overlay text can be printed at different sizes by using the `/s` text qualifier. The overlay size and the underlay size are separate parameters.

38.80 Page *

By default, page numbers start from one. The **page** directive can be used to specify that they should start at a different number. It takes the number of the first page as its first parameter.

There is also a second, optional parameter which gives the increment by which page numbers are advanced. For example,

```
  page 3 2
```

might be used in a file containing the *primo* part of a piano duet. It causes the pages to be numbered 3, 5, 7, etc.

Occasionally there is a requirement to skip a page number in the middle of a piece – to insert a commentary page in a critical edition, for example. See the **[page]** stave directive for a means of doing this.

38.81 Pagefooting

The **pagefooting** directive defines text to be printed at the foot of pages. If a **footing** directive is present, it overrides **pagefooting** for the first page only. The **lastfooting** directive can be used to override it for the final page of a piece.

The parameters for **pagefooting** are the same as those for **footing**, but long strings are not automatically split up into multiple lines. Note the use of the escape sequences `\p\`, `\po\`, and `\pe\` to include page numbers in footing lines.

38.82 Pageheading

The **pageheading** directive defines text to be printed at the head of pages other than the first. Its parameters are the same as those for **heading**, but long strings are not automatically split up into multiple lines. Note the use of the escape sequences `\p\`, `\po\`, and `\pe\` to include page numbers in heading lines.

See chapter 24 and also the **[newmovement]** directive for discussions of headings when there is more than one movement in a file.

38.83 Pagelength *

See section 38.57 (*Linelength and pagelength*) above.

38.84 Playtempo

This directive is used to specify the tempo that is used when PMW creates a MIDI output file. The **playtempo** directive has one obligatory argument, which is the tempo to be used at the start of the movement. The tempo is always specified in crotchets per minute, whatever the time signature. The initial setting can be followed by pairs of numbers separated by slashes, which specify changes of tempo at specified bars. For example,

```
  playtempo 100 24/120 60/90
```

specifies that the initial tempo is to be 100, but at the start of bar 24 it changes to 120, and at the start of bar 60 it changes to 90. Bar numbers are given in the standard PMW style: if there are uncounted bars then decimal fractions can be used to refer to them (see chapter 23 for details).

If there is more than one movement, the initial tempo specified in a **playtempo** directive carries over to the next movement (unless it contains its own **playtempo** directive, of course), but tempo changes within a movement do not.

If no **playtempo** directive is present, the default tempo of 120 is used.

38.85 Playtranspose

By default, PMW plays music exactly as written (except for recognizing transposing clefs). If it is a score that contains a part for a transposing instrument it will not play correctly. The **playtranspose** directive is provided to help with this.

It is used to specify that particular staves are to be played at a pitch different to that at which they are printed. It is followed by pairs of numbers separated by slashes; the first number of each pair is a stave number and the second is a transposition in semitones. For example,

```
  playtranspose 1/-3
```

specifies that stave 1 is to be played a minor third lower than written.

38.86 Playvolume

The **playvolume** directive can be used to set different relative volumes for different staves. The value for a relative volume lies between 0 (silent) and 15 (maximum volume). By default, all staves are set at the maximum volume. A single number sets the volume for all staves; this can be followed by pairs of numbers separated by slashes, which specify relative volumes for individual staves. For example,

```
  playvolume 6 2/15
```

specifies that stave 2 is to be played at maximum volume, while all other staves are to be played at volume 6. See also the **[playvolume]** stave directive.

38.87 PMWversion

This directive checks that a given version of PMW is in use. It must be followed by a version number, for example,

```
  pmwversion 4.00
```

If the wrong version is used, a message is output and PMW stops.

38.88 Printtime

Time signatures are occasionally printed in unusual formats. This directive specifies how a given time signature is to be printed. There may be many occurrences in a single input file. It has the following syntax:

`printtime` <*time signature*> `"`<*top string*>`"` `"`<*bottom string*>`"`

Whenever the given time signature is to be printed, the two given strings are printed instead, one above the other, with their centres aligning. If the second string is empty, the first is printed on the second stave line; otherwise they are printed on the third and first stave lines, respectively. Some examples of possible uses are:

```
  printtime 8/8 "3+3+2" "8"
  printtime 12/8 "3 2" "2 2"
  printtime 3/4 "" ""
```

The last example suppresses all printing for the 3/4 time signature.

The default font at the start of each string is the font specified in the most recently preceding **timefont** directive, so the order of **timefont** and **printtime** matters. If **timefont** is not used, the default font is the bold font. However, changes of font are permitted within the strings.

The default size of the text printed by **printtime** is that specified by **timefont**, with 11.8 points as the ultimate default. However, it is possible to follow each text string with `/s` and a number, to specify a particular size for a given string. The number refers to the list of text sizes specified by the **textsizes** directive.

38.89 Psfooting

This directive makes it possible to include raw PostScript in PMW output at the end of the first page. This facility is for PostScript experts only. The directive must be followed by a string in double quotes. If the first character of the string is ‘<’ then the rest of the string is taken as a file name from which to copy PostScript into the PMW output at the footing point.

When the PostScript is inserted, the environment is as for the music that has just been printed, with the origin at the left-hand bottom corner of the page. Any magnification is still in force.

The string is *not* processed as a normal PMW string. This means that, if any backslashes are required in the PostScript output, they must *not* be doubled in the input string.

38.90 Psheading

This directive makes it possible to include raw PostScript in PMW output at the head of the first page, in the same format as for **psfooting**.

38.91 Pslastfooting

This directive makes it possible to include raw PostScript in PMW output at the end of the final page, in the same format as for **psfooting**.

38.92 Pspagefooting

This directive makes it possible to include raw PostScript in PMW output at the ends of pages other than the first, in the same format as for **psfooting**.

38.93 Pspageheading

This directive makes it possible to include raw PostScript in PMW output at the heads of pages other than the first, in the same format as for **psfooting**.

38.94 Pssetup *

This directive is used to include private PostScript at the end of the prologue which is output by PMW before the PostScript for the first page, in the same format as for **psfooting**.

38.95 Rehearsalmarks

The style of printing of rehearsal marks can be set by this directive, which has the syntax

`rehearsalmarks` <*style*> <*size*> <*fontname*>

All three parameters are optional, except that if <*fontname*> is present, either <*style*> or <*size*> (or both) must precede it.

The style must be one of the words ‘boxed’ (enclosed in a rectangular box), ‘ringed’ (enclosed in a ring), or ‘plain’ (not enclosed). If no word is given, there is no change of style. The size is the font size to be used, while the third parameter specifies the font to be used (see chapter 31 for details of font names). For example:

```
rehearsalmarks boxed 13
rehearsalmarks ringed italic
rehearsalmarks 11 bolditalic
```

By default, rehearsal marks are printed enclosed in a box, in a 12-point roman font.

38.96 Repeatbarfont

The font used printing the numbers on first and second time bars can be set by this directive. Its arguments are an optional size followed by a (non-optional) font name. For example,

```
repeatbarfont 8 extra 4
```

The default size is 10 points, and if this directive is not used, these numbers are printed in the roman font.

38.97 Repeatstyle

This directive specifies how repeat marks are to be printed. The default is the conventional combination of two dots with a thin and a thick vertical line. The directive must be followed by one of the following numbers:

0	normal style
1	no thick vertical line
2	no thick vertical line, and thin line dotted
3	four dots only
4	as 0, but alternate amalgamated form

Style 4 is exactly the same as style 0 (the default) except when the end of a repeated section is immediately followed by the start of another repeated section (typically coded as `:)|(:` in the input file).

In style 0, a thin line, thick line, and second thin line are printed between the dots. This style is recommended in Gardner Read's *Music Notation* and also shown in Kurt Stone's *Music Notation in the Twentieth Century*. However, some editors prefer to have two thick lines between the dots, and this is what style 4 gives.

38.98 Selectstave(s)

This directive is used to specify a selection of staves to be printed. It overrides any selection given by the **-s** option on the command line. The directive is followed by a list of staves and/or ranges of staves, and is intended for use in conjunction with the **Format** option. For example,

```
*if chorus
  selectstaves 1-4
*fi
*if cello
  selectstave 1
*fi
```

Any tests that rely on a particular stave selection must follow this directive, if it is present.

38.99 Sheetdepth, Sheetwidth, and Sheetsize *

These three directives are concerned with specifying the size of page image that PMW creates. **Sheetdepth** and **sheetwidth** can be used to specify the vertical and horizontal dimensions individually, but for standard sizes it is usually simpler to use **sheetsize**, which must be followed by one of the words ‘A3’, ‘A4’, ‘A5’, or ‘B5’. Its effect is to set the sheet depth and width parameters to suitable values for the given paper size, and also to set the **linelength** and **pagelength** values, as follows:

Size	*Sheetwidth*	*sheetdepth*	*Linelength*	*Pagelength*
A3	842	1190	730	1060
A4	595	842	480	720
A5	421	595	366	480
B5	499	709	420	590

Adjustments to the line length or page length must be made after any appearance of **sheetsize**, which should also precede any occurrence of the **landscape** directive.

If A5 or B5 is specified and the page is printed on A4 paper, it appears by default at the bottom left-hand corner. This position can be adjusted by using the **-printadjust** command line option.

38.100 Shortenstems

Some editors like to shorten note stems that are pointing the ‘wrong’ way (upward stems for notes above the middle of the stave or downward stems for notes below the middle of the stave). PMW can be made to do this shortening automatically. **Shortenstems** must be followed by one number, which is the maximum amount by which a stem may be shortened. For example,

```
shortenstems 4
```

permits PMW to shorten stems automatically by up to 4 points. The default value of zero causes no automatic shortening. Additional shortening (or lengthening) can be specified explicitly for any given note, and this is added to any automatic shortening that may be set. PMW maintains an overall minimum stem length beyond which stems cannot be shortened, so specifying a large limit such as 99 permits shortening down to this minimum length.

Automatic shortening reduces a stem’s length by 0.5 points for each note position on the stave, so, for example, a note on the top line has its upward-pointing stem shortened by 2 points (provided the **shortenstems** limit allows this).

38.101 Sluroverwarnings

By default, PMW does not draw slurs and ties that extend over the ends of lines across warning time and key signature bars. This directive requests it to do so.

38.102 Smallcapsize

When the escape sequence `\sc\` is used in a string to change to small caps, it selects a new font of the same typeface as before, but at a relative size which can be set by this directive. The default value is 0.7.

38.103 Startbracketbar †

The directive

```
startbracketbar <number>
```

causes the joining bracket and/or brace that normally appears at the left-hand end of each system to be ‘indented’ by <*number*> of bars, on the first system only. Typically <*number*> has the value 1. If the word ‘join’ appears before the number, the joining lines as specified by the **join** and **joindotted** directives are repeated at the indented position; by default they are not (but usually there is a bar line present). See chapter 30 for an example of the use of **startbracketbar**.

38.104 Startlinespacing

This directive controls the spacing of clefs, key signatures, and time signatures at the start of lines of music. It can be followed by up to four dimensions. Omitted numbers are taken as zero. The syntax is

```
startlinespacing <c> <k> <t> <n>
```

where each number specifies additional space *before* a particular item at the start of each stave:

<*c*>	is the extra space before the clef
<*k*>	is the extra space before the key signature
<*t*>	is the extra space before the time signature
<*n*>	is the extra space before the first note

The parameters can be given negative values to move the items closer together. If an item is absent on a stave, the associated extra space is also omitted.

When a mid-line bar starts with a clef (rare in the ordinary course of events, but can occur, for example, after an incipit) then the **startlinespacing** values are used for the clef and any signatures that follow it, exactly as at the start of a line.

See **midkeyspacing** and **midtimespacing** for ways of handling key and time signatures that occur at the start of mid-line bars.

38.105 Startnotime †

This directive, which has no parameters, causes no time signature to be printed at the start of a piece, but does not suppress the printing of subsequent time signature changes. This is useful for printing parts of pieces. The **notime** directive suppresses all time signatures in a piece.

38.106 Stavesize(s)

This directive specifies that certain staves are to be printed at a different size to the rest. It is followed by pairs of numbers, separated by slashes. The first of each pair is a stave number, while the second is the size of the stave relative to the standard stave size. For example,

```
  stavesize 1/0.8
```

specifies that stave 1 is to be printed at 0.8 times the normal size, while

```
  stavesizes 4/1.2 5/1.2 6/1.2
```

specifies that staves 4–6 are to be printed at 1.2 times the normal size.

A change in the relative size of a stave affects everything that prints on that stave, both notes and text items. However, the text that appears to the left of the stave (i.e. the instrument name) is not affected, and neither are bar numbers or rehearsal letters.

Bar lines are printed thicker or thinner, as necessary. It is conventional to break bar lines between staves of different sizes. If this is not done, the join between thick and thin may be visible, depending on the size difference.

A size may be specified for stave zero if required. As no notes are ever printed on this stave, this affects text items only.

38.107 Stavespacing

This directive controls the amount of vertical white space between staves. The distance between staves is measured from the bottom (or top) of one stave to the bottom (or top) of the next. The default is 44 points.

If the **stavespacing** directive is followed by just one number, this sets the spacing for all staves to that value. After such a single number, further items can be given to set different spacings for individual staves. For example,

```
  stavespacing 50 1/54 3/60
```

sets the spacing for all staves to 50 points, except for staves 1 and 3, which have their own settings.

The initial overall number is optional. The remaining parameters for this directive consist of pairs or triples of numbers, separated by a slash.

The first number is always a stave number. In the case of number pairs, the second number specifies the spacing between the stave and its successor on the page. For example

```
  stavespacing 1/36 4/50
```

would ensure that staves 1 and 2 were nearer together than the default, at 36 points, while staves 4 and 5 were further apart at 50 points (assuming that all these staves were selected for printing).

Sometimes there is a requirement to specify the amount of space *above* a stave. For example, in a piece with an accompaniment and four vocal lines, not all of which are present throughout the piece, it is a common requirement that there be more space between the last vocal stave (whichever it is) and the first accompaniment stave. Changing the stave spacing every time the last vocal line is suspended or resumed can be avoided by using a triple in the **stavespacing** directive.

Whenever three numbers appear as a parameter to **stavespacing**, the second number specifies a *minimum* space *above* the given stave, while the third specifies the space below it. For example,

```
  stavespacing 1/46 2/50 3/50/48
```

specifies that stave 3 always has at least 50 points above it, even when stave 2 is suspended.

Space specified above the top stave is ignored, and if it is desired to specify space above the last stave, some dummy third number must be given to fulfil the syntax requirement of three numbers.

The spacing between staves can be varied in the middle of a piece. See the stave directives **[sshere]** and **[ssnext]** (section 47.81).

A value of zero may be given for the spacing. This causes two successive staves to print on top of each other, and can be useful for printing two lines of music on the same stave. It can also be useful for printing a figured bass line, using invisible notes to set the horizontal positioning for the figures.

Note that if only a few bars of a piece require overprinting, the **[reset]** stave directive may be more convenient than the use of a complete overprinted stave.

38.108 Stemlengths

This directive is followed by up to six numbers, which specify adjustments to stemlengths for unbeamed minims, crotchets, quavers, semiquavers, demisemiquavers, and hemidemisemiquavers, respectively. The values may have fractions, and negative values (indicating stem shortening) can be used. Unbeamed notes that are shorter than a semiquaver have their stems automatically lengthened by 2 and 4 points, respectively, in order to fit in the extra tails.

38.109 Stemswap

This directive can be used to alter the way in which PMW chooses stem directions for notes lying on the stem swap level for the stave. Specifying this directive has the effect of altering rules N5 and N6 as described in chapter 43 (*Stem directions*). Note that rule N3 is *not* affected.

```
  stemswap up
```

All notes at the stem swap level that are not otherwise constrained have stems that go upwards. This can be useful when there is vocal underlay.

```
  stemswap down
```

This gives the opposite effect and may be useful for American publishers.

```
  stemswap left
```

This makes these notes depend on the stem direction of the note to their left, viewing the part as one long stave (i.e. on the previous note).

```
  stemswap right
```

This makes them depend on the next note to their right that is not also on the stem swap level. However, this does not extend beyond the end of a bar. If the final note(s) of a bar are on the stem swap level, their stem direction is taken from the preceding note.

38.110 Stemswaplevel

This directive specifies, for each stave, the level at which stems normally swap from pointing down to pointing up. The default value is zero, which specifies the middle line of the stave. On the swap level itself, the stem may go either up or down, depending on the surrounding notes and the option set by **stemswap**.

Stemswaplevel can be followed by a single number, in which case it refers to every stave, or it can be followed by pairs of numbers separated by slashes, rather like **stavespacing**. The change in swap level may be positive or negative, and its units are note positions. Thus, for example,

```
  stemswaplevel 1/1 2/-1
```

requests that on stave 1, the swap level is moved to the third space instead of the third line, while on the second stave it is moved down to the second space.

38.111 Suspend †

The **suspend** heading directive specifies the suspension of certain staves from the beginning of a piece. It must be followed by a list of staves, in the same format as the **bracket** and **brace** directives. For example:

```
  suspend 1,3,5-9
```

A detailed description of the suspension mechanism is given in the section on the stave directive of the same name (section 47.86).

38.112 Systemgap

The vertical distance between systems of staves is measured from the bottom (or top) of the last stave of one system to the bottom (or top) of the first stave of the next system. The default distance is 44 points, the same as the spacing between staves in a system. Thus by default, the entire output on a page is on evenly spaced staves (if there is no vertical justification). The **systemgap** directive enables the user to specify a different amount of space between systems. It takes a single dimension as its parameter.

The spacing between systems can be varied in the middle of a piece. See the stave directives **[sghere]** and **[sgnext]** in section 47.73.

38.113 Textfont *

By default, all text characters that form part of a page of music are printed using the *Times* series of fonts. This directive can be used to specify alternative fonts and also to define up to twelve additional fonts. It takes the following form:

  `textfont` <*fontword*> `"`<*full font name*>`"`

The first parameter must be one of the words ‘roman’, ‘italic’, ‘bold’, ‘bolditalic’, ‘symbol’ or ‘extra’ followed by a number in the range 1–12, specifying which text font is being defined. The second parameter is the full name of the font, in double quotes. For example,

```
  textfont bold "Palatino-Bold"
```

This changes the bold face font from the default (which is Times-Bold) to Palatino-Bold. An example which defines the first of the twelve available extra fonts is

```
  textfont extra 1 "Helvetica"
```

This font is accessed in text strings by the escape sequence `\x1\`. See section 34.6 for details of font-changing escape sequences. The capitalization of font names is important.

38.114 Textsize(s)

Text which is specified with the music on a stave can be printed in twelve different sizes in addition to the default sizes for underlay, overlay, and figured bass text. The **textsizes** directive specifies the sizes that are required. It is followed by up to twelve font sizes, which may include stretching factors and shear angles. Any unspecified sizes are set to 10 points. For example,

```
  textsizes 10.5 11 7.6 9/1.1
```

By default, ordinary text is printed using the first size specified, while underlay, overlay, and figured bass text is printed using the size specified by the **underlaysize**, **overlaysize**, or **fbsize** heading directives, respectively. To print text at any of the other sizes, the `/s` qualifier must be used – details are given in chapter 44.

38.115 Thinbracket

This directive, which has the same syntax as **bracket** and **brace**, causes a thin, square bracket to be drawn to join two or more staves. Like **brace**, nothing is drawn if it covers only one stave, and it is drawn outside the thicker bracket, if that is present. This sign is sometimes used in scores to join staves containing multiple parts for the same instrument.

38.116 Time †

The **time** directive defines a common time signature for all the staves of the piece or movement, in a similar way to the key signature. Again, changes can be made during the music or for individual staves, which are permitted to have different time signatures. See the **[time]** directive for details. The default time signature is 4/4.

38.117 Timebase

This directive can be used at the start of a new movement to cancel the effect of **notimebase** in the previous movement.

38.118 Timefont

The **timefont** directive can be used to specify the default font and size for the printing of time signatures. Its syntax is:

timefont <*size*> <*name*>

The size is a number, giving the size of font required. If it is omitted, a font size of 10 points is used. The name must be one of the words 'roman', 'italic', 'bold', or 'bolditalic', or the word 'extra' followed by a number in the range 1–12. It cannot be omitted. When this directive is not used, an 11.8-point bold font is used for printing time signatures.

The parameters set by **timefont** do not affect the printing of the time signatures C and A – they affect only numeric time signatures, or those printed via the **printtime** directive.

Changing the size of the time signature font does not affect the positioning of the characters. The facility is intended for selecting a suitable size when a font other than Times-Bold is used.

As an example of the use of **timefont**, consider the printing of an original time signature in the form of a circle, for a piece which has three minims to the bar. If this is the only time signature that is to be printed, it can be specified as follows:

```
timefont 10 bold
printtime 3/2 "\**147\" ""
time 3/2
```

A 10-point font is required, to match the music font with which the music itself is printed. The word 'bold' is required by the syntax of the **timefont** directive, even though the bold font is not itself actually used. Character 147 in the music font (requested by the asterisks) is a circle of the right size.

38.119 Timewarn

This directive can be used at the start of a new movement to cancel the effect of **notimewarn** in the previous movement.

38.120 Topmargin

See section 38.13 (*Bottommargin and topmargin*) above.

38.121 Transpose †

A **transpose** directive given in the heading sets a transposition for the whole movement. It does not affect subsequent movements. It must be followed by a positive or negative number which specifies the number of semitones of transposition up or down, respectively. If a transposition is also specified from the command line, the two values are added together.

Chapter 29 gives more details about transposition.

38.122 Transposedacc

By default, PMW always prints an accidental on a transposed note if an accidental was present on the original, thereby preserving cautionary accidentals. The directive

```
transposedacc noforce
```

changes this behaviour such that accidentals are printed only when strictly necessary. The standard behaviour can be reinstated for subsequent movements by specifying ‘force’. It is possible to force either behaviour for individual notes.

38.123 Transposedkey

When there is a choice of key signature after transposition, PMW uses a fixed default. For example, it uses the key of G♭ rather than F♯. There is a complete list of the key signatures in chapter 29. The default can be overridden by specifying

```
transposedkey <key1> use <key2>
```

which means ‘if transposing a key signature yields <*key1*>, then use <*key2*> instead’. There may be many occurrences of this directive. Thus, to ensure transposition into F♯, use

```
transposedkey G$ use F#
```

The **transposedkey** directive has other uses when transposing music which is notated using the 18th century convention of fewer accidentals in the key signature than in the tonality. It makes it possible to print the transposed music either with a modern key signature, or using the same convention.

A transposition of zero is different to no transposition at all, and if it is specified, any settings of **transposedkey** are consulted. This makes it easy to print the same piece of music with or without a key signature.

The **transposedkey** directive is relevant when chord names in strings are being transposed, because the name is transposed in exactly the same was as the note would be.

38.124 Trillstring

When a trill is indicated for a note, the glyph *tr* is printed from the music font. The **trillstring** directive enables this to be changed for another character or characters. For example,

```
trillstring "\it\tr"
```

replaces it by the letters *tr*, printed in italic. The string may be preceded by a number, specifying the size of font to be used.

38.125 Tripletfont

This directive specifies the size and style of the text font used to print the ‘3’ over triplets, and also similar numbers over other irregular note groups. The syntax is

```
tripletfont <fontsize> <name>
```

The size is a number giving the font size (with an optional stretching factor and shearing angle). If it is omitted, a size of 10 points is used. The name must be one of the standard font name words, e.g. ‘bolditalic’ (see chapter 31). It cannot be omitted. When this directive is not used, a 10-point roman font is used for printing triplet numbers.

38.126 Tripletlinewidth

This directive is be used to set the width of lines used for the brackets of irregular note groups. The default width is 0.3 points.

38.127 Underlaydepth

If two or more character strings, all designated as underlay, are attached to the same note, then they are automatically printed one below the other. The distance between the baselines of the strings can be set by this directive. The default depth is 11 points.

A negative parameter can be given to this directive for special effects, such as printing alternative words above a stave. However, this is probably easier to achieve by using the overlay facilities.

The depth parameters for underlaid and overlaid text are separate and independent.

38.128 Underlayextenders

In a second or subsequent movement, this directive restores the printing of extender lines at the ends of underlay words whose last syllable extends over more than one note if it was suppressed by **nounderlayextenders** in an earlier movement.

38.129 Underlaysize

By default, text which is specified as being vocal underlay is printed using a 10-point font. This directive enables a different size to be chosen for underlaid text. For example,

```
underlaysize 9.5
```

Individual items of underlay text can be printed at different sizes by using the `/s` text qualifier. The size parameters for underlaid and overlaid text are separate and independent.

38.130 Underlaystyle

By default, PMW centres underlay and overlay syllables under each note. There is a tradition, ‘now frequently ignored’ (Kurt Stone, *Music Notation in the Twentieth Century*), which calls for multinote syllables to be aligned flush left with the initial note. The **underlaystyle** directive is used to request PMW to align underlay and overlay in this traditional manner. Its argument is a number: style 0 is the default, while style 1 sets multinote syllables flush left.

When operating in style 1, individual multinote syllables can be centred by making use of the ^ character, which is still recognized in this style. In effect, style 1 causes the automatic insertion of a ^ character at the start of any multinote syllable that does not already contain one.

38.131 Unfinished †

The directive **unfinished**, which has no parameters, indicates that the musical data supplied is not a complete piece. This has the effect of suppressing the solid bar line at the end. It applies only to the movement in which it appears. It is not necessary to specify **unfinished** if the piece ends with a double bar line.

38.132 Vertaccsize

The size of accidentals that are printed above or below notes (see section 41.5) is controlled by this heading directive; the default size is 10 points, which causes them to be the same size as normal accidentals. For example,

```
vertaccsize 9
```

causes them to be printed slightly smaller.

39. Stave data

This chapter is the first of a set of chapters in which we describe the format of the musical data for a single stave, which consists of a sequence of notes and rests, interspersed with other items such as bar lines, key and time signatures, clefs, text strings, etc. The items that are not notes or rests are of the following forms:

- A few common items which can conveniently be represented in the computer's character set are represented by one or more special characters. An example is the use of the vertical bar to indicate a bar line. These items are described in the following chapter, *Non-note characters in stave data*.
- Textual items, such as '***f***', 'a tempo' etc., are coded as text strings enclosed in double-quote characters, and are described in chapter 44.
- Other non-note items take the form of *stave directives*, enclosed in square brackets. There are several different formats, and these directives are described in alphabetical order in chapter 47.

Notes, rests and other items may be interspersed freely, as required. PMW makes no attempt to check on the musical sense of what it is asked to print, other than to check the length of each bar (except when this check is disabled).

Space characters and line breaks can be used to separate items to make the input easier to read, though they are not necessary. Alternatively, semicolons can be used instead of, or as well as, spaces. However, if a semicolon follows a note which could be beamed, it causes a beam break to occur.

40. Non-note characters in stave data

A number of characters or character sequences may occur interspersed among the notes and directives of a stave, and these are described in this chapter.

40.1 Bar lines

Bar lines in the music are indicated by means of the vertical bar character. A single vertical bar gives a single bar line; two successive vertical bars gives a double bar line. To encode a totally empty bar it is therefore necessary to include a space between the two vertical bar characters.

Barlines may be printed in 6 different styles (see section 38.7). The default style can be set by **barlinestyle** (for the whole piece) or **[barlinestyle]** (for an individual stave). In addition, the style of any individual bar line may be specified by following the vertical bar character with a digit in the range 0–5. Note also that the **breakbarlines** directive can be used to specify breaks in bar lines at particular staves.

PMW checks the notes and rests in a bar to ensure that they agree with the time signature, though this check can be suppressed (see the **nocheck** and **[nocheck]** directives).

Occasionally it may be necessary to put in a dummy bar line in order to allow PMW to start a new system in the middle of a bar – something it does not normally do. If a vertical bar character in the input is immediately followed by a question mark, it behaves exactly as a normal bar line, except that nothing is printed.

The bars on either side of such an invisible bar line are typically of abnormal length, so the bar length check will need to be turned off for each of them (using **[nocheck]**), and if bar numbers are being printed, the **[nocount]** stave directive should be used to stop one of them from being counted.

Normally, the end of a bar marks the end of a set of beamed notes. It is, however, possible to carry a beam over the bar line and into the next bar. This is done by following the vertical bar character by an equals sign in the input. A full description of this facility is given in section 42.2.

The amount of horizontal space which is inserted after a bar line is controlled by the **barlinespace** directive. This applies to both visible and invisible bar lines.

40.2 Repeated sections

The beginnings and ends of repeated sections are marked by the character sequences

`(:`	for the start of a repeated section
`:)`	for the end of a repeated section

These need not be at the beginning or end of a bar, though if they are, the repetition sign is amalgamated with the bar line in the conventional manner. When the end of one repeated section is ‘back to back’ with the start of another repeated section, two different forms of amalgamated marking are available, controlled by the **repeatstyle** directive (section 38.97).

PMW does not split lines of music other than at the ends of bars (but see the previous section concerning ‘invisible bar lines’). It makes no check that the repetition signs make musical sense.

When a bar starts with a new time signature and a repeat mark, the order in which these are printed depends on the order in which they appear in the input.

```
[time 4/4] (:
```

causes the time signature to be printed first, followed by the repeat mark, while

```
(: [time 4/4]
```

causes the repeat mark to be amalgamated with the previous bar line, with the time signature following. If, at the same point in the music, these items appear in different orders on different staves, then the repeat sign is printed first on all staves.

40.3 Hairpins

The characters `>` and `<` are used within a stave to encode hairpins. They are always used in pairs, and they enclose the set of notes above or below which the hairpin is to be drawn. Thus

```
  a b > c d e >
```

specifies a diminuendo hairpin that extends under the three notes C, D, and E. The hairpin starts at the left-hand edge of the first note, and ends at the right-hand edge of the last note. Unterminated hairpins are automatically terminated at the start of a hairpin of the opposite kind.

If the end of a hairpin is given at the start of a bar, before the first note, then the hairpin is terminated just past the bar line, unless it is the first bar of a line, when it is extended far enough to be of reasonable size. (See also the `/bar` option below.)

A minimum length of 10 points is imposed on hairpins. If a hairpin would be shorter than 10 points, it is extended on the right until it is 10 points long. As well as the case of a hairpin terminating at the start of a system, this can also happen if a hairpin is specified with only a single note between the angle brackets.

Hairpins can extend over bar boundaries; if a hairpin extends over the end of a system, it is terminated, and a fresh one started on the next system. However, the end of the first part of a decrescendo or the start of the continuation of a crescendo is drawn with a small gap to indicate the continuation.

The positioning of hairpins takes account of the notes under or above which they lie. However, manual adjustment is also provided. The hairpin characters can be qualified using `/u`, `/d`, `/l`, and `/r` and a number to move up, down, left and right. If `/u` or `/d` is given at the start of a hairpin, it causes the whole hairpin to be moved up or down. Thus

```
  a b </d4 c d <
```

prints a hairpin that is 4 points lower than the default position. The `/l` and `/r` qualifiers, on the other hand, affect only the end of the hairpin at which they are specified. Thus

```
  </l5 a b c d </r5
```

stretches the hairpin by 5 points at each end. If the `/u` or `/d` qualifiers are used on the terminating < or > character of a hairpin, they cause that end of the hairpin to be moved up or down relative to the left-hand end. For example,

```
  < abc </u10
```

specifies a crescendo hairpin which slopes upwards to the right. Adding `/u` or `/d` to the left-hand end would move the whole hairpin up or down, without affecting the angle of slope.

If a hairpin is split, specifying `/u` or `/d` at its start moves both halves of the hairpin up or down, but specifying one of them on the final angle bracket moves just the final end point, as for non-split hairpins.

The vertical positions of the intermediate ends of split hairpins can be controlled by the options `/slu`, `/sld`, `/sru`, and `/srd`. They must be given on the starting angle bracket of a hairpin.

The rather confusing abbreviations ‘sl’ and ‘sr’ stand for ‘split left’ and ‘split right’. They refer to the two ends of the split as they would be on one long system before it is split up. Thus the ‘sl’ refers to the right-hand end of the first part of a split hairpin, while ‘sr’ refers to the left-hand end of the second part. To move the second part of a hairpin down by 10 points say, you would use `/srd10` on the starting angle bracket, and `/d10` on the final bracket.

The horizontal position of the start or the end of a hairpin can be set to be halfway between the relevant note and the one that follows it, or the end of the bar, by means of the `/h` option. For example,

```
  >/h GAB >/h B
```

starts the hairpin halfway between G and A, and ends it halfway between the two Bs. Without `/h`, it would have started at the G and ended at the first B. The use of `/h` always moves the end point to the right.

In fact, the `/h` option is more general; it can be followed by a number indicating the fraction of the distance that is required. For example,

```
  < GGG </h0.8 |
```

ends the hairpin 0.8 of the way between the last note and the bar line. If `/l` or `/r` are given as well as `/h`, the effect is cumulative.

Another option that adjusts the horizontal position of hairpins is `/bar`. If the character indicating the start of a hairpin is followed by `/bar` then the hairpin starts at the horizontal position of the previous bar line, except at the start of a system, where it starts after the clef and key signature. If the character indicating the end of a hairpin is followed by `/bar` then the hairpin ends at the *next* bar line. The use of `/bar` overrides `/h`.

Hairpins are printed under the stave by default, but the **[hairpins]** directive (section 47.35) can be used to make them print above instead. It can also specify a fixed distance above or below the stave, or a general vertical adjustment for all hairpins.

Individual hairpins can be forced to be above the stave, below the stave, or in the middle between the current stave and the one below, by means of the options `/a`, `/b`, and `/m`. Do not confuse `/m` with `/h`. One way of remembering the difference is to associate `/h` with ‘horizontal’ rather than ‘halfway’.

Individual hairpins can be printed at a fixed level above or below the stave by following `/a` or `/b` by a dimension, which specifies the position of the point of the hairpin. For example

```
  >/a10 bc'eb > |  </b12 gfag < |
```

prints the hairpins with their points 10 points above the top of the stave and twelve points below the bottom of the stave, respectively.

The width of the open end of an individual hairpin can be set by following the initial `<` or `>` character with `/w` and a dimension. For example,

```
  </w4 ga <
```

The default width is 7 points, but this can be changed by the **hairpinwidth** heading directive (for the whole piece) or by the **[hairpinwidth]** directive (for the current stave). There is also **hairpinlinewidth** directive, which is used to change the thickness of the lines in hairpins. The default thickness is 0.2 points.

40.4 Caesurae

A caesura in the music is shown in the input in very much the way it is printed, by two successive slashes:

```
  c'B // r-c'- |
```

It is normally printed as two strokes, but the **caesurastyle** directive can be used to obtain a single-stroke form.

40.5 Dotted bar lines

The character `:` (colon) may appear on its own in the middle of a bar. It causes a dotted bar line to be printed at that point. The bar line is subject to the normal controls for whether it extends down to the next stave or not. A colon is not counted as the end of a bar.

41. Notes and rests

The information for a note consists of five parts, of which only the first two are mandatory. These are, in order: pitch, length, expression and/or options, tie or slur information, and beam break information. A rest has only a length and an options part.

Notes and rests do not have to be separated by spaces in the input, though spaces can be inserted for readability if required. A sequence such as

```
abcd
```

is perfectly valid input for four notes. Spaces may not, however, appear within the encoding for one note, except in the options part as specified below.

41.1 Chords

PMW can deal with certain kinds of chord, notated by enclosing a number of notes in round brackets. The notes must either all be of the same musical length, or all but the first must consist of just a lower case letter, in which case the length is taken from the first note. For example,

```
(gb)  (c′-#g′-)  (A++A′++)  (g=-bd′g′)
```

The notes do not have to be in any particular pitch order. If there are to be accents on the chord (staccato, etc.) then these must be specified on the first note. If the chord is to be tied, then the underline character must appear *after* the closing bracket. Note that an underline character cannot be used for short slurs when chords are involved. The **[slur]** directive must be used instead.

Chords consisting of quavers or shorter notes are beamed in the usual way; a semicolon after the closing bracket breaks the beaming, while a comma breaks secondary beams only.

PMW is capable of automatically positioning accidentals on chords. It does this provided that none of the notes in the chord contains an explicit accidental positioning request (see below). If any of the notes contains an explicit accidental position setting, then no automatic positioning is done, and it is assumed that the user has positioned all the accidentals in the chord by hand.

41.2 Note pitch

The pitch of a note is indicated by one of the usual note-letters, A to G. As is conventional in music, the letters represent the notes C to B, starting at the octave below middle C. The case of the letter (upper case or lower case, that is, capital or small letter) does *not*, however, form part of the pitch information (contrary to musical convention). Instead it is used to indicate the note *length*, as described below.

A note’s pitch is raised one octave by following the letter by a single quote character (apostrophe); two octaves require two quotes, and so on. Similarly, a note’s pitch is lowered one octave by following the letter by a grave accent character.

Accidentals are indicated by special characters before the note letter. The sharp character is the obvious one to use for indicating a sharp sign, but there are no obvious candidates for flats or naturals. Therefore two keys that are adjacent on most keyboards, and next to the sharp sign on some, are used: the dollar sign for flat and the percent sign for natural. Double sharps and double flats are indicated by two sharp signs or two dollars.

Here are some examples of notes of different pitches:

`c′`	middle C
`C′′`	the C above middle C
`#g`	G sharp below middle C
`$b′`	B flat above middle C
`%c`	C natural below middle C
``##g` ``	G double sharp, below the C below middle C

It is possible, when specifying the clef, or by using the **[octave]** directive, to request that all subsequent notes be transposed up or down by a given number of octaves. This is normally used with the treble clef and some of the C clefs. When, for example, one octave of transposition has been requested, the note letter C on its own represents middle C.

When two or more parts are being overprinted on the same printed stave, certain accidentals on one part are often omitted, because an accidental in another part serves, in the printed music, for both. However, if a MIDI file is being generated, the music does not sound correct when played.

Invisible accidentals are provided to change the note that is played, without causing anything to be printed. Following an accidental character with a question mark (e.g. `#?g`) causes it to become invisible. As for normal accidentals, the effect of invisible accidentals lasts until the end of the bar. Invisible accidentals may not be specified as bracketed.

41.3 Bracketed and moved accidentals

Accidentals are sometimes printed in round or square brackets. This is requested by following the accidental with a closing bracket of the appropriate type, for example

```
#)a   $]b   ##)c
```

41.4 Moved accidentals

Occasionally it is necessary to instruct PMW to move an accidental sign to the left of where it would normally print. If the character `<` follows the accidental sign, the accidental is printed 5 points to the left of its normal position, scaled to the stave size. Two successive `<` characters move it 10 points left, and so on.

Alternatively, a number may follow the `<` character to specify exactly how far left to move the accidental. Thus, for example:

`#<A`	the sharp is moved left by 5 points
`$<<b`	the flat is moved left by 10 points
`%<4.25C`	the natural is moved left by 4.25 points

41.5 Accidentals above and below notes

Some editors like to print editorial accidentals above or below notes. Text strings can be used for this, but they do not transpose, and the music is not played correctly if a MIDI file is generated.

The letters `o` and `u` are used to request that an accidental be printed over or under the note (since `a` and `b` cannot be used because they are note names). Thus:

```
#oa $ub
```

prints a sharp above the note A, and a flat under the note B. These accidentals affect the playing pitch of the note for MIDI output, but do *not* affect subsequent notes in the bar. They change with transposition, just as ordinary accidentals do.

The size of the accidentals is controlled by the heading directive **vertaccsize**; the default size is 10 points, which causes them to be the same size as normal accidentals.

It is possible to move these accidentals by following the letter with `/u` or `/d` and a number. (In fact, `/l` and `/r` are also available, though unlikely to be useful.) For example:

```
#o/u4c'  %u/d2f
```

If bracketed accidentals are required, the bracket must follow `o` or `u` and any up/down movement specification.

41.6 Transposed accidentals

PMW can be forced to print a given transposed note in a particular way (e.g. with a double sharp instead of a natural). This facility is provided for cases when the normal transposition rules are inappropriate, and it is done by following the accidental for the note (if any) with one of the following character sequences:

`^#`	print with a sharp (‘black’ notes and C and F)
`^$`	print with a flat (‘black’ notes and B and E)
`^##`	print with a double sharp (‘white’ notes except C and F)
`^$$`	print with a double flat (‘white’ notes except B and E)
`^%`	print with a natural (all ‘white’ notes)

For example, if a note which is specified as `#^##G` is transposed up by one semitone, and would normally be printed as A-natural, it will in fact be printed as G-double-sharp.

In the absence of any special indication, a subsequent note of the same pitch in the bar will automatically print in the same way.

Normally, PMW always prints an accidental sign for a transposed note if there is an accidental in the input, thus preserving cautionary accidentals. Occasionally this is not required. Suppression of an unnecessary accidental can be requested by following the accidental by the sequence

```
  ^-
```

If an accidental is actually necessary in the transposed music, it will not be suppressed.

The suppression of unnecessary transposed accidentals can be enabled for all notes by means of the **transposedacc** directive. When this is done, individual accidentals can be put back by means of the sequence

```
  ^+
```

after the input accidental.

If a bracketed accidental is required, the bracket must follow the transposition option, which in turn must follow any request to print the accidental above or below the note.

41.7 Rests

There are three ‘note letters’ which are used instead of pitch letters to specify rests. The letter R is used to indicate a normal rest. It may not be preceded by accidentals or followed by quote characters or grave accents. The letter Q is similar, but it causes nothing at all to be printed. It is often called an ‘invisible rest’. It is useful for special effects when overprinting staves or using coupled staves.

The letter S has exactly the same effect as R except when it is used to specify a complete bar’s rest. Such bars are normally candidates for amalgamation with surrounding bars, leading to the printing of ‘long rest’ bars where possible. When a rest bar is specified using S it is always printed as an individual bar and never amalgamated (see section 41.12 for more details). You can think of S as standing for ‘single’.

41.8 Length of notes and rests

The primary length of a note or rest (visible or invisible) is indicated by the case of its letter. An upper case (capital) letter is used for a minim, and a lower case (small) letter for a crotchet.

Notes or rests shorter than a crotchet have ‘flags’: a minus sign is a single flag for a quaver, an equals sign is two flags for a semiquaver, an equals followed by a minus sign is three flags for a demi-semiquaver, and two equals signs are four flags for a hemi-demi-semiquaver. If the note letter is followed by quotes or grave accents as part of its pitch, the flags follow these.

Notes or rests longer than a minim are constructed by the addition of plus signs, each of which doubles the length. One plus makes a semibreve, two make a breve.

One or two dots may follow a note or rest as in conventional music, to extend its length by half and three-quarters, respectively. There is also support for Emmanuel Ghent’s notation for extending the length of a note by one quarter (as reported in Gardner Read’s book *Music Notation*). The PMW encoding for this is to follow the note with a dot and then a plus sign. The length of the note is extended by one quarter, and it is printed as the normal note followed by a plus sign.

This facility is particularly useful when there are five beats in a bar. For example, the notatioin

```
  [time 5/4] A+.+
```

prints a semibreve followed by a plus, indicating a note whose length is equal to five crotchets. Here are some examples of notes and rests of different lengths:

`A++`	breve
``#B`+``	semibreve
`G+.+`	semibreve followed by plus
`F.`	dotted minim
`R`	minim rest
`e..`	double dotted crotchet

`$$g`	crotchet
`r-.`	dotted quaver
`c'-`	quaver
`d=`	semiquaver
`e''=-`	demi-semiquaver
`%b`==`	hemi-demi-semiquaver

41.9 Horizontal movement of augmentation dots

It is occasionally necessary to move augmentation dots to the right, usually when printing multiple parts on the same stave with notes close together. If an augmentation dot is preceded by the character `>` it is moved right by 5 points (scaled to the stave size). A different distance can be specified by preceding the `>` with a dimension. For example:

```
a>.   g6.2>..
```

In a chord, the `>` character must be used on the first note, and not on any others. It affects all the dots in the chord, because they are always vertically aligned.

41.10 Vertical position of augmentation dots

The vertical position of dots for notes on lines can be controlled by the **[dots]** directive and the `\:\` option. (See section 41.13 below for full details of note options.)

This option affects only notes on lines which have augmentation dots. Normally such dots are printed in the stave space above, but if the colon is present, they are printed instead in the space below. The default position can be changed by means of the **[dots]** stave directive; when the default is below, the colon item causes the dot for a particular note to be printed above. For example,

```
[treble]     e.\:\     @ dot below
[dots below] g..\:\    @ dot above
```

The colon can be used in individual notes within a chord. However, PMW overrides the dot position setting when an interval of a second occurs in a chord. In this case, the lower note always has its dot below, if it is on a line, and the upper note always has its dot above, if it is on a line.

The `\:\` option does not affect notes in spaces. However, it is sometimes useful to be able to move an augmentation dot into the space above. The option `\::\` achieves this; it has no effect if used on a note that prints on a line.

For example, the chord `(e.g.a.)` in the treble clef prints by default with only two dots. If three dots are required, there are two ways in which this can be achieved:

```
(e.\:\g.a.)  (e.g.a.\::\)
```

The first one moves the dot on the lowest note down, while the second moves the dot on the highest note up.

When there is an interval of a second in a chord and the higher note has its dot moved up by this means, the lower note's dot is no longer automatically moved down.

41.11 Whole bar rests

There is one other special character that may follow the letters R, Q, or S, but not any of the note letters. This is the exclamation mark, and it is used to indicate that the rest fills an entire bar. Without this, it is not possible to specify a complete bar's rest as one item in all time signatures.

The difference between `R!` and `Q!` is that the former causes the printing of a conventional whole bar rest sign, while the latter causes nothing at all to be printed in the bar. This is useful when staves are being overprinted. `S!` behaves like `R!` except that the bar in which it appears is never eligible for amalgamation into a single multiple rest bar with the bars on either side of it. A bar containing `S!` is always shown on its own.

Whole bar rests specified using an exclamation mark are normally printed as semibreve rests, centred horizontally in the bar. The form of the whole bar rest sign can be altered for certain time signatures by means of the **breverests** heading directive.

If a bar contains only a whole bar rest or a single note on every stave, it sometimes looks better if the notes are also centred. This can be done by using the `\C\` option for the notes (see section 41.13 below).

Rests which happen to fill the bar, but which are not specified with exclamation marks, are printed as rests of the appropriate length. For example, in 3/4 time the rest `R.` is printed as a dotted minim rest. If bar lengths are being checked, such a rest is printed centred in the bar, but if they are not, it is printed at the left-hand end.

The amalgamation of a sequence of whole bar rests into a single multi-bar rest still occurs, whether exclamation marks are used for the rests or not.

41.12 Repeated rest bars

PMW automatically detects when a rest completely fills a bar. It also checks for a succession of rest bars. If a sequence of two or more rest bars is found, a conventional 'long rest' sign is drawn, with the number of bars rest printed above it.

This is only done, of course, if *all* the staves in the current printing have rest bars, which normally happens when one or more parts are being extracted from a score.

A bar is considered eligible for amalgamation with its neighbour(s) in this way if it contains nothing but an unadorned rest item. A rest bar with a fermata on the rest (for example) always prints as a separate bar.

However, the initial bar of an amalgamated sequence is permitted to contain items such as key and time signatures and a beginning repeat mark, and the last bar in a sequence may end with a terminating repeat sign. A text item is also permitted in the first bar of a sequence.

There are occasions where it is required that a rest bar should not be considered eligible for amalgamation. An example is a rest bar with a text item such as 'G.P.' which occurs at the start of a sequence of rest bars. In such cases, the letter `S` should be used to notate the rest. An example of its use might be:

```
  [10]R! | "G.P."S! | [8]R! |
```

If `R` were used instead of `S` here, the last nine bars would be printed as a single multi-bar rest.

41.13 Note expression and options

The expression/options portion of a note includes all additional marks such as staccato, emphasis, trills, mordents and fermatas. It can also indicate that the note is a grace note, force the stem of the note to point up or down, indicate the lengthening or shortening of the note's stem, change the position of accents and augmentation dots, etc.

For many notes there are no such special markings and this part will not be present. If it is present, it consists of two backslash characters, between which there are one or more letters or other characters indicating the expression or option required. For example, a dot and a minus sign signify a staccato dot or a solid line emphasis, respectively.

The possible character sequences that can occur are as follows:

`\/\`	single tremolo mark
`\//\`	double tremolo mark
`\///\`	three tremolo marks
`\~\`	'upper' mordent sign
`\~|\`	'lower'mordent sign
`\~~\`	double 'upper' mordent sign
`\~~|\`	double 'lower' mordent sign
`\!\`	print accent on stem side, trill or fermata below
`\.\`	staccato dot
`\:\`	invert augmentation dot position (notes on lines)
`\::\`	move augmentation dot up (notes in spaces)
`\-\`	solid line emphasis mark
`\>\`	horizontal wedge emphasis mark
`\'\`	'start of bar' accent
`\a<`*n*`>\`	accent number <*n*> (see below)

`\ar\`	arpeggio mark
`\ard\`	arpeggio mark with downward arrow
`\aru\`	arpeggio mark with upward arrow
`\c\`	print on coupled stave
`\C\`	centre if only note in bar
`\d\`	string down bow (organ heel) mark
`\f\`	fermata (pause) above note
`\f!\`	fermata (pause) below note
`\g\`	grace note
`\g/\`	grace note with slanted line
`\h\`	don't print on coupled stave
`\m\`	masquerade note (see below)
`\o\`	small circle over note (harmonic)
`\sd\`	force note stem down
`\su\`	force note stem up
`\sw\`	swap note stem direction in beam
`\sl`<*n*>`\`	lengthen stem
`\sl-`<*n*>`\`	shorten stem
`\sp\`	spread chord
`\t\`	turn
`\t|\`	inverted turn
`\tr\`	trill
`\tr#\`	trill, with a sharp sign above
`\tr$\`	trill, with a flat sign above
`\tr%\`	trill, with a natural above
`\u\`	string up bow (organ toe) mark
`\v\`	small, closed vertical wedge accent
`\V\`	large, open vertical wedge accent
`\x\`	cancel default expression

More than one of these character sequences can be present between the backslashes, and spaces can be used to separate them, for example,

`#g\.-\`	staccato and tenuto
`\tr sd\`	trill and stem down

Notes that are marked as grace notes can be of any length – crotchets, quavers, semiquavers, etc. PMW beams grace notes where possible. The stems of grace notes always point upwards, whatever the pitch, unless an explicit downward stem is requested by specifying `\g sd\`. If there is more than one grace note in sequence, specifying a downward stem for the first one causes all of them to have downward stems.

The sequence `\x\` is used to cancel any default expression that may be in force (see section 47.1).

The sequences `\c\` and `\h\` are used to override the default note placing when coupled staves are in use (see **[couple]**).

When there is a whole bar rest in some staves, and just a single note in the remaining staves, it sometimes looks odd that the rest is centred horizontally in the bar and the note is not, especially if the note is a semibreve. The option `\C\`, if used on the first and only note in a bar, causes it to be centred like a whole bar rest, provided that the note has a length equal to the current bar length. (Don't confuse `\C\` with `\c\`.)

The single and double colon options are concerned with the vertical placement of augmentation dots; details are given in section 41.10.

The item `\a`<*n*>`\` is a general notation for specifying accents. The values that <*n*> may take are:

1 staccato dot ·
2 horizontal bar −
3 horizontal wedge >
4 small, closed vertical wedge ▾
5 large, open vertical wedge ∧
6 string down bow ⊓
7 string up bow ∨
8 ring (harmonic) ∘
9 ‘start of bar’ accent ˈ

By default, accents and the harmonic ring are printed on the opposite side of the notehead to the stem, but fermatae, trill signs, and other ornaments are printed above the note, independent of the stem direction. String bowing marks are always printed above the stave (but see the **[bowing]** directive).

The `!` note option causes PMW to print an accent or harmonic ring on the same side of the notehead as the stem, which is occasionally necessary when more than one part is being printed on the same stave. If `!` is used with a fermata or trill or other ornament, the sign is printed below instead of above the note. String bowing marks are not affected by the use of the `!` option.

41.14 Stem lengths

The note option consisting of the letters `sl` followed by a number is meaningful for notes shorter than a semibreve. It specifies a lengthening or shortening of the note’s stem. The number specifies the amount by which the stem is to be changed; positive numbers cause lengthening, negative numbers cause shortening. For example,

```
a\sl3.4\  b\sl-1.2\
```

lengthens the stem of the first note by 3.4 points and shortens the stem of the second by 1.2 points. PMW maintains a minimum stem length beyond which shortening is ignored.

The **shortenstems** heading directive can be used to request PMW automatically to shorten note stems that point in the ‘wrong’ direction – if this is happening, any explicit adjustment is added to the automatically computed value.

If a note which is part of a set of beamed notes has its stem length changed, this may cause the vertical position of the beam to change. However, it is not always easy to see which is the note whose stem actually determines the beam’s vertical position. A better way to adjust beams is to use the **[beammove]** directive.

41.15 Moving accents and ornaments

It is possible to move all accents and ornaments up and down, or left and right. This is done by placing `/u`, `/d`, `/l`, or `/r`, as appropriate, followed by a number of points, after the accent or ornament specification. For example,

```
a\./u4\ g\f/u10\
```

raises the staccato dot by 4 points and the fermata by 10 points.

For both accents and ornaments, the vertical movement specified is scaled by the relative size of the stave. Moving an accent does not affect the placement of anything else. For example, if there is text below a note with an accent that is also below it, moving the accent does not affect the vertical position of the text.

There is a possibility of ambiguity if a tremolo and a moved accent or ornament are specified on the same note, as the tremolo notation is a slash. To avoid this, the tremolo must be specified before the fermata: `g\/f\` is correct, but `g\f/\` causes an error, because it is taken as a fermata with an incomplete movement request.

41.16 Masquerading notes

For special effects (for example, tremolos between notes – see **[tremolo]**) it is sometimes desirable to print a note or rest of one kind in place of another, for example a crotchet instead of a quaver, or a breve instead of a semibreve. PMW supports this kind of *masquerading*. It is requested by the letter `m` in the options part of the note, and the type of note required is indicated in the same way as for normal notes.

When a masquerade is requested, an augmentation dot can be requested with it, and if it is not, no dot is printed, even if the original note is augmented. The ability to add augmentation dots makes it easier to print renaissance music in the style with a dot before a bar line instead of a tie to a quaver in the next bar. Thus

`G+\M++\`	prints a breve instead of a semibreve
`g-\m\`	prints a crotchet instead of a quaver
`g.\M\`	prints an undotted minim instead of a dotted crotchet
`g\m.\`	adds a dot to a crotchet without lengthening it

The only effect of masquerading is to substitute a different note for printing; the position of the note is not affected.

If the note is beamed, this option is restricted in its use: the only available facility is to print a minim notehead instead of a crotchet notehead. For example:

```
g-\M\ b- d'-
```

Masquerade requests for noteheads other than minims are ignored within beams. However, masqueraded *rests* are not restricted within beamed groups. This makes it possible to print (unconventionally) a crotchet rest under a beam, by using a construction such as `r-\m\q-` within a beamed group.

41.17 Expression items on rests

Accent marks are not supported on rests, but pause marks (fermatae) are permitted. Other ornaments such as turns are allowed on invisible rests only. This gives a way of printing these characters on their own at particular positions in a bar.

41.18 Changing rest levels

A note option consisting of the letter `l` followed by a number is permitted for rests only. A negative number may be specified. This has the effect of moving the rest vertically up (for positive numbers) or down (for negative numbers) by the given amount. For example,

```
R\l4\
```

prints a minim rest on the fourth instead of the third line. If rests are generally to be printed at a non-standard level, the **[rlevel]** directive can be used to avoid having to give this option on every rest. If this option is used in conjunction with **[rlevel]**, the effect is cumulative.

41.19 Triplets and other irregular note groups

PMW uses brace characters (curly brackets) to enclose notes that form a group which is not a standard division of a larger note. The opening brace can be followed by a number indicating the non-standard division. If this is omitted, the group is assumed to be a triplet. For example

`{a b c}`	is a triplet of crotchets
`{2 g-a-}`	is a duplet of quavers
`{5 c-d-e-f-g-}`	is a group of five quavers

By default, PMW assumes that an irregular group is a division of one, two, four or eight notes of the appropriate larger size. For example,

```
{7 f-g-a-b-f-a-g-}
```

is taken as a division of two minims into seven quavers. However, a division of *three* minims into seven is notated on the musical stave in exactly the same manner – music notation is ambiguous in this respect. It is not possible to determine what a group of quavers with a ‘7’ above it actually means, without looking at the rest of the bar.

PMW is not capable of analysing bars in this detail. In some circumstances, therefore, it is necessary to specify two numbers after the opening curly brace, to specify how many of the larger notes are being subdivided. The numbers are separated by a stroke, the first being the additional information. For example

```
{3/7 f-g-a-b-f-a-g-}
```

is the notation for a division of three (instead of two) minims into seven quavers. This notation makes it possible to increase the length of the sub-divided regular group. For groups with larger numbers of notes, it is sometimes necessary to specify a smaller size of regular group than the default. This is possible by following the given number with a minus sign. For example,

```
{3-/11 g-g-g-g-g-g-g-g-g-g-g-}
```

specifies the division of three (instead of four) crotchets into eleven.

By default, PMW prints the marking for an irregular note group (e.g. the ‘3’ for a triplet) on the same side of the noteheads as the stems. If the notes are beamed, just the number is printed; if not, a horizontal ‘bracket’ is printed as well. The **tripletfont** directive is used to specify the size and type of font used.

The marking can be moved, forced to be above or below the stave, and the horizontal bracket can be omitted. The marking may also be totally suppressed.

The **[triplets]** directive can be used to set defaults for some of these options. For individual triplets, the following qualifiers may appear after the opening curly bracket, following any numbers that may be present:

`/a`	put mark above
`/a`<*n*>	put mark <*n*> points above
`/b`	put mark below
`/b`<*n*>	put mark <*n*> points below
`/n`	omit bracket
`/x`	suppress mark altogether
`/lx`	invert left-hand jog
`/rx`	invert right-hand jog
`/d`<*n*>	move mark down <*n*> points
`/l`<*n*>	move mark left <*n*> points
`/r`<*n*>	move mark right <*n*> points
`/u`<*n*>	move mark up <*n*> points
`/ld`<*n*>	move left end of bracket down <*n*> points
`/lu`<*n*>	move left end of bracket up <*n*> points
`/rd`<*n*>	move right end of bracket down <*n*> points
`/ru`<*n*>	move right end of bracket up <*n*> points

In fact, `/x` suppresses the marking in the default state only. If **[triplets]** is used to suppress all subsequent irregular note markings, `/x` causes the mark to be printed. In other words, it inverts the mark printing state.

When a dimension is given after `/a` or `b`, the value given is the position above or below the stave of the baseline of the numerical text for a horizontal bracket. Subsequent adjustment of either end of the bracket is then possible, as described above. If no dimension is given after `/a` or `/b`, the vertical position is computed from the positions of the notes that form the group.

The left and right movements are available only if no horizontal bracket is being printed; they are ignored otherwise. Here are some examples of the use of these options:

```
{3/5/d1 a-a-a-b-b-} {/b a-b-c-}
{/a/u3 dfg}  {2/d2/n a-a-}
```

If either of the `/a` or `/b` options is specified, it is assumed that the marking is being moved to the other side of the noteheads, and therefore the bracket is automatically added. The `/n` qualifier must be used if a bracket is not required in this circumstance.

By default, PMW draws the brackets for triplets and other irregular note groups horizontal. Occasionally a sloping bracket is required; these can be obtained by means of the `/lu`, `/ld`, `/ru`, and `/rd` options on the opening curly bracket. They have the effect of moving the left or right hand ends of the bracket up or down, respectively, by an amount specified after the option. For example:

```
{/ld10 dfa}
```

Very occasionally, when using coupled staves, it is useful to be able to alter the direction of the ‘jog’ at one end of a triplet bracket so that it points in the opposite direction to the jog at the other end. The qualifiers `/lx` and `/rx` request this, for the left-hand and right-hand jogs, respectively.

The appearance of an irregular note group does not of itself cause a break in the beaming, and an explicit beam break must be specified if required. Strictly, beam breaking indicators and the tie indicator are supposed to come immediately after the final note, before the terminating } character, but in fact PMW allows the } character to precede or follow the tie and beam indicators. Thus the following are all permitted

```
  {g=g=g=,}g=    {g=g=g=},g=
  {g=g=g=_}g     {g=g=g=}_g
```

When both a tie and beam break indicator are present, the } character must come either before or after both of them, not in between.

41.20 Ties and short slurs

Two adjacent notes may be tied, or a slur generated between them, by ending the first note with an underline character. PMW does not distinguish between a tie and a slur between two adjacent single notes, except that when the underline represents a tie (i.e. when the two notes have the same pitch), the stem direction of the second note is forced to be the same as that of the first note, and if a MIDI file is being generated, the tie is honoured. The stem forcing does not happen in the case of a short slur.

In contrast, when an underscore follows a chord, it causes tie lines to be drawn between notes of the same pitch in the chord and the following chord. Thus an underscore always represents one or more ties when it follows a chord. The **[slur]** directive must be used when slurs are required between adjacent chords.

If two notes (or chords) that form part of a beam are tied, it does not cause the beam to be broken. An explicit beam break must be specified if required.

Ties are normally printed on the opposite side of the noteheads to the stems. A tie on a single note can be forced to be above or below the notehead by adding the qualifier `/a` or `/b` after the underline character, for example,

```
  a_/a | a e'_/b | e'
```

Such an indication takes precedence over the **[ties]** stave directive, which sets a default position for all subsequent ties.

The same qualifiers are also available for chords, where they force *all* the tie marks to be drawn in the specified direction. It is also possible, for a chord, to specify that only some of the tie marks are to be drawn above or below the noteheads, the remainder appearing on the opposite side. This is done by inserting a digit between the `/` character and the letter which follows. For example, either of the notations

```
  (ace)_/1a   (ace)_/2b
```

indicates that one of the three tie marks is to be drawn above the noteheads, with the other two below.

Slurs that include chords, or span several single notes, must be specified using the **[slur]** stave directive.

41.21 Editorial and intermittent ties

Ties can be marked editorial, or printed as dashed or dotted, by means of the following qualifiers:

`/e`	editorial – a short line is drawn through the tie
`/i`	intermittent – that is, dashed
`/ip`	intermittent periods, that is, dotted

These are the same options as for slurs.

41.22 Hanging ties

Occasionally there is a requirement to print tie marks that do not end on another note or chord, but simply extend some distance to the right to indicate that the note or chord should be held on for some time. These can be notated by making the second note or chord invisible using the stave directive **[notes off]**. In the case of a chord, the ends of all the tie marks are vertically aligned in such cases.

To help with the positioning of the ends of this kind of tie, tie marks are allowed to continue over rests (usually invisible ones). For example,

```
  (cde)_ | qq [notes off] (cde) [notes on]
```

extends the ties to the position of the third crotchet in an otherwise empty bar.

41.23 Glissando marks

Glissando marks are available for single notes. They are *not* available for chords. (PMW will accept the notation for chords, but will not do the correct thing with it.) The glissando notation is an extension of the short slur notation. If a short slur mark (underscore) is followed by `/g` then a glissando line is drawn between the relevant notes. If both a slur and a glissando mark are required, then `/s` must be added. If the slur is being forced above or below with `/a` or `/b` then it is not necessary to use `/s`. For example:

`f_`	slur only
`f_/g`	glissando only
`f_/s/g`	glissando and slur
`f_/g/a`	glissando and slur, slur above

It may occasionally be necessary to insert extra space between notes that are joined by glissando marks.

41.24 Input short cuts

Some short cuts are available to reduce the amount of typing needed. They do not affect the appearance of the output in any way.

- The previous note or chord can be repeated exactly, within the same bar, by using the letter `x` instead of a note letter. Accidentals are *not* reprinted, though they apply if a MIDI file is being generated. An optional number may follow `x` to indicate the number of repetitions. A beam break or a tie may follow, and a subsequent `x` can be used for further repetitions. For example, a bar of eight quaver chords, broken in the middle, can be written like this:

  ```
    (e-gb) x3; x4 |
  ```

  (Beam breaking is described in the next chapter.) A rest may intervene between the original note and its copy, but they must both be in the same bar, and there must be no clef, octave, or transposition change between them.

- The previous note or chord's pitches can be copied, with a different note length and with different options, by using the letter `p` instead of a note letter, within the same bar. The length of the note is determined by the case of the letter `p` and any hyphens, equals, or plus signs that follow, and normal note options such as staccato, etc., may follow as well. As in the case of `x`, accidentals are not reprinted, and there must be no clef, octave, or transposition change between the original and the copy. For example, a dotted crotchet chord followed by a quaver chord of the same pitches can be notated thus:

  ```
    (d.fa) p-
  ```

`p` can be followed by another `p`, possibly with a different note length, and `x` may follow `p` and *vice versa*, provided they are all within the same bar.

For technical reasons, `x` and `p` are not available after notes that print their accidentals above or below.

42. Note beaming

PMW makes no beaming decisions of its own based on note length or position in the bar or any other criteria, but leaves it entirely up to the creator of the input file to specify what is required. The **beamthickness** heading directive can be used to set the thickness of line used for drawing beams.

42.1 Beam breaking

Notes and chords shorter than a crotchet are automatically beamed together unless they are separated in the input by one of the following beam breaking characters, which must always follow the end of a note, without any intervening spaces.

`;`	break all beams
`,`	break secondary beams only

For example, three pairs of beamed quavers could be notated in any of the following ways:

```
g-e-;f-a-;b-a-
g-e-; f-a-; b-a-
g- e-; f- a-; b- a-
```

A single digit may follow a comma to specify how many beams not to break – no digit is equivalent to 1. For example,

```
g=-a=-b=-,2 c'=-d'=-e'=-
```

results in two solid beams, with one broken in the middle. A value which is too large is simply reduced. A value of zero causes all beams to be broken; this is different to the normal semicolon beam break, because it causes the beams on either side of the break to align with each other.

If any other character terminates a note, the beam is not broken. The terminating character should follow the tie character (underscore) if present. If the note is the last of an irregular group (e.g. a set of triplets) then the character may appear after the closing curly bracket. In the case of a chord, a beam breaking character follows the closing bracket.

After a secondary beam break (introduced by a comma), a small amount of extra horizontal space (1.3 points) is inserted. Without this, the gap that has only a primary beam appears to be too narrow.

PMW automatically beams across rests that are shorter than a crotchet, unless a break in the beam is specified by the presence of a semicolon.

[Users of a previous version of PMW may be in the habit of using an alternative beam-breaking convention. This can be requested by the **oldbeambreak** directive, but it is no longer documented.]

42.2 Beaming over bar lines

Consistent syncopation employing beamed notes is more logical and more graphic when the beaming is carried across the bar lines. This effect can be achieved in PMW by following the bar character by the = character, for example,

```
[time 2/4] r- g g- |= g-; g g- |= g-; g r- |
```

This notation causes a beamed group to extend into the subsequent bar; in the example above, two pairs of beamed quavers are printed, each straddling a bar line. Without the = characters after the bars, each quaver would be printed separately.

A single beam can be carried over one bar line only; it will not be continued over a second. However, as the example shows, a bar may have carried-over beams at either end.

When such a beamed group occurs at the end of a system, the beam is drawn to the final bar line, and, on the next system, an 'incoming' beam is drawn to indicate the continuation. By default, this has the same slope as the first part of the beam. This can, however, be altered by means of the **[beamslope]** directive, placed at the start of the second bar. A **[beammove]** directive can also be used in this position, where it will affect only the continued portion of the beam.

Warning: Because PMW normally works on only one bar at a time, continued beams are not handled in quite the same way as other beams. In particular, the computation of stem directions for the notes takes place independently in each bar, taking into account only the notes in that bar. This means that in some cases the notes in the second bar will by default have their stems in the opposite direction to the rest of the beam. This is not usually what is wanted, and it can give rise to errors when PMW cannot fit the notes on both sides of the beam. In these cases, it is necessary to specify a stem direction explicitly on the first note of the second bar.

42.3 Beaming across rests at beam ends

A recent innovation in notation is the continuation of beams over rests, even when they are at the start or end of the beam. This is thought to be helpful in indicating rhythmic groupings.

PMW handles rests in the middle of beams automatically. To cause it to include suitable rests at the ends of beams within the beams, the heading directive **beamendrests** is provided. There is also **nobeamendrests**, which can be used to cancel this effect in a subsequent movement. Explicit beam breaks can be used to prevent an individual beam from covering a rest.

Vertical movement of the rests is taken into account when computing the position of the beam. As in all beams, this can be adjusted by means of the **[beammove]** directive.

Beams covering rests at the end may be continued over bar lines, as described in the previous section, but only if there is at least one non-rest in the first bar.

42.4 Accelerando and ritardando beams

In modern music, accelerandos and ritardandos are sometimes notated by beams which fan out or in. PMW has some simple support for printing these. The stave directives **[beamacc]** and **[beamrit]** specify that the next beamed group in the bar is of the appropriate type.

The notes of the group should normally be input as quavers. In most cases they are an irregular group and need to be enclosed in curly brackets, with an appropriate number. It is not usual to print the number, so this is normally turned off by means of the `/x` option on the irregular note group.

PMW prints accelerando and ritardando groups by drawing the primary beam as normal, then drawing one or two more inside beams, with one end fixed and the other getting nearer to the noteheads. By default, three beams in total are drawn, but this number can be changed in the **[beamacc]** or **[beamrit]** directive by following it with the number 2, in which case only two beams are drawn. For example:

```
[beamacc 2]
```

This setting lasts until the end of the stave, or until a subsequent **[beamacc]** or **[beamrit]** directive containing the number 3 is encountered.

The slope of the primary beam is the same as it would be for a conventional beamed group. If this is horizontal, a ‘fanned’ beam does not always look right, and it is necessary to use the **[beamslope]** directive to change it. For example,

```
[beamslope 0.1]  [beamacc] {5/x g-g-g-g-g-} |
[beamslope -0.1] [beamrit] {5/x g-g-g-g-g-} |
```

In some cases it is also necessary to move the beams away from the noteheads using the **[beammove]** directive.

42.5 Beams with notes on both sides

The stem direction which is determined for a beamed group by the rules described in chapter 43 is the *default*, that is, it is the direction used for those notes in the group whose direction is not otherwise specified. It is possible to have notes on the non-default side of the beam by requesting an explicit stem direction for them. This facility is of most use in two-stave keyboard parts, where the staves are ‘coupled’ (see the **[couple]** directive). For example, in the group

```
[stave 1 treble 1 couple down]
g`-f`-a-\sd\
```

the default stem direction for the beam is up as a result of the two low notes, but the third note is printed on the other side of the beam with its stem down.

If there is a run of notes on one side of the beam followed by a run of notes on the other side, the note option `\sw\` can be used to swap the default stem direction for the note on which it appears and for all subsequent notes in the beam, but only if the first note of the beam has its direction explicitly specified. For example,

```
  e`-\su\f`-g`-g-\sw\a-b-
```

causes the first three notes to have their stems up, and the last three to have their stems down. This option can be used as many times as necessary in a beam.

If `\su\` were not present on the first note, then `\sw\` could not be used, as the default direction is not known at the time it is processed (because it depends on the pitches of all the notes in the beam).

The arrangement of beams and beamlets for beams with notes on both sides follows the general principle of attempting to avoid ‘beam corners’ wherever possible. Some variation in this arrangement can be obtained by making use of secondary beam breaks.

When there are only two notes in a beam, it is almost always possible to print them with their stems going in opposite directions, even though sometimes this leads to extremely slanted beams. When there are more than two notes, however, it is sometimes not possible to find a way of positioning the beam if the notes are too close together in pitch. When this happens, PMW outputs an error message.

The slope of a beam can be forced by means of the **[beamslope]** directive, and the vertical position can be adjusted by **[beammove]**.

43. Stem directions

This chapter documents the default rules for choice of stem direction for notes and chords. Some variation in the rules can be made by means of the **stemswap** heading directive. The ‘stem swap level’ is normally the middle line of the stave, but can be changed by the **stemswaplevel** directive.

43.1 Preliminary

1. The ‘pitch’ of a chord, for stem-decision purposes, is the average pitch of its highest and lowest notes.

2. The ‘pitch’ of a beamed group, for stem-decision purposes, is the pitch of the note which is farthest away from the stem swap level.

3. Stem directions are computed for all notes, even breves and semibreves. In the case of these long notes the notional stem direction can affect the stems of subsequent or previous notes, and also the printing of chords containing adjacent notes.

43.2 Rules for non-beamed notes and chords

These are in order of priority. ‘The previous note’ includes the last note of a previous beamed group, if relevant.

N1. If an explicit stem direction is specified on a note, it is used.

N2. If a default is set by the stave directive **[stems up]** or **[stems down]**, it is used.

N3. If the note is tied to the previous note, i.e. the previous note is followed by an underscore and has the same pitch, then the same direction as the previous note is used, even if this note is the first in a bar, provided the previous note’s direction does not depend on this note’s.

N4. If the note is above or below the stem swap level, its stem goes down or up, respectively.

N5. (The note is at the stem swap level.) If the note is the first in the bar, or if all preceding notes have used this rule, its stem goes the same way as the next note in the bar that does not use this rule. If there are no more such notes in the bar, its stem goes the same way as the last note of the previous bar. If this is the first bar of the piece, the stem goes up.

N6. The stem goes the same way as the previous note.

43.3 Rules for beamed groups

B1. If the stem direction of the first note in the group is forced by N1, N2, or N3 above, then that direction is used as the default for the group.

B2. If the ‘pitch’ of the beamed group is above or below the stem swap level, the stems go down or up, respectively, by default.

B3. The default stem direction is taken from the previous note. If there isn’t one, the stems go upwards.

Normally, all the notes in a beam are printed on the same side of the beam, with their stems in the default direction for the beam, but it is possible to specify that some are to be printed on the other side of the beam (see section 42.5).

44. Text strings in stave data

The following chapter gives details of the special facilities that are applicable only to underlay or overlay text, that is, the sung words in a voice part. The current chapter applies to text in general, with some particular features that are relevant only for non-underlay text.

Stave text strings are coded in among the notes of a stave, and are, like all strings, enclosed in double quote characters. The escape character conventions using the backslash character that apply to all PMW strings are relevant (see chapter 34). In particular, within any text string, the font can be changed by the use of the appropriate escape sequences.

Rehearsal markings are a special form of text and are specified in a slightly different manner (see chapter 46).

By default, text strings are printed below the stave in an italic font, and positioned according to the following note. The **[textfont]** directive can be used to specify a default font for ordinary (i.e. not underlay, overlay, or figured bass) text.

The **[text]** directive provides a way of changing the default position of the text to be above the stave, rather than below; it can specify a fixed position (above or below the stave) or allow the position to be determined by PMW. Alternatively, **[text]** can specify that unqualified strings are underlay, overlay, or figured bass text. Any individual string can always be explicitly qualified to indicate its type.

Underlay, overlay, and figured bass text is by default printed in the roman typeface. The directives **[underlayfont]**, **[overlayfont]**, and **[fbfont]** can be used to change the default font for these kinds of text.

The closing double-quote of the string may be followed by one or more options, separated from the quote and from each other by slash characters. The following are available:

`/a`	print above the stave	
`/a`<*n*>	print at fixed distance above the stave	
`/ao`	print above the stave, at the overlay position	
`/b`	print below the stave	
`/b`<*n*>	print at fixed distance below the stave	
`/bu`	print below the stave at the underlay position	
`/m`	print below the stave, in the middle	
`/ul`	this text string is underlay	
`/ol`	this text string is overlay	
`/fb`	this text string is figured bass	
`/h`	position halfway between notes	
`/bar`	position at start of bar	ignored for underlay/overlay
`/ts`	position at time signature	ignored for underlay/overlay
`/c`	centre the text	ignored for underlay/overlay
`/e`	align end of text	ignored for underlay/overlay
`/nc`	do not centre	ignored for underlay/overlay
`/ne`	do not align the end	ignored for underlay/overlay
`/box`	print enclosed in a box	ignored for underlay/overlay
`/ring`	print enclosed in a ring	ignored for underlay/overlay
`/rot`<*n*>	rotate by <*n*> degrees	ignored for underlay/overlay
`/s`<*n*>	print using size <*n*>, where <*n*> is between 1 and 12	
`/u`<*n*>	move up <*n*> points	
`/d`<*n*>	move down <*n*> points	
`/l`<*n*>	move left <*n*> points	
`/r`<*n*>	move right <*n*> points	
`/ps`	insert raw PostScript (for experts only)	

If any of the movement options are repeated on a string, their effect is cumulative. Thus, for example,

```
"allargando"/u6/d2
```

has the same effect as

```
"allargando"/u4
```

If more than one of `/a`, `/ao`, `/b`, `/bu`, `/m`, `/ul`, `/ol`, or `/fb` is present, the last one takes precedence. If none of them are present, the string type is taken from the last **[text]** directive. If **[text]** has not been used on the current stave, then `/b` is assumed.

There is an important difference between `/bu` and `/ul`, and similarly between `/ao` and `/ol`. When `/bu` is specified, the text is treated as non-underlay text, but its default vertical position is the underlay level. This contrasts with `/ul`, which indicates that the text is underlay, and subject to special processing as described in the next chapter.

The `/m` option is like `/b`, except that the default vertical position of the text is in the middle of the space between the current stave and the one below it, provided this is lower than than the normal `/b` position would be. This is useful when printing dynamic markings in keybord parts.

If two over-printing staves are being used for a keyboard part, text with the `/m` option may appear with either of them, because if the space after the current stave is set to zero, the space for the next stave is used when positioning such text.

The default vertical position of text is adjusted to take account of the next note, unless it is forced to the overlay or underlay level by `/ol` or `/ul`, or to an absolute position by `/a<`*n*`>` or `/b<`*n*`>`. For example,

```
"at underlay level"/ul
"six points above the stave"/a6
"twenty points below the stave"/b20
```

Subsequent appearances of `/u` or `/d` on the string can always be used to adjust the level. This can be useful if the original level is specified in a macro.

44.1 Horizontal alignment

The alignment of underlay and overlay strings is described in the next chapter.

By default, a non-underlay text string is printed with its first character aligned with the left-hand edge of the next note or rest in the bar, or with the bar line, if there are no following notes or rests in the bar.

However, if `/bar` is present, the alignment point is the previous bar line, or the start of the system for the first bar in a system. If the `/ts` option is present, the alignment point is the time signature at the start of the bar. If there isn't one, the alignment point is the first note in the bar. For both `/bar` and `/ts` the vertical position of the string still depends on the note that follows it.

If the `/e` qualifier is present on the text string, it is the end of the string that is aligned with the alignment point. The `/ne` option can be used on text strings to cancel the effect of a previous `/e`. This can be useful for overriding options on strings defined as macros.

If the `/c` qualifier is present, the text is centred at the alignment point. If this is used on text immediately before a whole bar rest that is centred in the bar, then the text is centred in the bar. This applies to both visible and invisible whole bar rests. The `/nc` option can be used on text strings to cancel the effect of a previous `/c`. This can be useful for overriding options on strings defined as macros.

The `/h` option causes the alignment point to be halfway between the next note or rest and the note or rest that follows, or the end of the bar if there is only one note or rest following in the bar. The `/e` and `/c` options can be combined with `/h` to specify end or centre alignment at the halfway position, respectively. If no notes follow the text string in the bar, `/h` has no effect, and it is also ignored if `/bar` or `/ts` are present.

Positions other than the halfway point can be specified by a number given after `/h`. For example, `/h0.75` specifies the three-quarter point between the next note or rest and the one following. The `/h` option can be used with underlay and overlay strings, but it applies only to the first syllable of such strings.

44.2 Enclosed text

The `/box` and `/ring` options are applicable only to non-underlay text. The longer the string is, the more elliptical a ring will be. For a single character, the ring is approximately circular.

44.3 Text sizes

The `/s` option refers to the sizes of text defined by the **textsize** heading directive; `/s1` specifies the first size, `/s2` specifies the second size, and so on. If, for example, the heading directive

```
textsizes 10.5 11 7.6
```

has been used, then

```
"eleven point"/s2
```

would print at a size of 11 points. By default, text is printed in the first size, unless it is underlay or figured bass, which have their own default sizes (set by the **underlaysize** and **fbsize** directives). The `/s` option can, however, be used with underlay and figured bass text to specify a non-default size.

44.4 Rotated text

Stave text strings that are not underlay or overlay can be rotated through any angle by following the string with `/rot` and a number in degrees. Positive rotation is anticlockwise. For example:

```
"gliss"/rot40/a0/r4 c'_/g [space 8] c''
```

The centre of rotation is on the text baseline, at the left-hand end of the string.

44.5 PostScript text

If the `/ps` qualifier appears on a text string, the contents are assumed to be raw PostScript which is to be inserted into PMW output at the point where a text string would have been output. This facility is for PostScript experts only. The string is preceded by a call to the PostScript **gsave** operator and followed by **grestore**. The origin is the x-coordinate at which a text string would have been output, and the bottom line of the stave plus any vertical adjustment that is specified for the string. No processing is done on the string; any backslash characters it may contain are not treated specially.

45. Vocal underlay and overlay text (lyrics)

PMW supports both underlay (words under the stave) and overlay (words over the stave). Overlay is comparatively rare, and to save clumsy repetition of ‘underlay or overlay’ in what follows, the description is written mainly in terms of underlay only. However, all the features are equally applicable to overlay.

A text string is marked as underlay or overlay either by using the `/ul` or `/ol` options, or by using the **[text]** directive to set underlay or overlay as the default, and then not using any of the other text type options (`/a`, `/b`, etc.)

The usual escape character conventions apply to underlay text, and in addition, the characters `#` (sharp), `-` (hyphen), `=` (equals), and `^` (circumflex) have special meanings.

45.1 Underlay syllables

Underlay can be input one syllable at a time, each syllable preceding the note to which it refers. This permits the maximum possible control, since each syllable can be moved up, down, left or right as required. However, it is normally easier to input underlay in longer strings.

If a string of underlay text contains space characters it is automatically split up by PMW and allocated to the notes which follow it. Rests are excluded from this process (with one exception, which is described in section 45.5). As a simple example of this facility,

```
  "God save our" g g a |
```

would be an appropriate way to start the British National Anthem. Each space delimits a word, and each word is associated with one note. PMW does not check that the number of words matches the number of notes, except that it warns if words are left over at the end of the stave.

Each syllable of underlay text is normally centred horizontally about the next note in the bar. Sometimes it is necessary to move words slightly to the left or right. A convenient way to do this is to include the character # in the underlay string. This character prints as a space, but does not count as a space when PMW is splitting up the text into words. The width of a printed space is 5 points (in a 10-point font), so, for example,

```
  "God# save #our" g g a |
```

would print ‘God’ 2.5 points to the left of where it would otherwise appear, and ‘our’ 2.5 points to the right.

Sometimes several words are required to be printed under a single note, and only the first is to be centred on it. The # character can be used to separate such words, to prevent them being assigned to separate notes. If the character `^` (circumflex) appears in an underlay syllable, then only those characters to the left of it are counted when the string is being centred. The circumflex itself is not printed. For example, consider

```
  "Glory^#be#to#Thee, O God."
  G+ g #F
```

The words ‘Glory be to Thee’ are all associated with the semibreve, but because of the circumflex, ‘Glory’ is centred under it, and the rest stick out to the right. If a syllable starts with a circumflex, it is not centred, but instead starts at the note position.

If two circumflex characters are present in a syllable, then the text between them is centred at the note position. This makes it possible to cause text to stick out to the left of a note.

When a word consists of more than one syllable, the syllable breaks must be delimited by hyphens:

```
  "God save our gra-cious Queen" g g a | f. g- a |
```

When a syllable is terminated by a hyphen, PMW prints one or more hyphens between it and the next syllable, depending on their distance apart.

The heading directive **hyphenthreshold** can be used to specify the distance between syllables at which more than one hyphen will be used. The default value is 50 points. If the space is less than this, a single hyphen is printed, centred in the space. Otherwise, a number of hyphens are printed, the distance between them being the threshold value divided by three.

It is possible to cause PMW to print en-dash characters (or any other characters) as ‘hyphens’ between syllables of underlaid text. See the **hyphenstring** heading directive for details. Whatever is printed, the syllable separator in the input remains a single hyphen.

When a syllable extends over more than one note, equals characters must be inserted into the input string, one for each extra note. This includes tied notes, because PMW does not distinguish between ties and short slurs. For example,

```
"glo-==ri-a"
F. | B`. | C. | E. | F. |
```

```
"glo-========ri-a"
a-e-a- | b-c′=b=a=b= | c′- c′- b- |
```

PMW automatically draws an extender line after a word which ends with an equals, finishing underneath the last note, provided that the line is of reasonable length. The vertical position of the extender level is just below the baseline of the text, but this can be altered (see section 38.34).

By default, PMW centres all underlay and overlay syllables at the position of their respective notes. The **underlaystyle** directive (section 38.130) can be used to request PMW to align underlay and overlay multinote syllables flush left with the initial note. The circumflex character can still be used to specify that particular multinote syllables be centred.

Text for two or more verses (up to any number) can be specified in multi-syllable fashion before the relevant notes by giving each verse as a separate string:

```
"God save our gra-cious Queen"
"Thy choi-cest gifts in store"
g a a | f. g- a |
```

The vertical distance between verses can be altered by means of the **underlaydepth** and **overlaydepth** directives, which control independent values. For overlay, the second verse is printed above the first one, and so on.

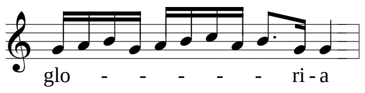
The multi-syllable underlay feature in PMW is purely an alternative input notation. The effect is exactly as if the individual syllables were input immediately preceding the notes under which they are printed. The following two alternative examples produce the same output:

```
"God= save our Queen" e′-c′- b a | G. |
```

```
"God=" e′- c′- "save" b "our" a | "Queen" G. |
```

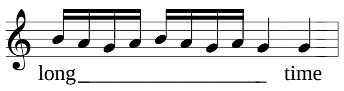
If an underlay string ends with a hyphen, the equals characters can be omitted; PMW automatically prints a sequence of hyphens up to the next underlay syllable. This can be useful when syllables last for many notes, for example:

```
"glo-" g=a=b=g=; a=b=c′=a=; b-. "ri-a" g= b
```



If the final syllable of a word extends over many notes, only a single equals character is needed if it is at the end of an input string. However, because extender lines are drawn only as far as the last note for the syllable, rather than to the next underlaid word, it is necessary to supply the final equals character at the start of the next string, to tell PMW which is the final note for the syllable. For example:

```
"long=" b=a=g=a=; b=a=g=a=; "= time" g g
```



If there are more notes on the stave, but no more words, then a syllable consisting just of a # character can be used to stop PMW drawing an extender line further than is required.

If any positioning qualifiers are specified on an underlay input string (`/u`, `/d`, `/l`, or `/r`) then the same amount of movement applies to each of the syllables in the string independently. Specifying vertical movement in this way can sometimes be a convenient alternative to the use of the **[ulevel]** directive.

Warning: There is one important restriction on the use of multi-syllable underlay text strings. Because they are processed during the input stage of PMW, they cannot in general be used successfully with the notation for repeating bars. Each syllable in such a string is allocated to *the next note read from the input*, but a bar repeat count simply duplicates the bar in which it appears, without reading any more notes.

45.2 Underlay and overlay fonts

Two separate sets of fonts are provided for underlaid and overlaid text, and the size of these can be set independently of the other text fonts by the **underlaysize** and **overlaysize** directives. However, individual underlay or overlay strings can specify different sizes by means of the `/s` option.

45.3 Underlay and overlay levels

Text that is marked as part of the underlay or overlay is always printed at the same level below or above the stave in any one system of staves; the line of words is always horizontal. PMW chooses an underlay and an overlay level for each line of music according to the notes that appear on that line, but these can be overridden by means of the **[ulevel]** and **[olevel]** directives. Individual words or syllables can be moved up or down relative to the standard level by means of the `/u` and `/d` qualifiers.

45.4 Underlay and overlay spreading

PMW is capable of spreading out the notes of a piece to take into account the width of underlaid or overlaid words. This facility should be used with care, because the music can become very poorly spaced if the width of the words is allowed to have too much influence on the separation of the notes.

The spreading facility operates only within individual bars, and not between bars. It applies only to underlay or overlay text, not to other kinds of text. ‘Hard spaces’ (notated by sharp sign characters) in the text are taken into account when examining the available space. The minimum space allowed between syllables is one space character in the appropriate font.

There is a heading directive, **nospreadunderlay**, which disables this facility for both underlay and overlay, and it is recommended that those who place great importance on the spacing of notes should use it.

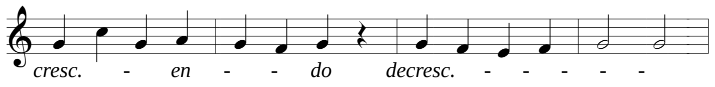
The automatic facility is intended as an insurance for less demanding users against the occasional wide syllable. In order that it function in this way, it is important that a suitable note spacing be set, and a suitable size of underlay or overlay font be chosen, such that most of the syllables fit on the line without the need for any adjustment of the notes. The default setup is not always suitable for music with words; multiplying the note spacing by 1.2 and choosing a font size of 9.5 is a better default.

45.5 Other uses of underlay and overlay

The underlay and overlay facilities can be used for printing things other than the words of a vocal part. It is common, for example, for the word *crescendo* to be printed in a stretched-out manner, in the style of underlay, or alternatively, for an abbreviation such as *cresc.* to be followed by a number of hyphens. In the latter case, the final ‘syllable’ of the word does not exist, but it can be specified as a single sharp character, which does not cause anything to be printed (because # prints as a space in underlay).

The text can be given as a single string, with equals characters for each note under which hyphens are to be drawn, or each syllable can be given with the relevant note. In the latter style, the final syllable can be moved left or right to adjust the end point of the hyphens. Here is a simple example of both kinds of approach:

```
"\it\cresc.-==en-==do"/ul gc'ga | gfgr |
"\it\decresc.-"/ul gfef | G "#"/ul/r6 G |
```



Normally, underlay (and overlay) syllables cannot be associated with rests, but because a final empty syllable is often required when using underlay to print rows of dashes, and ending at a rest is common, an exception has been made for the string `"#"`, which should not occur in normal underlay usage. If this string is specified as underlay or overlay, and immediately precedes a rest, it will be associated with the rest rather than the following note. This exception applies only to strings consisting of a single # character.

PMW supports multiple verses, so there is no difficulty in mixing this kind of usage with real vocal words, though normally the vocal line would be printed as underlay and the other text as overlay.

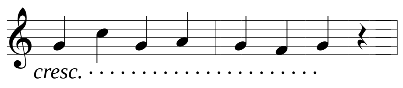
Hyphen strings for underlay are printed with hyphens fairly far apart, and at varying separations. Sometimes a more uniform hyphen separation is required, and some editors prefer some other character to the hyphen after items like *cresc*. Some additional features are provided for use strings in these cases.

If a second string is provided as an option to an underlay or overlay string (i.e. following a slash) it is used instead of hyphens to separate the syllables of a word. The string is repeated as many times as possible in the available space. This option should be given after any other options for the main string; in particular it must follow the `/ul` or `/ol` option.

The default font for the second string is the default underlay or overlay font, as appropriate, and the default size is the size of the first string. However, the second string may be followed by `/s` and a number to specify a different size.

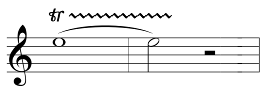
The second string may also be followed by `/u` or `/d` to specify that it is to be moved up or down, relative to the following syllable. In this example, a full stop is used as the repeating character, and it is moved up so as to be approximately at the middle of the letters:

```
"\it\cresc.-"/ul/" ."/u2  gc’ga | gf "#"/ul gr |
```



The second string is a normal PMW string, and may contain font changes and other escape sequences. Hence it can be used to print trill signs followed by wiggly lines, by selecting the appropriate characters from the music font:

```
"\*136\-"/ol/"\*96\" E’+_ | "#"/ol/r8 E’R |
```



Character 136 is the *tr* character, and character 96 is the tilde (~), which gives a wiggly line when repeated. The invisible final syllable is moved right eight points to ensure that the ‘hyphens’ (i.e. the wiggly line) cover the final note.

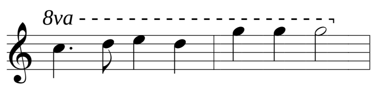
If such features are required in several places in a piece, the best thing to do is to use the macro facility to save having to type the complicated strings each time. This approach is taken in subsequent examples below.

The conventional octave marking of *8va* followed by a line of dashes can be printed in this way. However, it is normal to print a small ‘jog’ on the final dash to indicate the end of the section. To achieve this, an additional feature has been provided.

If an underlay or overlay option string contains a vertical bar character, only those characters to the left of the vertical bar are used as the repeating sequence, but those characters to the right of the bar are printed at the end of the sequence, once. (If, by some chance, a real vertical bar is required to be repeated, it can be specified as character number 124.)

There are some angle-shaped characters in the music font which can be used for printing the ‘jogs’, as follows:

```
*define s8 "\it\8va-"/ol/" -| \mf\\159\"/u0.3
*define e8 "#"/ol/r8
&s8 c’.d’-e’d’ | g’g’ &e8 G’ |
```



One further feature is available to cope with repeated strings that extend over the end of a musical system. If another optional string is given, it is printed at the start of each continuation line, before the start of the repeating strings. The only option permitted after this string is `/s`, to set its size (which defaults to the size of the original underlay or overlay string).

Using this feature to cause a small ‘8’ to be printed at the start of continuation lines, the macro definition from the above example becomes:

```
  *define s8 "\it\8va-"/ol/" -| \mf\\159\"/u0.3/"\it\8"/s2
```

where it is assumed that a suitable size is defined using the **textsizes** directive.

46. Rehearsal markings

Rehearsal markings are specified as text items enclosed in square brackets. The text may be longer than one character. It is printed above the stave, and by default is enclosed in a rectangular box, and printed in bold type. The **rehearsalmarks** directive can be used to change the size of the font, and to specify printing inside a ring instead of a box, or printing with no enclosure.

If necessary, a rehearal mark can be moved up, down, left or right, in the same manner as other text. For example:

```
["A"/u2]
```

Normally, a rehearsal marking is given at the start of a bar, and in this case it is printed immediately to the right of the preceding bar line (except at the left-hand side of the page). If a rehearsal marking is given in the middle of a bar, it is aligned horizontally with the next note, exactly as for other text.

Rehearsal markings are normally printed above the top stave of a score only, though in very large scores they are sometimes repeated part of the way down. If parts are to be extracted from a score, then the rehearsal markings should be specified on stave 0, as they will then be printed above the top stave, whichever staves are selected for printing. Chapter 35 gives details of the stave 0 facility.

47. Stave directives

This chapter describes the directives that can appear interspersed with the notes and rests of a stave. Each directive must be enclosed in square brackets, though if there are several in a row, a single set of brackets suffices. After the first three, which do not involve the use of an alphabetic name, the directives are given in alphabetical order.

Clefs are specified by directives which are the names of the clefs, and these appear in their alphabetic positions below. The following are provided:

alto	C3
baritone	F3
bass	F4
cbaritone	C5
contrabass	F4 with *8* below
deepbass	F5
hclef	percussion H clef
mezzo	C2
noclef	nothing printed
soprabass	F4 with *8* above
soprano	C1
tenor	C4
treble	G2
trebledescant	G2 with *8* above
trebletenor	G2 with *8* below
trebletenorb	G2 with *(8)* below

47.1 Repeated expression marks

If a long sequence of notes are all to be marked staccato, or with accents, this can be specified by giving the expression syntax for one note inside square brackets. For example

```
[\.\] a b c d
```

causes all the notes to be marked staccato. The only characters that may appear within backslashes in this context are

`.`	staccato
`-`	accent
`>`	horizontal wedge accent
`v`	small, closed vertical wedge
`V`	large, open vertical wedge
`'`	‘start of bar’ accent
`o`	ring (harmonic)
`d`	string down bow mark
`u`	string up bow mark
`a`<*n*>	accent number <*n*>
`/`	single tremolo mark
`//`	double tremolo mark
`///`	triple tremolo mark
`!`	put accents on other side of notes

To cancel a setting, two backslashes with nothing between them should be given as a directive (in square brackets). Cancellation can also be carried out for an individual note by means of the note option letter `x`. For example, to print a non-staccato note in the middle of a run of staccato notes, the following could be used:

```
[\.\] a b c d\x\ e f g [\\]
```

The note `d` would be printed without a staccato dot. Expression/option items are processed from left to right; if there are two or more things being defaulted, one can be put back again after the `x` if required.

47.2 Repeated bars

A bar which is repeated in the input need only be input once. The appearance of a number enclosed in square brackets causes those items to the right of it in the bar to be repeated that number of times. This is most commonly used for repeated rest bars, but it can be used with any bar. For example,

```
[45] R! |
[key C] [10] R! |
```

In the second example, the key signature is printed in the first bar only. If it had followed [10] it would have been printed in every bar.

There is danger of confusion between repeated bars and rehearsal markings. Accidental omission of the quotes from a numerical rehearsal marking such as

```
["42"]
```

can lead to some very strange effects.

Warning: Repeated input bars should not be used with multi-syllable underlay texts, as the syllables are apportioned to notes as they are read from the input, and the repeated bars are not re-read.

47.3 [1st], [2nd], etc.

First and second time bars (and third and fourth, etc. if needed) are specified by enclosing the number, followed by one of the sequences ‘st’, ‘nd’, ‘rd’ or ‘th’ in square brackets at the start of a bar. The marking for the final one is terminated by the appearance of the directive **[all]**. For example

```
[1st] g a b g :) | [2nd] g a b c′ | [all] b a g f
```

More than one bar of music may appear between the items. The **[all]** directive must not be present if the piece ends with the second time bar.

Often these markings are printed above the top stave of a score only. If parts are to be extracted from a score, then the rehearsal markings should be specified on stave 0, as they will then be printed above the top stave, whichever staves are selected for printing. Chapter 35 gives details of the stave 0 facility.

It is possible to specify vertical movements for 1st and 2nd time bar markings, to cope with unusual cases. This is done by entering `/u` or `/d` followed by a number in the directive. For example,

```
[1st/u4]
```

specifies a first time bar whose marking is to be 4 points higher than it would be by default.

PMW normally puts the second time marking at the same level as the first, unless high notes in the second time bar force it up.

More than one of these markings can be given in the same bar, for example:

```
[1st] [2nd] Grg | [3rd] GR |
```

When this is done, the numbers are printed with a comma and a space between them. Any movement qualifiers must be specified on the first one.

The left-hand ends of these markings can be moved left and right as well as up and down by means of `/l` and `/r` qualifiers. It is also possible to change the text that is printed by supplying a text string after a slash, for example:

```
[1st/"primero"]
```

If there are several in one bar, separate strings can be supplied for each of them. The font size is set by the **repeatbarfont** directive, which also sets the default font to be used.

47.4 [All]

See the immediately preceding section.

47.5 [Alto]

This specifies a C clef with its centre on the third stave line. If it is given without a parameter, no change is made to the current default octave. However, it may be followed by a number to indicate a new setting for the current octave, for example,

```
[alto 1]
```

A clef setting has no bearing on the interpretation of the pitch of the notes that go to make up a part (apart from the octave setting). Changing the clef directive at the start of a part causes the music to be printed out in the new clef, but at the same absolute pitch as before.

Clef changes in the middle of a stave that are not in the middle of a bar are normally notated immediately before a bar line rather than immediately after. The **clefsize** heading directive is used to specify the size of such clefs.

47.6 [Assume]

When an overprinted stave contains a sequence of skipped bars (see **[skip]**), the clef, key and time signature for its partner stave may have changed while its bars were being skipped. The **[assume]** directive can be used to set these things without causing anything to be printed. For example,

```
[skip 60] [assume bass 0] gbc |
```

has the effect of changing the stave into the bass clef so that 'gbc' are printed in this clef, and a bass clef is printed at the next start of line, but no clef is printed where the directive occurs. Similar syntax is used for setting the key and the time:

```
[assume key E$]
[assume time 3/4]
```

The use of this directive is not confined to overprinted staves.

47.7 [Baritone]

This specifies an F clef based on the third stave line. See **[alto]** for further details of clef directives.

47.8 [Barlinestyle]

This directive must be followed by a number, and it sets the bar line style for subsequent bar lines in the stave. See section 38.7 for details of the available styles.

47.9 [Barnumber]

The heading directive **barnumbers** (see section 38.9) is used to request PMW to number bars automatically. The stave directive **[barnumber]** is used to control the printing of numbers for individual bars.

If **[barnumber]** appears without any parameters in the stave data, then a number is printed for the current bar, independently of the overall setting. The size of font used and whether or not the number is printed in a box or ring is controlled by the heading directive. (The default is not to use boxes, and the default size is 10 points.)

The position of the bar number can be altered by following the directive with a slash, one of the letters `l`, `r`, `u`, or `d`, and a number. This is sometimes necessary when there are notes on high ledger lines at the start of a numbered bar. For example,

```
[barnumber/l10/d5]
```

prints a number on the current bar, 10 points to the left and 5 points down from where it would appear by default.

If **[barnumber]** appears followed by the word 'off' then no bar number is printed for the current bar, even if the heading directive implies there should be one.

47.10 [Bass]

This specifies a bass clef. See **[alto]** for further details of clef directives.

47.11 [Beamacc]

This directive causes the next beam to be drawn as an accelerando beam. See section 42.4 for details.

47.12 [Beammove]

This directive, which takes a single number as its parameter, causes the following beam to be moved vertically without altering its slope. A positive number moves it upwards, and a negative one downwards. An attempt to move a beam too near the noteheads may give strange results.

Use of this directive is preferable to adjusting the stem length of one or more notes in the beam, as it is not always clear which notes in the beam are those whose stems control the beam position.

47.13 [Beamrit]

This directive causes the next beam to be drawn as an ritardando beam. See section 42.4 for details.

47.14 [Beamslope]

PMW contains rules for choosing the slope of a beamed group which usually have the right effect. However, it is possible to override them by means of the **[beamslope]** stave directive. This directive takes as its parameter a number specifying the slope of the next beamed group on the current stave. For example,

```
[beamslope 0.2] g-g-
[beamslope 0] c-g-
[beamslope -0.1] g-c'-
```

Positive slopes go upwards to the right, negative ones downwards. A slope of zero specifies a horizontal beam. The values given are in the conventional form for gradients, with a slope of 1.0 giving an angle of 45 degrees. When a beam's slope is specified explicitly, it overrides the setting of the maximum beam slope (see **maxbeamslope**).

When a beam has notes on either side of it, it may not be possible to use the specified slope because of the position of the notes. In this case, the default rules will come into play again and a smaller slope will be chosen.

47.15 [Bottommargin]

This directive provides a way of changing the value given by the **bottommargin** heading directive for a single page only. If there is more than one occurrence on the same page, the last value is used. To leave 30 points at the bottom of one particular page, for example, use

```
[bottommargin 30]
```

in any bar on that page.

47.16 [Bowing]

String bowing marks are normally printed above the stave. The **[bowing]** directive is provided for changing this. It must be followed by one of the words 'above' or 'below'.

47.17 [Breakbarline]

An occurrence of this directive causes the bar line at the end of the current bar not to be extended downwards onto the stave below, unless it is at the end of a system. See also **[unbreakbarline]**.

47.18 [cbaritone]

This specifies a C-clef on the 5-th stave line. See **[alto]** for further details of clef directives.

47.19 [Comma]

The **[comma]** directive causes PMW to insert a comma pause mark above the current stave.

47.20 [Contrabass]

This specifies a bass clef with a little ‘8’ printed below it. See **[alto]** for further details of clef directives.

47.21 [Copyzero]

This directive takes a dimension as an argument, and adjusts the vertical level of any stave zero material in the current bar when stave zero is printed at the level of the current stave. For example,

```
[copyzero 4]
```

raises the stave zero material in the current bar by 4 points.

It is not necessary for there to be an instance of the **copyzero** heading directive specifying the current stave for **[copyzero]** to take effect. In the default case, **[copyzero]** takes effect whenever the stave in which it appears is the top stave of a system.

When first and second time bar markings are specified in stave zero, and there is a need to adjust their height for certain staves, it should be noted that the markings are drawn when the bar in which their end point is determined is processed. Consequently, it is that bar in which **[copyzero]** should appear. The same applies to slurs and lines (though they are rarely specified in stave zero).

47.22 [Couple]

A single musical part, notated as one PMW stave, can be spread across a pair of bass and treble staves when actually printed. This is commonly found in keyboard music. The directive

```
[couple up]
```

which should be given on a bass clef stave, specifies that notes higher than middle C should be printed on the stave above, which is assumed to be a treble clef stave. Similarly,

```
[couple down]
```

couples a treble clef stave to the stave below, while

```
[couple off]
```

terminates the coupling. A stave can be coupled only one way at once. However, there is no reason why a pair of staves should not both be simultaneously coupled to each other. An example of music printed in this way is given in section 15.4.

Occasionally it is desirable to cause individual notes that would not normally be printed on the coupled stave to be so printed. A notation for this is provided in the form of the `\c\` note option. For example,

```
[treble 1 couple down] g-e-c-\c\g`-
```

The middle C in this beam would normally remain on the original (upper) stave, but the use of `\c\` forces it down onto the lower one.

If the `\c\` option is used when coupling is not in force, the note is coupled upwards if it is on or above the centre line of the stave; otherwise it is coupled downwards.

Similarly, there is a note option `\h\` (for ‘here’) which prevents a note that would normally move onto the other stave when coupling is in force from so doing.

Warning: Coupling requires the spacing between the staves to be a multiple of 4 points if it is to work properly in all circumstances. The default spacing of 44 points satisfies this requirement.

47.23 [Cue]

The directive **[cue]** causes the subsequent notes of the current bar, on the current stave, to be printed using the cue note font, instead of the normal font. Typically, the note spacing needs to be reduced as well. This feature is normally used only when single parts are being printed; the conditional features of PMW can be used to control this, as in the following example:

```
[35] R! |          @ 35 bars rest
*if score
  R! |             @ if full score, rest bar
*else
  [cue] [ns *1/2]  @ if cue bar halve note spacing
  "[flute]"/a      @ print above stave
  g a-g-f-e- e
  | [ns]           @ restore at next bar start
*fi                @ end conditional section
```

The effect of the **[cue]** directive is automatically cancelled at the end of the bar in which it appears, but it can also be explicitly cancelled by **[endcue]**. In addition to their use for cue bars, **[cue]** and **[endcue]** can be used for printing complicated ornaments or optional notes. When cue notes are dotted, the dots are spaced horizontally in proportion to the size of the cue notes. However, when printing optional notes with full-sized notes above or below on the same stave, it is sometimes better to arrange for all the dots to be aligned. You can request this by specifying

```
[cue/dotalign]
```

The space between the cue notes and their dots is increased in this case.

47.24 [Deepbass]

This specifies an F clef based on the fifth stave line. See **[alto]** for further details of clef directives.

47.25 [Dots]

Augmentation dots are normally printed in the space above when a note appears on a stave line. The directive **[dots]** is provided for changing this. It must be followed by one of the words ‘above’ or ‘below’, and it applies to all subsequent notes on the stave, with the exception of certain adjacent notes in chords. Note that the position for an individual note can be overridden by means of a note option consisting of a colon (see section 41.13).

47.26 [Draw]

The **[draw]** directive is described in chapter 37.

47.27 [Endcue]

See **[Cue]** (section 47.23) above.

47.28 [Endline]

See **[line]** (section 47.40) below.

47.29 [Endslur]

See **[slur]** (section 47.75) below.

47.30 [Endstave]

The data for each stave of music must end with the directive **[endstave]**. This can be followed only by the start of a new stave or the start of a new movement or by the end of the file.

47.31 [Ensure]

The **[space]** directive always inserts extra space before a note. Sometimes all that is needed is an assurance that a certain amount of space is available, for example, when using the drawing facilities to print markings that PMW doesn’t know about. The **[ensure]** directive provides this facility. If the requested amount of space is not available between the next note (or the end of the bar) and the previous note (or the start of the bar) on the current stave, a suitable amount of space is inserted. Consider, for example,

```
G [ensure 32] G
```

If this is the only stave, then because minims are normally printed 20 points apart, the **[ensure]** directive has the effect of inserting 12 points of space. However, if there is another stave containing four crotchets, which print 16 points apart, there is already 32 points between the two minims, and no extra space is inserted.

The additional space is inserted immediately before the note, thus moving it further away from any other items, such as clefs, which lie between it and the previous note.

47.32 [Fbfont]

The default typeface for figured bass text that is printed with a stave can be set for an individual stave by means of the **[fbfont]** directive. This directive takes as its parameter one of the standard font names.

For example, supposing that the third extra font had been defined for use in figured bass text, then

```
[fbfont extra 3]
```

would be used at the start of that stave. The default typeface for figured bass text is roman. In any given text string it is always possible to change typeface by using the appropriate escape sequence.

47.33 [Fbtextsize]

This directive must be followed by a number in the range 1 to 12. It selects the default size to be used for figured bass text on the current stave. The actual font sizes that correspond to the twelve available sizes are set by the **textsize** heading directive. If this directive is not used, the size set by the **fbsize** heading directive (which is different from any of the sizes set by **textsize**) is used.

[Fbtextsize] is normally only needed if you want different sizes of figured bass text on different staves.

47.34 [Footnote]

The stave directive **[footnote]** defines a text string that is printed at the foot of the page on which the current bar is printed. Footnotes are different from footings, in that the space in which they are printed is taken from the normal page length; consequently the bottom system of music is printed higher up the page, in order to leave room for footnotes.

The syntax of **[footnote]** is the same as the syntax of the **heading** and **footing** directives, and like them, if the text is longer than the line length, it is automatically split into several lines (see section 38.45 for full details of this). For example:

```
[footnote "A close friend of Schumann and
Mendelssohn, Sheffield-born Sterndale
Bennett founded the Bach choir, was for ten
years the conductor of the Philharmonic
Society, and in 1866 became principal of
the Royal Academy of Music."]
```

The initial font is roman, and the default size is 9 points, but this can be changed by the **footnotesize** heading directive.

If there are several footnotes on one page, vertical white space is left between them. The default amount of space is 4 points, but this can be changed by the **footnotesep** heading directive.

If there are several footnotes in one system, they are ordered by stave, those for the lowest numbered stave being printed first.

47.35 [Hairpins]

Hairpins are normally printed below the stave. The **[hairpins]** directive is provided for changing this. It must be followed by one of the words ‘above’ or ‘below’. It can also be followed by ‘middle’, which causes hairpins to be printed below the stave, half way between it and the following stave (unless low notes on the upper stave force them lower down).

Hairpins in fixed positions above or below the stave can be made the default by following ‘above’ or ‘below’ in the **[hairpins]** directive by a dimension. For example,

```
[hairpins above 10]
```

Individual hairpins can be moved from this position by the normal `/u` and `/d` qualifiers. In addition, `/a` and `/b` can be used without a dimension to specify the default type of hairpin, whose vertical position depends on the notes it covers.

It is also possible to set up a default adjustment for variable-position hairpins, by giving a dimension preceded by `+` or `-` in the **[hairpins]** directive. For example, after

```
[hairpins below -4]
```

has been encountered, all hairpins are positioned as if they were followed by `/d4`.

Note the distinction between

```
[hairpins above 8]
```

and

```
[hairpins above +8]
```

The former causes all hairpins to be printed 8 points above the stave, while the latter adds 8 points to whatever position PMW computes from the notes under the hairpin.

47.36 [Hairpinwidth]

This directive, which must be followed by a dimension, sets the width of the open ends of any subsequent hairpins on the current stave.

47.37 [Hclef]

This directive causes a percussion 'H-clef' to be used on the current stave. This behaves as a treble clef as far as note positioning is concerned. See **[alto]** for further details of clef directives.

47.38 [Justify]

The justification parameters can be changed by the appearance of this stave directive. Unlike the heading directive of the same name, it specifies changes to the justification parameters only, and its effect lasts only until the end of the current movement.

It must be followed by a `+` character (for adding a justification) or a `-` character (for removing a justification) immediately preceding one of the words 'top', 'bottom', 'left', or 'right'. For example, if the last page of a piece uses only slightly more than half of the page depth, and vertical justification is not wanted, then

```
[justify -bottom]
```

should be included in any bar on that page. Changes of parameter take effect from the system in which they are encountered, and persist until a subsequent change. More than one change may be given at once. For example,

```
[justify -right -bottom]
```

might be used in the last system of a piece.

47.39 [Key]

A change of key signature is indicated by the directive

`[key <`*key signature*`>]`

If the change of key falls at the start of a system, a cautionary key signature is printed at the end of the previous line unless the word 'nowarn' is included in the directive, for example:

```
[key E$ nowarn]
```

There is also a heading directive, **nokeywarn**, for suppressing all cautionary key signatures.

Key signature changes are printed in 'modern style'. That is, unless the new key is C major (or A minor), all that is printed is the new signature. If 'old style' is required, where the new key signature is preceded by an explicit cancelling of the old one with naturals, the new signature should be preceded by a change to C major. For example,

```
[key C][key A]
```

prints a number of naturals to cancel the previous signature before printing three sharps.

When a bar starts with a new key signature and a repeat mark, the order in which these are printed depends on the order in which they appear in the input.

```
[key G] (:
```

causes the key signature to be printed first, followed by the repeat mark, while

```
(: [key G]
```

causes the repeat mark to be amalgamated with the previous bar line, with the key signature following. If, at the same point in the music, these items appear in different orders on different staves, then the repeat sign is printed first on all staves.

47.40 [Line]

There are a number of situations in music where it is required to draw a straight line above or below a sequence of notes, with or without small 'jogs' at the ends. With the 'jogs' one has a sort of horizontal bracket.

There is a stave directive **[line]** to do this. It works exactly like **[slur]**, except that what is drawn is a straight line with a vertical 'jog' on each end. There is an **[xline]** directive that corresponds to **[xslur]**.

The end of the line is marked by **[endline]** or **[el]** and there are the same options as for **[slur]** – for example, `/b`, `/u`, `/d`, `/rr` etc. The `/co` and `/ci` options affect the length of the 'jogs'; however, the other options starting with `/c`, which for a slur move the Bezier curve control points, are ignored for lines.

Unlike slurs, these markings are by default always positioned above or below the stave itself, never actually overprinting it. Like slurs, they follow the 'shape' of the notes underneath (or above) to some extent.

Lines can be positioned at fixed positions above or below the staves or at the underlay or overlay levels. The fixed positions refer to the main part of the line, excluding the jogs, if any. See **[slur]** for details of the relevant options.

The **[linegap]** directive can be used to leave gaps in lines and cause drawing or printing to take place in the gap.

The following options can also be given with **[line]** in addition to those available for **[slur]**:

`/ol`	requests that the line be 'open on the left'
`/or`	requests that the line be 'open on the right'

An 'open' line has no 'jog' on the end. There is a directive **[xline]** that works like **[xslur]**, and lines can be 'tagged' in the same way as slurs. Note that the `/i` and `/ip` qualifiers are available, and cause a dashed or dotted line to be drawn, respectively.

47.41 [Linegap]

The directive **[linegap]** requests that a gap be left in a line that was set up by the **[line]** directive. The following options are provided:

- `/=`<*letter*> is used to identify which line is being referred to, in exactly the same way as it is used on the **[endline]** directive.
- `/w` followed by a number is used to specify the width of the gap; if it is not given, a width of four points is used. The width is measured along the line.
- `/h` specifies that the centre of the gap is to be halfway along the line. It can be followed by a number in the range 0–1.0 to specify a different fraction of the length; for example, `/h0.75` specifies that the centre of the gap is to be three-quarters of the way along the line.

  If `/h` is not specified, the centre of the gap is aligned with the centre of the notehead of the next note, or with the barline if there are no more notes in the bar.
- `/l` and `/r` are used to move the position of the gap to the left or to the right by a given number of points.

- `/draw` <*arguments*> <*name*> specifies that the named drawing is to be associated with the gap. When the drawing code is obeyed, the origin of the coordinate system is set at the centre of the gap, and the variables **linegapx** and **linegapy** define the start of the right-hand portion of the line relative to this origin. For example, for a horizontal line, **linegapx** is half the width of the gap, and **linegapy** is zero.
- `/"`<*text*>`"` associates the text with the gap. For example

  ```
  [linegap/"\it\ad lib."]
  ```

  The default font for the text is roman. If no width for the gap is given by the `/w` option, the width is set to the length of the text plus a little bit. The text is printed centred in the gap, rotated so that it has the same slope as the (imaginary) line joining the ends of the gap.

  The text option must be the last option for **[linegap]** because any further options are taken as options that apply to the string. The following text options are available: `/u`, `/d`, `/l`, `/r`, `/s`, `/box`, and `/ring`. The movements are relative to a coordinate system whose 'horizontal' axis lies on the line joining the ends of the gap.

If `/h` is used when a line is split between two systems, it is applied to whichever part of the line the **[linegap]** directive falls in.

If a gap passes either end of the line, the ending jog is never drawn, even if specified. A drawing can be associated with the start or end of a line by using `/h0` or `/h1`, respectively.

To associate a drawing with a particular point on a line, but without leaving a gap, `/w0` can be used.

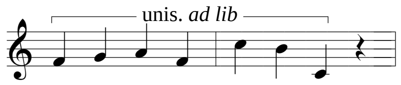
When the `/draw` option is used, take care not to leave out the `/` by mistake. If a space is used instead, the drawing is no longer associated with the line gap, but with the following note.

Any number of gaps may be specified for a single line. They are processed from left to right, and all the drawings for a single line on one system are processed together in succession.

When defining drawings that are to be used with line gaps, it is useful to know that the width of lines drawn by **[line]** is 0.3 points.

Here is an example when a gap is used in order to print some text in the middle of a horizontal line:

```
[line/a/h linegap/h/"unis. \it\ad lib"] fgaf |
c'bc [el] r
```



The `/c` option is used with the string in the drawing, to centre it at the current point, which is specified as slightly below the origin because text strings are printed with their base lines at the current point.

Because the `/h` option is used, the **[linegap]** directive can be given right at the start of the line. Do not confuse `/h` as used with **[line]**, where it means *horizontal*, with `/h` as used with **[linegap]**, where it means *halfway*. This latter usage was chosen to be similar to the `/h` option on hairpins.

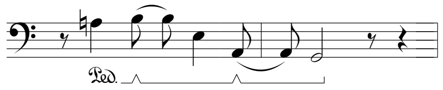
Another use of the **[linegap]** facility is for drawing conventional piano pedal markings. Because these appear often in a piece, it is sensible to define macros for the relevant directives:

```
draw blip
  linegapx linegapy moveto
  0 linegapx 2 mul lineto
  linegapx neg linegapy neg lineto
  0.3 setlinewidth stroke
enddraw

draw ped
  0 0 moveto "\**163\ " show
enddraw

*define ped  [line/=P/b/h/ol/d4 linegap/h0/w30/draw ped]
*define blip [linegap/=P/draw blip]
*define ep   [endline/=P]
```

```
[stave 1 bass 0]
r- &ped %a &blip b-_; b-; e &blip a'-_ |
a'- G' &ep r-r |
```



The **ped** macro starts the line, which is specified as horizontal and open (i.e. no jog) on the left. It is also moved down four points, to allow for the height of the 𝒫𝑒𝒹 text, which is printed at the start by means of the immediately following **[linegap]** directive.

The **blip** macro creates a gap in the line at the next note, and causes a 'blip' to be drawn. Notice that the size of the blip is relative to the width of the gap. Thus the same drawing could be used for different sized blips if required.

The **ep** macro simply ends the line. In simple cases it would be just as quick to type **[el]**, the abbreviation for **[endline]**, but using the macro makes it clear that it is a pedal line that is terminating as well as using the `/=` option to specify exactly which line is being referred to.

47.42 [Mezzo]

This specifies a C clef with its centre on the second stave line. See **[alto]** for further details of clef directives.

47.43 [Midichannel]

The **[midichannel]** directive can be used to change the MIDI channel that the current stave uses, from the next note onwards:

```
[midichannel 5]
```

It can also be used to change the voice allocation of the new channel at the same time:

```
[midichannel 6 "flute"]
```

The voice change takes effect at the time of the next note (or rest), and of course it affects any other staves that may be using the same channel. A relative volume may be given after the voice name:

```
[midichannel 1 "trumpet"/12]
```

See the **midichannel** directive for how to set up channels at the start of a piece.

47.44 [Midipitch]

The **[midipitch]** directive is used to alter the MIDI playing pitch for a stave, for the purpose of selecting a different untuned percussion instrument. Its argument takes the same form as the final argument of the **midichannel** heading directive:

```
[midipitch "bongo"]
```

To stop forcing the playing pitch on a stave, specify an empty string in quotes.

47.45 [Midivoice]

The **[midivoice]** directive can be used to change the MIDI voice without changing the channel:

```
[midivoice "french horn"]
```

Again, it will affect any other staves that are using the same channel.

47.46 [Move]

The directive

```
[move <n>]
```

causes the next non-textual thing that is to be printed on the current stave, whether it be a note or something else, to be moved horizontally by <*n*> points, without affecting the position of anything else in the bar (except slurs or ties that are attached to a moved note). If <*n*> is positive, movement is to the right; if negative, it is to the left. Certain items can also be moved vertically, by specifying

```
[move <n>,<m>]
```

where <*m*> is the vertical movement required. For example, to print a mid-stave clef four points higher than usual one might have

```
[move 0,4][treble 1]
```

The second parameter may be a positive or negative number; positive movement is upwards. If two or more **[move]** directives appear in succession on a stave, they act cumulatively.

Vertical movement does not apply to notes, and is ignored if specified. Other features such as text and slurs have their own syntax for vertical movement. Those items to which the **[move]** directive applies are: clefs, key signatures, time signatures, dotted bar lines, repeat marks, caesuras, commas, ticks, and notes (horizontal movement only).

When staves of different sizes are in use, any vertical movement specified by **[move]** is scaled for the current stave, but horizontal movements are not scaled. However, there is a related directive called **[rmove]** which scales in both directions.

A **[move]** directive can also be present in the first bar of a sequence of rest bars. If they are packed up into a single multi-bar rest, the **[move]** is applied to the number that is printed above the long rest sign.

See also **[ensure]**, **[rmove]**, **[rsmove]**, **[rpace]**, **[smove]** and **[space]**.

47.47 [Name]

The **[name]** directive has two entirely separate functions. If its parameters are one or more strings or calls to drawing functions, it is an alternative way of defining text or drawings to be printed at the left-hand side of the stave. This can be useful when portions of the text are being skipped conditionally. For example,

```
[stave 1 treble 1]
*if score
[name "Flute" "Fl."]
*fi
```

prints the name in the score, but not in the part. In this form, the parameters for **[name]** are exactly the same as the text and drawing parameters of the **[stave]** directive.

Sometimes it is necessary to change what is printed at the start of a stave part-way through a piece, for example, if a double choir becomes a single choir, or *vice versa*, and this is what the second form of **[name]** is used for. In this usage, its parameter is a single number, which selects which string and/or drawing is to be used on the current and subsequent staves.

Each ‘item’ for printing at the start of a stave consists of a string or a call to a drawing function, or both, and they are numbered starting at one. For example, if the stave starts with

```
[stave 2 "Alto" "A" "Alto I"]
```

Then by default in the first system the stave is labelled ‘Alto’, and in all subsequent systems it is labelled ‘A’. However, if

```
[name 3]
```

is used at any point, then the system in which it appears and any subsequent ones will use the text ‘Alto I’ for this stave. A number greater than the number of items can be used to suppress printing of anything at all; alternatively, empty strings can be used.

47.48 [Newline]

The directive **[newline]** can be used in stave data to force a new line (system) of music to be started at a particular point. It need appear in one stave only. If a stave is not selected for printing, however, an appearance of **[newline]** within it is ignored. An occurrence of **[newline]** should always be at the start of a bar.

47.49 [Newmovement]

This directive must always appear immediately after an **[endstave]** directive. It signals the start of a new section of music, and is followed by an optional set of new heading directives, and then more staves of data.

When **[newmovement]** is used without a parameter, PMW looks to see if it can fit the new heading lines (if any) and the first system of the new movement onto the current page. If it cannot, a new page is started. (In the case of a system consisting of one stave only, PMW tries to fit on two systems, because a single line of music at the bottom of a page doesn't look good.)

By specifying **[newmovement newpage]**, you can force PMW always to start a new page. You can also specify **[newmovement thispage]** to stay on the current page when only a single stave of music will fit.

When a new movement starts at the top of a page, any page headings that are in force are printed, in addition to the headings for the movement. This means that, for example, if page numbers are specified in the first movement by a page heading, they will be printed on all subsequent pages. Any page headings specified in the new movement completely replace those set up in the previous movement. Chapter 24 contains more details about the interaction of headings and footings with new movements.

Sometimes, however, it is required to suppress page headings at the start of a new movement, for example if they are being used to print the name of the movement at the head of pages. This can be done by adding the keyword 'nopageheading' to the **[newmovement]** directive. For example:

```
[newmovement nopageheading]
```

This option can be used with or without the 'newpage' option; it takes effect only if the new movement actually starts at the top of a page.

The **lastfooting** directive sets up footings for the final page of a piece. In some circumstances it may be desirable to print a special footing at the end of an individual movement, and this can be done with

```
[newmovement uselastfooting]
```

In this case, if the new movement starts on a new page, the footing on the previous page is the **lastfooting** from the previous movement, if present. To force a new page and cause the **lastfooting** text to be printed, use

```
[newmovement newpage uselastfooting]
```

Use of this option does not cancel the **lastfooting** text; it is carried forward to the new movement, but of course can be cancelled by a new **lastfooting** directive in the new movement.

Another possible form of the directive is **[newmovement thisline]**. This is for somewhat specialized circumstances. It has the effect of starting a new movement without advancing the current vertical position on the page. If there are no headings, the first system of the new movement prints with its first stave at the same level as the first stave of the last system of the previous movement.

Two different uses are envisaged for this:

- Music for church services often contains very short sections of one or two bars, and it is sometimes desirable to print two of them side by side.
- One style of printing incipits has white space between the incipit staves and the start of the main system.

In both cases it is necessary to specify left justification for the last system of the first movement, and right justification for the first system of the second.

47.50 [Newpage]

The directive **[newpage]** can be used in stave data to force a new page of music to be started at a particular point. It need appear in one stave only. If a stave is not selected for printing, however, an appearance of **[newpage]** within it is ignored. An occurrence of **[newpage]** should always be at the start of a bar.

47.51 [Nocheck]

It is sometimes useful to disable PMW’s checking of bar lengths (at the start or end of a piece, for example). The directive **[nocheck]** suppresses this check, for the current bar only. (See also the heading directive of the same name for suppressing the check globally.) It is permitted to omit **[nocheck]** from bars in which absolutely nothing appears other than a whole bar rest indication.

47.52 [Noclef]

This specifies an invisible clef. It acts as a treble clef as far as note pitch is concerned. It is useful when setting incipits where no clef is required. It is also useful when setting fragments and musical examples.

47.53 [Nocount]

In certain circumstances it may be necessary to prevent a bar in the middle of a piece from being counted for the purposes of bar numbering, for example, when using an ‘invisible bar line’ to make PMW split a bar over two lines on the page. Also, if the first bar of a piece is incomplete, it is conventional not to include it in the bar numbering. The directive **[nocount]** causes the current bar not to be counted; such a bar never has its number printed. It need only appear in one stave. If it appears in a bar whose contents are repeated by means of a number in square brackets, all the repeated bars are uncounted.

47.54 [Noteheads]

Three alternative note head shapes are supported: diamond-shaped for string harmonics, cross-shaped, and invisible (i.e. no noteheads at all). It is also possible to print noteheads without stems.

The character ✓ (which is called ‘direct’) is sometimes seen on a stave in musical extracts (and examination papers) to indicate a pitch without specifying a time value. This character is in PMW’s font and can be positioned as a text item, but it is also available as an exotic form of notehead.

The stave directive **[noteheads]** is used to control these features. It must be followed by one of the words ‘normal’, ‘harmonic’, ‘cross’, ‘none’, ‘direct’, or ‘only’. For example,

```
a b [noteheads cross] c d | [noteheads normal] e f
```

prints the second two notes in the first bar with cross-shaped noteheads.

Because this is quite a long directive to type, and one which might appear frequently in some music, abbreviations are provided as follows:

`[o]`	is equivalent to	`[noteheads normal]`
`[h]`	is equivalent to	`[noteheads harmonic]`
`[x]`	is equivalent to	`[noteheads cross]`
`[z]`	is equivalent to	`[noteheads none]`

When no note heads are being printed, ledger lines are omitted, and breves and semibreves are totally invisible. With cross-shaped note heads there is no difference between the appearance of a crotchet and a minim. Breves are distinguished from semibreves only when normal note heads are being printed.

When `[noteheads direct]` is selected (to print notes as ✓), ledger lines are printed as normal, but no stems or beams are printed.

For printing stemless notes, the directive

```
[noteheads only]
```

requests that all stems and beams be suppressed until another occurrence of **[noteheads]** with a parameter other than ‘only’.

47.55 [Notes]

The directive

```
[notes off]
```

suppresses the printing of notes and their stems on the stave. However, if the notes would have been beamed, the beams are still drawn. This can be used for placing beams in non-standard places. In addition, if the ornaments or fermatas are specified, these are also printed.

Making the note or chord at the end of a tie invisible is a convenient way to print 'hanging' tie marks. The effect of **[notes off]** can be reversed by **[notes on]**.

47.56 [Notespacing]

The **[notespacing]** directive may appear as a directive in stave data, to specify a temporary change in the horizontal distances between notes. If the directive is given with no parameters, it resets to the values that were current at the start of the movement. **[Ns]** is an abbreviation for **[notespacing]**.

The most commonly used form of the **[notespacing]** directive is the one that changes each element in the note spacing table by a multiplicative factor. This is done by following the keyword by an asterisk and a number (possibly containing a decimal point) or a rational number, as in the following examples:

`[ns *0.75]`	change to three-quarter spacing
`[ns *3/4]`	change to three-quarter spacing
`[ns *2]`	double the spacing
`[ns *1.5]`	multiply the spacing by one and a half
`[ns *3/2]`	multiply the spacing by one and a half

Internally, note spacings are held to an accuracy of 0.001 of a point.

Alternatively, the directive name can be followed (in the brackets) by up to eight numbers, which give the *change* in the note spacing, in points, for notes of different lengths, starting with the value for breves. (See the **notespacing** heading directive.) Trailing zero values can be omitted.

```
[notespacing 0 0 3 -2]
```

specifies that minims are to be printed three points further apart, while crotchets are to be two points closer together.

The **[notespacing]** directive takes effect for the remainder of the current bar on the stave where it occurs, and for all the notes in the same bar on staves of higher number (that is, those that print below it on the page), and then for all notes in subsequent bars. Of course, this may have the effect of moving notes in previous staves, in order to keep the music properly aligned.

Warning: To avoid unexpected effects, **[notespacing]** is best used only at the beginnings of bars, and preferably in the top part. When changing the spacing for a single bar, it is all too easy to reset the note spacing within the bar, for example,

```
[notespacing *0.8] a-b- g d [ns] |
```

This may behave strangely because PMW processes bars stave by stave. It will therefore obey the resetting directive before considering the other staves, and only the one stave will have been processed with the altered spacing. The correct form is

```
[notespacing *0.8] a-b- g d | [ns]
```

If a change of note spacing is always required, whatever combination of staves is selected for printing, then it can be given on stave 0.

47.57 [Octave]

It is often useful to input music at a different octave from that at which it is to be printed. For example, when a part is in the treble clef it is easier to enter if the letters `c–b` represent the octave starting at middle C rather than the one below it. The directive

`[octave <`*number*`>]`

requests transposition by the number of octaves given. The octave can also be set at the same time as the clef (see the **[alto]** directive for details). Each such octave setting replaces the previous one; they are not cumulative. If the number is positive, transposition is upwards; if negative, it is downwards. Octave transposition is in addition to any general transposition which is in force.

47.58 [Olevel] and [olhere]

These directives control the position of the overlay level in exactly the same way as [ulevel] and [ulhere] do for the underlay level. See section 47.103 for details.

47.59 [Oltextsize]

This directive must be followed by a number in the range 1 to 12. It selects the default size to be used for overlay text on the current stave. The actual font sizes that correspond to the twelve available sizes are set by the **textsize** heading directive. If this directive is not used, the size set by the **overlaysize** heading directive (which is different from any of the sizes set by **textsize**) is used.

[Oltextsize] is normally only needed if you want different sizes of overlay text on different staves.

47.60 [Omitempty]

When a stave is about to be suspended, it is sometimes desirable not to print stave lines after the final bar which contains notes, and similarly, when a stave is resumed, empty bars preceding the resuming bar may not be required.

If the directive **[omitempty]** appears at the start of a stave's data, then *nothing at all* is ever printed for bars for which no data is supplied. Such bars can be set up by means of the **[skip]** stave directive, or simply by omitting them at the end of a stave's data. Note that a bar which is specified as a rest bar, visible or invisible, counts as a bar for which there is data, and a clef specification also counts as data. Therefore, if bars are to be omitted at the start of a stave, the input should be as in this example:

```
[stave 3 omitempty skip 20]
```

with the clef specification (possibly made invisible by means of **[assume]**) following.

It is not necessary for the suspend mechanism to be used with this feature, though if is is not, vertical white space will always be left for the stave, even if nothing is printed in that space.

When a non-empty bar follows an empty bar in a stave for which **[omitempty]** has been set, and it is not the first bar in a system, a bar line has to be printed at its start. By default, a conventional solid bar line is printed, but it is possible to specify other bar line styles, a double bar line, or an invisible bar line, by using the normal PMW notations for these things at the end of the preceding empty bar. For example,

```
[omitempty] gg | [skip 10] |? aa |
```

specifies that no bar line is to be printed at the start of the final bar if it falls in the middle of a system. Note that because of the way **[skip]** works, this example contains 11 empty bars, not 10.

One or more **[omitempty]** staves can be used for printing isolated bars on a page, using empty bars between them to cause horizontal white spaces to appear. The size of the white spaces can be controlled by the use of **[space]** directives on stave 0 – they can't be used in the empty bars, as that causes PMW to treat them as not empty.

47.61 [Overdraw]

When a drawing is associated with a note or bar line by means of the **[draw]** command, the drawing output happens before the note or bar line is output. The order does not matter when everything is black, but if the **setgray** drawing operator is being used, the drawing may need to be done last to achieve the correct effect.

[Overdraw] acts just like **[draw]** except that it saves up the drawn items, and outputs them only after everything else in the system has been output. Using **setgray** and **[overdraw]** it is possible to 'white out' parts of staves.

47.62 [Overlayfont]

The default typeface for overlay text that is printed with a stave can be set for an individual stave by means of the **[overlayfont]** directive. This directive takes as its parameter one of the standard font names. For example:

```
[overlayfont italic]
```

The default typeface for overlay text is roman. In any given text string it is always possible to change typeface by using the appropriate escape sequence.

47.63 [Page]

Occasionally there is a requirement to skip a page in the middle of a piece – to insert commentary in a critical edition, for example. The **[page]** stave directive can be used to set the page number for the page on which it appears, but it is not possible to decrease the page number. The **[page]** directive can specify an absolute page number, or an increment of the page number preceded by a plus sign, for example:

```
[page 45]
[page +1]
```

47.64 [Percussion]

*The **[percussion]** directive is now deprecated, having been superseded by the **[stavelines]** directive, which should be used instead in all new input files.*

The **[percussion]** directive, which has no parameters, specifies that the current stave is for untuned percussion. It has the following effects:

- The stave is printed as a single line instead of five. The line is positioned where the middle line of a five-line stave would be.
- No clef or key signature is printed at the start of the stave.
- Whole bar rests are printed under the third line (i.e. under the line which is printed) instead of under the (invisible) fourth line of the stave.
- No ledger lines are printed for notes off the stave.

Otherwise the stave behaves as normal. Ordinary note heads are printed. Although no clef is printed, the vertical positioning of notes is relative to the current clef. For example,

```
[stave 5 "Side Drum" "S.D." bass 0]
[10] d d-d- | [20]R! |
```

The use of the bass clef ensures that the note `d` prints on the middle line of the stave, that is, the single line of the percussion stave.

47.65 [Playtranspose]

The **[playtranspose]** directive can be used to change the playing transposition of a stave, which is initially set from the **playtranspose** heading directive. The value given in **[playtranspose]** is added to the current playing transposition for the stave at the point it is encountered. One use of this is to arrange for *8va* passages to be played (via MIDI) at the correct pitch. Changes made by **[playtranspose]** do not persist beyond the end of the movement.

This directive can be used to change the relative MIDI playing volume of a particular stave part way through. Its single argument is number between 0 and 15, specifying the new relative volume for the current stave.

47.66 [Printpitch]

When inputting a file that is both to be printed and played on a MIDI instrument, the **[midipitch]** directive can be quite cumbersome to use if a percussion part changes instruments frequently, even though the amount of typing can be reduced by using macros. An alternative facility which forces the printing pitch instead of the playing pitch is therefore provided.

The **[printpitch]** directive takes a note letter and optional octave indication as its argument. It causes all subsequent notes on the stave to be printed on the corresponding line or space, whatever pitch is specified for the note in the input. The input pitch can then be used to select different percussion instruments for MIDI output. To do this, you need to know that middle C corresponds to MIDI note 60, C-sharp is 61, and so on upwards, while B is 59 and so on downwards.

Of course, some indication in the printed music is also required to tell a human player what to do – this can take the form of graphic signs above the notes, or different noteheads or stem directions can be used.

Here is an invented example, where the first three beats of the bar are played on a snare drum (General MIDI pitch 38), with the last beat on the cowbell (General MIDI pitch 56), indicated by a downward pointing stem.

```
[stave 8 hclef 0 stavelines 1]
[printpitch b' stems up]
d'd'd' $a-\sd\$a-  |
```

The effect of **[printpitch]** can be cancelled by supplying an asterisk as its parameter.

When a percussion stave with more than one line is used to separate different instruments, or if notes are placed above and below the line, it is probably easiest to input each instrument's part on a separate PMW stave, and arrange for them to overprint each other. Then the appropriate MIDI sound can be permanently set for each stave.

47.67 [Reset]

Sometimes it is convenient to notate a bar as two different sequences of notes, to be overprinted on the stave. The stave directive **[reset]** has the effect of resetting the horizontal position to the start of the current bar. Anything that follows it is overprinted on top of whatever has already been specified. For example,

```
[stems up] gabc' [reset] [stems down] efga |
```

prints a bar containing the chords (eg), (fa), (gb) and (ac'), but with the stems of each component of the chord drawn in opposite directions.

More than one **[reset]** may appear if necessary, and only one set of notes need be of the correct length to satisfy the time signature. The facility for printing invisible rests, notated by the letter Q, can be useful in conjunction with **[reset]**.

If a large number of bars require overprinting, it may be more convenient to set up an entire overprinting stave by specifying a stave spacing of zero. **[Reset]** should in any case be used with caution, because it can cause unexpected effects if items such as slurs are in use.

Because PMW processes bars from left to right, **[reset]** must not appear between two notes which are connected in some way, for example, between two tied or slurred notes. It must also not appear between any other printing item and the note or bar line which follows, since such items are always 'attached' to the following note or bar line.

Specifically, **[reset]** must not follow any of the following: a clef, a tied note, the first note of a glissando, the start of a hairpin, a mid-bar dotted line, a repeat sign, a caesura, a text item, **[slur]**, **[xslur]**, **[line]**, **[xline]**, **[key]**, **[time]**, **[comma]**, **[tick]**, **[move]**, **[smove]**, **[space]**, or a rehearsal marking. Also, **[reset]** may not occur in the middle of an irregular note group.

47.68 [Resume]

This directive forces a resumption of a suspended stave – see **[suspend]** for details.

47.69 [Rlevel]

Rests are normally printed centrally on the stave, as is conventional for single parts. When two staves are being overprinted to combine two different parts, it may be necessary to move rests up or down. There is a note option that can be used to do this for individual rests (see section 41.13 above). The directive

```
[rlevel <n>]
```

specifies an adjustment that applies to all subsequent rests. Any adjustment specified for individual rests is added to the current rest level as set by this directive.

The parameter for **[rlevel]** may be positive or negative; it specifies a number of points by which the rest is moved vertically. A positive number moves upwards, and a negative one moves downwards. For example,

```
[rlevel -12]
```

causes rests to be printed 12 points lower than normal, so that a whole bar rest, which normally prints below the fourth line, now prints below the bottom line of the stave.

Semibreve and minim rests that are moved off the stave are printed with a single ledger line to indicate which they are.

Each occurrence of **[rlevel]** sets a level relative to the default position. They are not cumulative.

47.70 [Rmove]

This directive operates exactly as **[move]**, except that horizontal movements are scaled to the relative stave size.

47.71 [Rsmove]

This directive operates exactly as **[smove]**, except that horizontal movements are scaled to the relative stave size.

47.72 [Rspace]

This directive operates exactly as **[space]**, except that horizontal dimensions are scaled to the relative stave size.

47.73 [Sghere] and [sgnext]

[Sghere] and **[sgnext]** affect the system gap value, that is, the amount of vertical space that is left between systems. (When vertical justification is enabled, this value is the minimum amount of space.)

Sghere changes the spacing for the current system only (that is, the one in which the current bar is to appear), while **sgnext** makes the change for all systems that follow the current one.

In each case a single number is required as an parameter. It can be preceded by a plus or minus sign to indicate a relative change from the existing value. Note that when a single part is being printed, it is the system gap that determines the distance between staves.

If more than one occurrence of **[sghere]** is encountered in a single system, the largest spacing value is used. In the case of multiple occurrences of **[sgnext]**, the last value is used (for the next system).

47.74 [Skip]

When setting keyboard music it is common to use two overprinting staves for notes with stems in different directions. Frequently, though, there are long sequences of bars for which the second stave is not required.

Such a sequence can be notated using invisible whole bar rests, but if this is done it is still necessary to keep the clef and key signature in step with the other stave so that they are printed correctly at the beginnings of lines, and at least the final time signature change must appear in the correct place so that it is available for checking when notes resume.

An alternative is to use the **[skip]** stave directive, which should appear at the beginning of a bar, and which causes PMW to leave a given number of bars totally empty. For example,

```
  [stave 2] gg | [skip 50] aa | [endstave]
```

defines a stave in which only bars 1 and 52 are defined. When a totally empty bar occurs at the start of a system, the clef and key signature are not printed. Otherwise such bars are treated as if they contained invisible whole bar rests.

If **[skip]** is used at the very start of a stave, then no clef directive should be given, as otherwise the clef directive is taken as part of the resumed bar after the skip.

See also the **[assume]** directive.

47.75 [Slur]

Slurs between adjacent single notes can be input simply by inserting an underline character after the first note, as described in section 41.20 above. When a slur spans several notes, it must be coded using the **[slur]** and **[endslur]** directives. **[Es]** is an abbreviation for **[endslur]**.

The options for the **[slur]** directive are also applicable to the **[line]** directive, which is described in section 47.40 above.

47.75.1 Normal slurs

The **[slur]** and **[endslur]** directives enclose the notes which are to be slurred. For example:

```
  a [slur] b-a-g-f- [endslur] g
```

causes a slur to be drawn over the four beamed quavers. Slurs are drawn above the notes by default. The shape of slurs is correct in many common cases, but when there is a large variation in pitch in the notes being slurred, the slur mark may sometimes need adjustment.

To allow the user to adjust the printing of slurs, various options are provided for the **[slur]** directive. The options are separated from each other, and from the directive name, by slashes. The following are available:

`/a`	slur above the notes (default)
`/a`<*n*>	slur above, at fixed position above stave
`/ao`	slur above, at overlay level
`/b`	slur below the notes
`/b`<*n*>	slur below, at fixed position below stave
`/bu`	slur below, at underlay level
`/h`	force horizontal slur
`/ll`<*n*>	move the left end left by <*n*> points
`/lr`<*n*>	move the left end right by <*n*> points
`/rl`<*n*>	move the right end left by <*n*> points
`/rr`<*n*>	move the right end right by <*n*> points
`/u`<*n*>	raise the entire slur by <*n*> points
`/d`<*n*>	lower the entire slur by <*n*> points
`/lu`<*n*>	raise the left end by <*n*> points
`/ld`<*n*>	lower the left end by <*n*> points
`/ru`<*n*>	raise the right end by <*n*> points
`/rd`<*n*>	lower the right end by <*n*> points
`/ci`<*n*>	move the centre in by <*n*> points
`/co`<*n*>	move the centre out by <*n*> points

The options `/u` and `/d` are simply shorthand for specifying an identical vertical adjustment of both ends of the slur. Specifying `/ci` causes the slur to become flatter, while specifying `/co` causes it to become more curved.

The `/h` qualifier requests a horizontal slur, that is, one in which both ends are at the same horizontal level before any explicit adjustments are applied. This is implemented by forcing the right-hand end to be at the same level as the left-hand end.

Use of the `/a` or `/b` options with a fixed position (e.g. `/a8`), or use of the `/ao` or `/bu` options initializes the vertical positions of both end points, resulting in a horizontal slur by default. However, the normal options for moving the ends can also be applied. Since the total combination of options specifies the positions exactly, the `/h` option is not relevant, and is ignored if given.

The `/ao` and `/bu` options are probably more useful with **[line]** than with **[slur]**, for cases when several lines at the same level are required on a single system. For example, if lines are being drawn for piano pedal markings (see **[linegap]** for an example), using the `/bu` option causes them all to be at the same level below a given stave, and to be positioned just below the lowest note on that stave.

If there is overlay or underlay text for a stave, the overlay or underlay level is computed by taking into account only those notes which actually have associated text or dashes or extender lines. If not, all the notes on the stave are taken into account.

Repeated movement qualifiers are accumulated; the following two examples are equivalent:

```
  [slur/a/u4/ld2]
  [slur/a/lu2/ru4]
```

Here are some examples of the **[slur]** directive:

```
  [slur]
  [slur/b]
  [slur/u4]
  [slur/lu2/co4]
  [slur/rr6]
```

One particular use of the options for moving the ends of slurs horizontally is for printing a slur (or tie) that extends from the last note of a bar up to the bar line and no further, or from the bar line to the first note in a bar. These are needed for some kinds of first and second time bar. A slur that includes only one note, such as

```
[slur] a [endslur]
```

provokes a PMW error, since it represents an attempt to draw a slur of zero horizontal extent. However, input such as

```
[slur/rr15] a [endslur]
```

is acceptable. The slur starts at the note, and extends for 15 points to the right.

Slurs may be nested to any depth; for example,

```
a b [slur] c d | [slur/b] e f g [es] a |
f e [es] d c |
```

This would print as a long slur extending from the middle of the first bar to the middle of the third bar, with a shorter slur below three notes in the second bar. In other words, the first **[slur]** matches with the last **[es]**. A similar example, together with its output, is shown in chapter 11.

47.75.2 Additional control of slur shapes

Slurs are drawn using Bezier curves. They are defined by two end points and two control points. The control points are normally positioned symmetrically, giving rise to a symmetric curve. The `/co` and `/ci` ('curve out' and 'curve in') options described above are used to move the control points further from or nearer to the line between the endpoints, respectively.

Occasionally, non-symmetric slurs are needed, and so some additional options are provided to enable the positions of the two control points to be separately moved. These options are:

```
/clu<n>   move left control point up <n> points
/cld<n>   move left control point down <n> points
/cll<n>   move left control point left <n> points
/clr<n>   move left control point right <n> points
/cru<n>   move right control point up <n> points
/crd<n>   move right control point down <n> points
/crl<n>   move right control point left <n> points
/crr<n>   move right control point right <n> points
```

Thus, for example,

```
[slur/a/clu40]
```

draws a slur that bulges upwards somewhat on the left. Experimentation is usually needed to find out the precise values needed for a given shape.

The directions of movement for these options are not the normal ones, except when a slur is horizontal. When a slur's end points are not at the same level, the coordinate system is rotated so that the new 'horizontal' is the line joining the end points. In most cases this rotation is small, and so the difference is not great. In all cases, the left control point relates to the left-hand end of the slur, and the right control point relates to the right-hand end, whichever way up the slur is drawn.

47.75.3 Editorial and dashed slurs

Three alternative forms of slur are provided: dashed slurs, dotted slurs, and 'editorial' slurs. The latter have a short vertical stroke through their midpoint if they are symmetric in shape, or near the midpoint otherwise. The alternatives are specified by qualifiers on the directive:

`/i`	draw an 'intermittent' (i.e. dashed) slur
`/ip`	draw an 'intermittent points' (i.e. dotted) slur
`/e`	draw an editorial slur

These qualifiers can be freely mixed with the other slur qualifiers. If a slur is dashed or dotted, and also marked 'editorial', no attempt is made to ensure that the editorial mark coincides with a solid bit of slur.

47.75.4 Wiggly slurs

The option `/w` causes the curvature of the slur to change sides in the middle. Thus a wiggly slur below some notes starts curving downwards, but then changes to curving upwards. The slur may be solid, dashed, dotted, or editorial. If a wiggly slur crosses the end of a system, the portion on the first system curves one way, and the portion on the next system curves the other way.

47.75.5 Split slurs

Slurs are correctly continued if they span a boundary between two systems on the page. By default, such slurs are not continued over warning key or time signatures at the ends of lines, but PMW can be requested to do this by means of the **sluroverwarnings** heading directive.

The shape and positioning of the end of the first part of a split slur are controlled by the **endlineslurstyle** and **endlinesluradjust** directives.

The sections of a slur that extends over one or more line ends are numbered from 1. An option in a **[slur]** directive that consists just of a number means that subsequent options apply only to the given section. Thus, for example,

```
  [slur/3/lu4/co4]
```

moves the left-hand end of the third section upwards, and increases its curvature. Spaces are allowed between options, and these can be used to make a complicated slur more readable by separating the various sections. For example:

```
  [slur /1/co2 /2/lu4/rd6]
```

The only options that may appear after a section selector are those that move endpoints or control curvature, that is, `/u`, `/d` and those options beginning with `/l`, `/r`, and `/c`.

If a section number is given that is greater than the number of sections, its data is ignored, and when a slur is not split, all section-specific options are ignored.

Movement and curvature options that appear before the first section selector are handled as follows:

- All options beginning with `/c` apply only when the slur is not split.
- `/u` and `/d` always apply to all endpoints of all sections, whether the slur is split or not.
- Options beginning with `/l` (the letter) apply to the starting point of the slur, whether or not it it split. To move the starting point only when the slur is split (but not if it is not) these options can be given after `/1` (the digit), in which case they are added to any values given before the selector.
- Any vertical movement specified with `/lu` or `/ld` is also applied to the right-hand end of the first section of a split slur. To affect only the left-hand end, put these options after `/1` (the digit).
- Options beginning with `/r` apply to the final endpoint of the slur, whether or not it is split. To move the endpoint only when the slur is split (but not if it is not) these options can be given after `/`<*n*>, where <*n*> is the number of the final section, in which case they are added to any values given before the selector.
- Any vertical movement specified with `/ru` or `/rd` is also applied to the left-hand end of the final section of the slur. To affect only the right-hand end, put these options after `/`<*n*>.

If `/ao` or `/bu` is specified for a slur which is split, each section of the slur is positioned at the overlay or underlay level for its own stave, but can of course be moved by suitable options after a section selector. Similarly, `/a` and `/b`, if given with a dimension, cause all sections of a split slur to be positioned at the given vertical position.

If a wiggly slur is split, the first section curves one way, and all subsequent ones curve the other way.

Earlier versions of PMW used a more restricted set of options starting with `/s` to control split slurs. These are still supported, but are no longer documented and should not be used in new files.

47.75.6 Overlapping nested slurs

Normally, in music, slurs are properly nested, that is, if a second slur starts within a slur, the second slur ends before the outer slur. The slur notation in PMW is naturally nested, and automatically ensures that this convention is followed. Very occasionally, however, it is useful to be able to start a second slur within a slur and have it cross over the outer slur. More commonly, it is sometimes necessary to have one slur ending and the next beginning on the same note – a situation that is not possible using the normal PMW slur notation, because slur starts are notated before notes and slur ends afterwards.

It is possible to have the innermost nested slur crossing over the one immediately outside it, by using the special directive **[xslur]** (‘crossing slur’). Thus, for example, the input

```
[slur] a [xslur] b [es] c [es]
```

draws one slur covering the first two notes, and the next slur covering the second two notes. (The first **[es]** is made to refer to the **[slur]** directive and the second to the **[xslur]** instead of the other way round.) This facility is available for the innermost nested slur only.

47.75.7 Tagged slurs

In very complicated music, even the **[xslur]** facility is not powerful enough to describe what is wanted, and it is necessary to use ‘tagged’ slurs.

Any number of slurs may be started at any one time on a stave. The data for a given slur (starting coordinates, etc.) are placed on a ‘stack’ when the **[slur]** directive is obeyed. If another slur is started before the first one is complete, its data goes on top of the stack, temporarily ‘hiding’ any previous data that may be already there.

When a simple **[endslur]** directive is obeyed, it terminates the slur whose data is on the top of the stack (and that data is removed). This is why simple use of **[slur]** and **[endslur]** results in ‘nested’ slurs.

The **[xslur]** directive does *not* place its data on the top of the stack (unless the stack is empty). Instead, it places it one position down in the stack. Thus the next **[endslur]** terminates the previously started slur, leaving the latest one still incomplete.

The qualifier `/=` can be used within a **[slur]** directive to ‘tag’ a slur. It must be followed by a single identifying character. It is recommended that capital letters normally be used, as they are visually distinctive. A tagged slur is placed on top of the stack as normal. The **[endslur]** directive may also contain a tag, using the same syntax. When a tagged **[endslur]** directive is obeyed, the stack of unterminated slurs is searched for a slur with a matching tag, and if one is found, it is that slur which is terminated. If no matching slur is found, an error message is given and the slur on the top of the stack is terminated.

When **[endslur]** does not contain a tag, the topmost slur is terminated, whether or not it is tagged. Here is an example of the use of tagged slurs:

```
[slur/=A] [slur/b/=Z] ggg [slur/=B]
a [es/=A] a [es/=Z] a [es] |
```

47.76 [Slurgap]

The **[slurgap]** directive has the same options as **[linegap]**, and can be used to leave gaps in slurs where they would otherwise cross over other items. For example, to avoid drawing a slur through a key signature:

```
r [slur] G‘+ [slurgap/w30/r10] |
[key e$] c G’ [es] |
```

Specifying a gap associated with a text string or a drawing function provides a way of adding arbitrary annotation to a slur – a width of zero can be given if no actual gap in the slur is required. When an associated drawing function is obeyed, the origin is halfway along the straight line joining the edges of the gap, and the **linegapx** and **linegapy** variables are set as for **[linegap]**.

Bracketed slurs can be done using a drawing function, but the text option is probably easier:

```
[slur slurgap/h0/w0/"(  " slurgap/h1/w0/"   )"]
```

String options can be used to alter the size or position of the text as required.

A gap specified for a dashed slur is liable to result in partial dashes being drawn, unless its length is carefully adjusted.

47.77 [Smove]

This directive is a shorthand for combining a **[move]** and a **[space]** directive. The following two lines of input are equivalent:

```
[move  6] a [space 6]
[smove 6] a
```

This is common usage when adjusting the position of notes on overprinting staves. The space is not scaled by the stave size – use **[rsmove]** if you want scaled space.

47.78 [Soprabass]

This specifies a bass clef with a little ‘8’ printed above it. See **[alto]** for further details of clef directives.

47.79 [Soprano]

This specifies a C clef with its centre on the bottom stave line. See **[alto]** for further details of clef directives.

47.80 [Space]

The directive

```
[space <n>]
```

causes <*n*> points of space to be inserted before the next note in the bar, or before the bar line if there are no more notes. The remainder of the bar, including appropriate items on other staves, is adjusted accordingly. The number <*n*> can be positive or negative; a negative value removes space from the bar. The space is not scaled by the relative stave size. If you want to insert scaled space, use **[rspace]**.

Note that, unlike **[move]**, **[space]** always affects the position of the next *note* (or bar line if there are no more notes in the bar), even if some other item intervenes. Items other than notes are always printed in relation to the note (or bar line) which follows them. Therefore, adjusting the position of a note with **[space]** affects these items too. The following two examples have exactly the same effect:

```
[comma] [space 6] A
[space 6] [comma] A
```

because **[space]** does not affect non-note items such as commas. The **[move]** directive can be used in conjunction with **[space]** to insert space between a non-note item and the note to which it is related, for example

```
[space 6][move -6][comma] A
```

[Space] is obeyed when PMW is figuring out where to position the notes in the bar, while **[move]** is obeyed at the moment of printing.

When there are two or more occurrences of **[space]** at the same position in a bar, PMW takes the largest if previous ones specified a positive amount of space, and the smallest if they specified a negative amount. This normally gives the right effect if extra space is accidentally specified in two different staves.

See also **[ensure]**, **[move]**, **[smove]**, **[rspace]**, **[rmove]**, and **[rsmove]**.

47.81 [Sshere] and [ssnext]

Sshere and **ssnext** affect stave spacing. **Sshere** changes the spacing for the current system only (that is, the one in which the current bar is to appear), while **ssnext** makes the change for all systems that follow the current one.

If either of these directives is followed by a single number, this applies to the current stave only, except when the current stave is number zero, in which case the value applies to all staves (as it does for the **[stavespacing]** heading directive). They can also be followed by two numbers separated by a slash, in which case the first is a stave number and the second is a spacing. For example, a bar with very low notes might require notating thus:

```
[treble 1]
[sshere 60] f` a` c e |
```

This has the effect of setting the stave spacing to 60 points, for the current stave in the current system only.

An alternative format is available for specifying a change to the existing spacing. This is notated by preceding a single number by a plus or minus sign. For example,

```
[sshere +10]
```

adds 10 points to the stave spacing for the current stave only. If more than one occurrence of **[sshere]** is encountered in a single system, the largest spacing value is used. In the case of multiple occurrences of **[ssnext]**, the last value is used (for the next system). When **[ssnext]** is used with a plus or minus sign, the value is relative to the original spacing for the current stave, ignoring any changes that might have been made with **[sshere]**.

47.82 [Stave]

The first directive in each stave's data must be the **[stave]** directive. The most basic form is just

```
[stave <n>]
```

where <*n*> is the number of the stave whose data follows (terminated by **[endstave]**). The number must always be present. Further, optional, parameters may be given to specify text or drawings to be output at the start of the stave. Most commonly, **[stave]** is used just with text, and this form is described first.

47.82.1 Text at stave starts

The text-only form of **[stave]** has the following format:

```
[stave <n> "<string1>" "<string2>" ...]
```

There may be any number of string parameters. The first one is used for the very first stave of the piece, and the second one for all other occurrences of this stave, by default. These strings are normally used for the name of the instrument or voice for which the stave is intended. For example,

```
[stave 1 "Soprano" "S"]
```

If there is only one string, only the first stave will have text printed next to it. If a vertical bar appears in one of these strings, it specifies the start of a new line of text, for example,

```
[stave 5 "Trumpet|in G"]
```

The options `/c` and `/e` can be used to cause the text to be printed horizontally centred or right-justified, respectively. If both `/c` and `/e` are given, and the text consists of multiple lines, (delimited with `|` characters) then the longest line is right justified, and all the other lines for the stave have their centres aligned with the centre of the longest line.

Third and subsequent strings from **[stave]** directives are not used automatically, but can be selected at any point in the piece by means of the **[name]** stave directive, which also provides an alternative way of specifying text and drawings for the beginnings of staves.

It is possible to specify a size for text by following the string with `/s` and a number. The number selects a text size from the list given to the **textsizes** directive, as for any other text on staves. For example,

```
[stave 1 "Flute"/s2]
```

The size is not affected by any relative magnification that may be applied to the stave. If no size is specified, the text is printed using a 10-point font.

It is also possible to request that text be vertically positioned halfway between two succesive staves. This is specified by appending `/m` (for 'middle') to the text on the upper of the two staves, for example

```
[stave 1 "Piano"/m]
```

If two over-printing staves are being used for a keyboard part, the text may appear with either of them, because if the space after the current stave is set to zero, the space for the next stave is used when positioning such text.

Finally, it is possible to specify that the text be rotated through 90° so that it prints vertically up the page. This is specified with the `/v` option, for example:

```
[stave 3 "Chorus"/v]
```

When `/v` is combined with `/m`, the text is both rotated and moved down so that its centre is at the midpoint of the staves. To make other adjustments to the position, the space character and the moving characters in the music font can be used. Only a single line of text is supported when printing is vertical, and hence the vertical bar character has no special meaning in this case.

If more than one string is given for any stave, the `/c`, `e`, `/s`, `/m`, and `/v` qualifiers can be used on any of them, and apply only to those strings on which they appear.

47.82.2 Drawings at stave starts

It is possible to cause a drawing function (see chapter 37) to be obeyed when a stave name is to be output. This can be instead of, or as well as, a text string. The amount of space to the left of the stave is controlled by the text string, and so a string consisting of blanks can be used to ensure an appropriate amount of space is left. The syntax is as follows:

`[stave <`*n*`> <`*string*`> draw <`*arguments*`> <`*drawing name*`>]`

and this feature also available for the **[name]** directive. If both a string and a call to a drawing function are present, the string must come first. An example of this usage is:

```
[stave 3 "      " draw thing]
```

As in all drawings, the arguments (which may be numbers or strings) are optional. The origin of the coordinate system is at the left-hand margin of the page and at the level of the bottom line of the stave. The drawing variable **stavestart** contains the x-coordinate of the start of the stave itself.

Just as there may be more than one string specified, for use on different systems, there may also be more than one drawing function. They are simply listed in order, following the corresponding strings, if present. For example:

```
[stave 23  "Trumpet" draw 2 d1  "Tr." draw 3 d2]
```

There is an ambiguity if an item which consists only of a string (with no associated drawing) is followed by an item consisting only of a drawing. In this case, an empty string must be specified for the second item, to prevent the drawing being incorrectly associated with the first item.

The is also a possibility of ambiguity if the first item on the stave itself is a call to a drawing function, and there is no other intervening directive. The drawing must be put into a new set of square brackets to prevent this:

```
[stave 35 "Flute"] [draw thing3]
```

As the **stave** directive is terminated by the closing square bracket, the **draw** directive in this example is taken as part of the stave data and is associated with the following note in the usual way.

The **Contrib** directory on the PMW distribution disc contains an example where a drawing function associated with a stave is used to print a special kind of ‘clef’ for guitar tablature.

47.83 [Stavelines]

This directive specifies the number of lines to be drawn for the current stave. Its parameter is a number in the range 0–6. For example:

```
[stavelines 2]
```

A stave with no lines is an invisible stave. **[Stavelines]** applies to the entire stave, independently of where it appears. It has no implications for the printing of key signatures or clefs.

The **[percussion]** directive is equivalent to **[stavelines 1]** except that, in addition, it suppresses the printing of key signatures and clefs. This directive is retained for compatibility, but its use is deprecated. New input files should instead use

```
[stavelines 1 noclef key C]
```
or
```
[stavelines 1 hclef  key C]
```

The two-line and three-line staves have double the normal stave line spacing, and are centred about the middle line of the normal five-line position. They are designed for multiple percussion parts. A three-line stave at the normal spacing can be obtained by overprinting a one-line and a two-line stave.

The four-line and six-line staves are five-line staves with the top line missing and an additional line added above the top, respectively. When used for guitar tablature they should normally be enlarged by means of the **stavesize** heading directive. The **Contrib** directory on the PMW distribution disc contains an example of guitar tablature.

For all staves other than normal five-line staves, no ledger lines are printed for notes off the stave. On one-line staves, whole bar rests are printed under the single line which is printed, and on three-line staves they are printed under the top line. In all other respects the behaviour of PMW is unchanged by the number of stave lines.

47.84 [Stemlength]

The **[stemlength]** directive is used to set a default value for the stem length adjustment on a stave. For example, after

```
[stemlength -2]
```

subsequent notes will have stems that are 2 points shorter than normal. Giving a value of zero resets to the initial state. Any stem length adjustments that are given on individual notes are added to the overall default. The name **[sl]** is a synonym for **[stemlength]**. The default stem length can be changed as often as necessary on a stave.

PMW can also be instructed automatically to shorten the stems of notes whose stems point the ‘wrong’ way. See the **shortenstems** heading directive for details.

47.85 [Stems]

Normally PMW chooses for itself in which direction to draw note stems. Details of the rules it uses are given in chapter 43 (*Stem directions*); some variation is possible by means of the **stemswap** heading directive (section 38.109).

It is also possible to force note stems to be above or below the noteheads, wherever the noteheads are on the stave. For individual notes there is an option qualifier to do this. The directive

[stems <*direction*>]

forces the stem direction for all subsequent notes that are not explicitly marked. The direction must be one of the words ‘above’, ‘below’, or ’auto’ – the last causing a reversion to the default state.

47.86 [Suspend]

When a part is silent for a long time, it is often desirable in full scores to suppress its stave from the relevant systems. The term ‘suspended’ is used in PMW to describe a stave that is not currently being printed.

The directive **[suspend]** tells PMW that it may suspend the current stave from the start of the next system, provided that there are no notes or text items to be printed in that system. The suspension ends automatically when a system is encountered in which there are notes or text items to be printed. It can be ended earlier by including the **[resume]** directive in the stave data. For example,

```
[suspend] [108] R! |
```

specifies 108 bars rest, which can be suspended where possible. Since the **[suspend]** directive actually appears in the first rest bar, at least one rest bar will be printed before the stave is suspended. If it is desired that no rest bar need be printed before the suspension, **[suspend]** should be placed in the preceding bar:

```
abcd [suspend] | [108] R! |
```

If at least one rest bar is required when the stave is resumed, an explicit **[resume]** must appear in the last rest bar, as by default the stave may resume with a non-rest bar at the beginning of a system. For example

```
[suspend] [107] R! | [resume] R! |
```

When a single part is being printed, **[suspend]** has no effect, as multiple rest bars are automatically packed up into a single bar with a multiple rest sign.

Since the **[suspend]** stave directive only takes effect from the start of the following system of staves, it cannot be used to cause suspension right at the start of a piece. The **suspend** heading directive is provided for this purpose.

47.87 [Tenor]

This specifies a C clef with its centre on the fourth stave line. See **[alto]** for further details of clef directives.

47.88 [Text]

The default type for text within a stave (which implies a default vertical level and size) can be set for an individual stave by means of the **[text]** directive. It takes a word as its parameter:

`[text above]`	selects printing above the stave
`[text above <`*n*`>]`	ditto, at a given level
`[text below]`	selects printing below the stave
`[text below <`*n*`>]`	ditto, at a given level
`[text underlay]`	selects underlay printing
`[text overlay]`	selects overlay printing
`[text fb]`	selects figured bass printing

To override a default with an absolute position, for example,

```
[text above 15]
```

a plain `/a` or `/b` without a following number can be used (as well as `/ul`, `/fb`, or `/m`). Similarly, the appearance of a **[text]** directive without a number resets to the initial state, where the default vertical position depends on the next note.

Ordinary text that is printed above or below the stave is by default printed at size 1 (as specified by the **textsize** heading directive). Underlay, overlay, and figured bass text is printed by default at the sizes specified by the **underlaysize**, **overlaysize**, and **fbsize** heading directives.

The default text type can always be overridden by explicit qualifiers following the string. For example, if

```
[text underlay]
```

has been specified, then an italic dynamic marking to be printed above the stave is coded as

```
"\it\ff"/a
```

The text type can be changed many times within one stave.

47.89 [Textfont]

The default typeface for non-underlay, non-overlay text that is printed with a stave can be set for an individual stave by means of the **[textfont]** directive. This directive takes as its parameter one of the standard font names. For example, supposing that the third extra font had been defined for some special use in a stave's text, then

```
[textfont extra 3]
```

would be used at the start of that stave. The default typeface for non-underlay, non-overlay text is italic. In any given text string it is always possible to change typeface by using the appropriate escape sequence.

47.90 [Textsize]

This directive must be followed by a number in the range 1 to 12. It selects the default size to be used for text that is neither underlay nor overlay nor figured bass. The actual font sizes that correspond to the twelve available sizes are set by the **textsize** heading directive. If this directive is not used, the default size is size 1. Individual text strings can have their sizes set by means of the `/s` option.

47.91 [Tick]

The **[tick]** directive causes PMW to insert a tick pause mark above the current stave.

47.92 [Ties]

Normally PMW draws tie marks on the opposite side of the noteheads from the stem. However, it is possible to force ties to be above or below the noteheads. For individual ties there is an option qualifier to do this. In addition, the directive

```
[ties <direction>]
```

is available for forcing the tie direction for all subsequent ties that are not explicitly marked. The direction must be one of the words ‘above’, ‘below’, or ’auto’ – the last causing a reversion to the default state.

The effect on chords of forcing the direction of tie marks is to force *all* the marks for a chord to the given side of the noteheads.

47.93 [Time]

The time signature of a part can be changed at the start of a bar by the directive

```
[time <time signature>]
```

If the change of time falls at the start of a system, a cautionary time signature is printed at the end of the previous line unless the word ‘nowarn’ is included in the directive, for example:

```
[time 6/8 nowarn]
```

There is also a heading directive, **notimewarn**, for suppressing all cautionary time signatures.

PMW will not work sensibly by default if different time signatures are used on different staves, unless they represent the same number of musical notes. For example, if one stave is in 3/4 time and another is in 6/8 all will be well, but PMW cannot cope with 2/4 against 6/8 without additional input (see below).

Sometimes it is required to print two time signatures at the start of a piece (for example, if there are alternate bars in different times). The simplest way to do this is to make use of the **printtime** heading directive.

When a bar starts with a new time signature and a repeat mark, the order in which these are printed depends on the order in which they appear in the input.

```
[time 4/4] (:
```

causes the time signature to be printed first, followed by the repeat mark, while

```
(: [time 4/4]
```

causes the repeat mark to be amalgamated with the previous bar line, with the time signature following. If, at the same point in the music, these items appear in different orders on different staves, then the repeat sign is printed first on all staves.

47.93.1 Staves with differing time signatures

PMW requires no special action to handle staves with different time signatures if the actual barlengths (measured in notes) are the same. For example 3/4 and 6/8 bars both contain six quavers, and so are compatible.

PMW can also handle time signatures that are not compatible in this way, for example, 6/8 in one stave and 2/4 in another, but because PMW handles just one stave at a time when it is reading the music in, it is necessary to tell it what is going on by giving a second time signature in the **[time]** directive, preceded by `->`. For example:

```
time 6/8
[stave 1 treble 1] a. b-a-g- | [endstave]
[stave 2 bass 0 time 2/4 -> 6/8]
c-d-; e-f- | [endstave]
```

The first signature is the one printed, and this corresponds to the printed notes in the bar; the second signature is the one from the other stave, and the notes are stretched or compressed (in position when printing and in time when generating a MIDI file) to make the bar lengths match.

47.94 [Topmargin]

This directive provides a way of changing the value given by the **topmargin** heading directive for a single page only. If there is more than one occurrence on the same page, the last value is used. To leave 30 points at the top of one particular page, for example, use

```
[topmargin 30]
```

in any bar on that page.

47.95 [Transpose]

The directive

```
[transpose <n>]
```

specifies that the current stave is to be transposed by <*n*> semitones when printed. It normally appears at the beginning of a stave's data. If transposition is also specified at an outer level (either in the heading, or by using the **-t** command line option), the transposition specified here adds to, rather than replaces it.

Octave transposition, as specified in a clef-setting directive or by the **[octave]** directive, is added to any general transposition.

PMW does not transpose the current key signature when it encounters the **[transpose]** directive, but it does transpose any subsequently encountered key signatures. If you require a transposed key signature for a stave which has its own transposition specified, you must include the key signature after **[transpose]**, even if it is the same key signature that is specified in a heading directive for the whole piece. For example:

```
key G
[stave 1]
@...
[endstave]
[stave 2 transpose 1 key G]
@...
[endstave]
```

Note that it is the *old* key signature that is specified. In this example it is transposed to become A-flat.

Further details about transposition of notes are given in chapter 29, and details of the transposition of chord names are given in section 34.4.

47.96 [Transposedacc]

This directive must be followed by one of the words 'force' or 'noforce'. It changes the option for forcing the printing of an accidental on a transposed note when there was an accidental on the original, even if the accidental is not strictly needed. For details, see chapter 29.

47.97 [Treble]

This specifies a treble clef. See **[alto]** for further details of clef directives.

47.98 [Trebledescant]

This specifies a treble clef with a little '8' printed above it. See **[alto]** for further details of clef directives.

47.99 [Trebletenor]

This specifies a treble clef with a little '8' printed below it. See **[alto]** for further details of clef directives.

47.100 [TrebletenorB]

This specifies a clef that is exactly like the trebletenor clef, except that the little ‘8’ is printed enclosed in round brackets.

47.101 [Tremolo]

It is possible to print tremolo markings that appear as beams between notes that are not normally beamed, or as disconnected beams between notes.

This is requested by placing the stave directive **[tremolo]** between the two notes. There are two optional qualifiers: `/x` followed by a number specifies the number of beams to draw, while `/j` followed by a number specifies the number of beams which are to be joined to the note stems. The default is to draw two beams, neither of which is joined to the stems. Thus,

```
  g [tremolo] b
```

prints two crotchets, with two disconnected beams between them, while

```
  G [tremolo/x3/j3] B
```

prints two minims, joined by three beams. The `/j` qualifier should not be used with breves or semibreves.

For the most commonly encountered tremolos, it is necessary to print a note of the ‘wrong’ length. For example, a tremolo that lasts for the length of a crotchet is printed as two crotchets, at the positions two quavers would occupy, with tremolo bars between them. This effect can be achieved by using the masquerading feature described in section 41.13.

The **[tremolo]** directive must appear between two notes in a bar. It is ignored if it appears at the beginning or end of a bar, or if it is preceded or followed by a rest. It assumes that the notes are of the same kind, and have their stems in the same direction. Notes with flags should not be used, though tremolos can be added underneath the normal beams of a beamed group if necessary.

47.102 [Triplets]

This directive is used to control the printing of triplet and other irregular note group markings. Despite its name, it applies to all irregular note groups. It must be followed by one of the words

```
  on  off  above  below  bracket  nobracket auto
```

The words ‘on’ and ‘off’ are used to control the printing of the ‘3’, with or without a bracket, above or below a group of triplets (or equivalent for other groups). When the ‘off’ option is selected, nothing is printed. Note that the qualifier `/x` can be used to suppress printing for an individual triplet, or to enable it, if it has been disabled by the **[triplets]** directive.

The other words set default options for printing irregular note groups, and they are independent of each other. Specifying

```
  [triplets above]
```

causes all the irregular note markings to be printed above the notes, but (unlike the `/a` option on an individual group) it does not specify whether a horizontal bracket should be drawn or not.

The words ‘above’ and ‘below’ can be followed by a dimension, to set a fixed vertical position for all subsequent irregular note group markings. If the dimension is preceded by `+` or `-`, this does not set a fixed position, but provides a default vertical adjustment for subsequent irregular groups. For example,

```
  [triplets above +4]
```

causes subsequent markings to be printed four points higher than they would otherwise appear.

Brackets can be forced or inhibited by means of ‘bracket’ and ‘nobracket’. If neither has been specified, then a bracket is drawn unless the note group is beamed.

The ‘auto’ option resets both the position and the bracketing options to their initial states, where the markings may be printed above or below the notes, depending on their pitch, and the bracket is omitted only if the notes are beamed.

Options given on an individual note group override the defaults set by the **[triplets]** directive. Note that the use of `/a` or `/b` forces a bracket to be drawn, unless followed by `/n`.

47.103 [Ulevel] and [ulhere]

For each stave that it prints, PMW chooses a default underlay level at which to print underlay text. The standard position for this level (the base line level for the text) is 11 points below the bottom line of the stave, but a lower level is chosen if there are low notes present on the stave. There are two different ways of changing this level. The **[ulhere]** directive specifies a temporary change for the current line, while the **[ulevel]** directive sets an absolute level to be used until further notice.

[Ulhere] takes a positive or negative number as an parameter. This is interpreted as a number of points to add to the automatically computed level for the line in which the current bar appears. For example,

```
  [ulhere -2]
```

has the effect of lowering the current underlay line by two points. If a subsequent occurrence of **[ulhere]** appears in the same line for the same stave, it is accepted if its parameter is negative and specifies a lower level than the previous one, or if its parameter is positive and all previous ones were positive and it is greater than any of them.

[Ulhere] has no effect if an absolute underlay level is being forced by means of the **[ulevel]** directive, which sets a level relative to the bottom of the stave. Thus, for example,

```
  [ulevel -15]
```

requests a level fifteen points the bottom of the stave. This directive takes effect for the text that is printed under all the notes that follow it, even if the text was input earlier as part of a multi-syllable input string.

[Ulevel] may appear as often as necessary; its effect lasts until the end of the movement or its next appearance. However, if **[ulevel]** appears with an asterisk for an parameter:

```
  [ulevel *]
```

then the underlay level reverts to the value automatically selected by PMW, and any subsequent **[ulhere]** directives will be honoured.

47.104 [Ultextsize]

This directive must be followed by a number in the range 1 to 12. It selects the default size to be used for underlay text on the current stave. The actual font sizes that correspond to the twelve available sizes are set by the **textsize** heading directive. If this directive is not used, the size set by the **underlaysize** heading directive (which is different from any of the sizes set by **textsize**) is used.

[Ultextsize] is normally only needed if you want different sizes of underlay text on different staves.

47.105 [Unbreakbarline]

An occurrence of this directive causes the bar line at the end of the current bar to be extended downwards onto the stave below. This could be used, for example, to print a double barline right through a system at the end of a verse or other important point in a choral piece, where the barlines are normally broken after each stave. See also **[breakbarline]**.

47.106 [Underlayfont]

The default typeface for underlay text that is printed with a stave can be set for an individual stave by means of the **[underlayfont]** directive. This directive takes as its parameter one of the standard font names.

For example, supposing that the third extra font had been defined for use in underlay text, then

```
  [underlayfont extra 3]
```

would be used at the start of that stave. The default typeface for underlay text is roman. In any given text string it is always possible to change typeface by using the appropriate escape sequence.

47.107 [Xline]

See **[line]** (section 47.40) and subsection 47.75.6 within the description of **[slur]**.

47.108 [Xslur]

See subsection 47.75.6 within the description of **[slur]**.

48. The PMW musical font

This chapter contains a list of all the available characters in the musical font, which is called PMW-Music. Characters from the music font can be referenced by number in character strings. Those with character codes less than 127 can also be referenced by switching to the music font and entering the corresponding ASCII character. The following two examples produce the same effect:

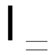
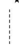
```
"\rm\this clef \*33\ is treble"
"\rm\this clef \mu\!\rm\ is treble"
```

The second method is more convenient when a whole sequence of music font characters is required.

Most of the characters in the music font print ‘on the baseline’ in the typographic sense, though some have ‘descenders’. The only exceptions to this are the constituent parts of notes, such as stems and quaver tails. The typographic character widths, which are not used by PMW when setting music, are set to values which are useful when these characters are printed as text.

Here is a list of the characters in the font, giving both their numbers and, where relevant, the corresponding ASCII characters. The printing width is also given, as a fraction of the font size. For example, when a 10-point treble clef is printed, the printing position is advanced to the right by 15 points.

	32	0.75		space
!	33	1.5	𝄞	
"	34	1.5	𝄢	
#	35	1.5	𝄡	
$	36	1.0	𝆯	piano end pedal sign
%	37	0.6	♯	
&	38	0.6	𝄪	double sharp
'	39	0.5	♭	
(	40	0.45	♮	
)	41	0.0	𝄐	
*	42	0.66	𝄺	breve rest
+	43	0.66	𝄻	semibreve rest
,	44	0.66	𝄼	minim rest
–	45	0.66	𝄽	
.	46	0.59	𝄾	
/	47	0.0	𝄑	
0	48	3.5	𝄩	many bars rest
1	49	1.34	𝅜	
2	50	0.84	𝅝	
3	51	0.84	𝅗𝅥	
4	52	0.84	𝅗𝅥	
5	53	0.84	𝅘𝅥	
6	54	0.84	𝅘𝅥	
7	55	1.2	𝅘𝅥𝅮	
8	56	0.84	𝅘𝅥𝅮	
9	57	1.2	𝅘𝅥𝅯	
:	58	0.84	𝅘𝅥𝅯	
;	59	0.0	𝅮	repeatable tail
<	60	0.0	𝅮	repeatable tail
=	61	0.0	—	ledger line
>	62	0.0	·	vertical dot (above note on base line)
?	63	0.4	.	horizontal dot
@	64	0.6	𝄀	bar line
A	65	0.76	𝄁	double bar line

B	66	0.76		thick bar line
C	67	1.0		normal stave
D	68	1.0		percussion stave
E	69	0.0		up quaver tail
F	70	10.0		long stave
G	71	10.0		long percussion stave
H	72	0.0		down quaver tail
I	73	0.6		
J	74	0.0		upward note stem
K	75	0.0		downward note stem
L	76	0.84		solid notehead
M	77	0.84		minim notehead
N	78	0.6		
O	79	0.0		
P	80	0.0		
Q	81	0.0		
R	82	0.0		
S	83	0.0		
T	84	0.0		horizontal bar accent
U	85	0.0		
V	86	1.0		
W	87	0.0		
X	88	0.0		
Y	89	0.0		
Z	90	0.0		
[	91	0.6		dashed bar line
|	92	1.0		single-line caesura
]	93	0.0		for use with clefs
^	94	1.0		
_	95	1.0		
`	96	0.4		suitable for following *tr*
a	97	0.0		
b	98	0.0		
c	99	1.5		
d	100	1.5		
e	101	0.0		
f	102	0.0		
g	103	0.0		
h	104	0.0		
i	105	0.0		
j	106	0.55		
k	107	0.76		
l	108	0.84		solid diamond notehead
m	109	0.84		diamond notehead
n	110	0.84		cross notehead
o	111	0.0		up stem for cross
p	112	0.0		down stem for cross
q	113	0.0		up stem fragment, 0.2 to 0.4
r	114	0.0		down stem fragment, 0 to –0.2
s	115	0.5		
t	116	0.55		dot for guitar box
u	117	0.55		circle for guitar box

v	118	0.0		move down by 0.1
w	119	0.0		move down by 0.4
x	120	0.0		move up by 0.4
y	121	-0.1		move left by 0.1
z	122	0.1		move right by 0.1
{	123	-0.33		move left by 0.33
|	124	0.0		move down by 0.2
}	125	0.55		move right by 0.55
~	126	0.0		move up 0.2
	127	-		unassigned
	128	0.6	√	
	129	0.0	⁄	accaciatura bar
	130	0.0	∖	accaciatura bar
	131	0.0	▦	for guitar chords
	132	0.6	ı	short bar line
	133	0.0	‖	
	134	0.0	∘	
	135	0.0	+	
	136	0.9	*tr*	
	137	0.6	'	short vertical caesura
	138	0.6	|	long vertical caesura
	139	0.35	[	
	140	0.35	]	brackets for accidentals
	141	0.35	(	
	142	0.35	)	
	143	0.5	/	
	144	0.0	˙.	
	145	0.0		for arpeggios – moves upwards by 0.4
	146	0.0		tremolo bar – moves upwards by 0.4
	147	1.0	○	
	148	1.0	⦶	
	149	0.0	◡	slur
	150	0.0	◡	slur
	151	0.0	↗	
	152	0.0	↘	
	153	1.0	ↄ	
	154	1.0		
	155	1.58		unison breve
	156	0.0	'	'start of bar' accent
	157	0.35	(	for bracketing *8*
	158	0.35	)	
	159	0.33	¬	for 8va lines etc.
	160	0.33	⌐	
	161	0.33	⌟	
	162	0.33	⌞	
	163	1.4	Ped.	
	164	0.0		for arpeggios – moves upwards by 0.4
	165	0.0		for arpeggios – moves upwards (sic) by 0.4
	166	0.0	⌒	harp nail symbol
	167	0.333	⌝	alternate bracket angles
	168	0.333	⌟	

169	1.0	=	2-line stave
170	1.0	≡	3-line stave
171	1.0	≣	4-line stave
172	1.0	≣	6-line stave
173	1.5	𝄥	percussion clef
174	1.5	𝄢	old-style F clef
175	1.5	𝄡	old-style C clef
176	0.0	⎧	bracket top
177	0.0	⎩	bracket bottom
178	1.0	𝆫	symbol for pitch without duration (‘direct’)
179	0.55	5⁺	
180	0.75	△	major chord sign (jazz notation)
181	0.675	○	diminished chord sign
182	0.675	∅	‘half diminished’ chord sign
185			prints nothing; moves left 0.42, up 0.4
186			prints nothing; moves left 0.76, down 0.4
187			prints nothing; moves up 1.2
188			prints nothing; moves down 1.2
247	10.0		long 2-line stave
248	10.0		long 3-line stave
249	10.0		long 4-line stave
250	10.0		long 6-line stave

The characters numbered 118–126 and 185–188 do not cause anything to be printed; instead they just cause the current printing position to be moved by a distance which depends on the point size of the font. The values given above are the factors by which the font’s point size must be multiplied in order to get the relevant distance. For example, if a 10-point font is in use, then character number 119 (w) moves the current printing position down by 4 points. If a space character (number 32) is printed from the music font, it moves the printing position by 0.75 units to the right.

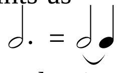
The larger round and square brackets (characters 139–142) are designed so that they can be printed directly before and after an accidental, except that for a flat they need to be raised by one note pitch (2 points).

The large circle characters have a diameter of two stave spaces, and are intended for printing original time signatures (see the example in section 38.118).

The slur characters are not used by PMW itself, but are for printing ties when using note characters in text. The first is the correct width for two successive note characters; the second is the correct width when the first note is followed by a dot. Because they have zero width, they should be printed before the notes. For example, the PMW string

```
"\**m.\ = \mf\\149\\51\\53\"
```

prints as



The slanting arrows are for use at the ends of staves when a stave containing multiple parts is about to be split into two or more staves, and *vice versa*.

49. The PMW-Alpha font

Richard Hallas contributed an auxiliary font for use with PMW and with other programs. It is called PMW-Alpha, because it is designed for printing musical symbols in conjunction with normal alphabetic text. The font was originally designed as an Acorn RISC OS font; the PostScript version was generated automatically from the original.

The characters that PMW-Alpha contains fall into four classes:

- There is a set of letters such as ***f*** and ***p*** which are in a style commonly seen in music, and which can be used to print dynamic markings.
- There is a set of digits in a style commonly seen in time signatures, together with a matching plus sign.
- There is a set of fractions, suitable for use in organ registrations. There are also two sets of small digits, one raised and one lowered, that can be used to build additional fractions.
- There are small versions of many musical characters such as notes, accidentals, and clefs. These are at appropriate sizes for mixing with text fonts of the same nominal size, which makes it easier to include them with text when using desktop publishing programs.

49.1 Use of PMW-Alpha from within PMW

Here is an example of some heading directives that could be used to make use of the PMW-Alpha font from within PMW:

```
Textfont extra 1 "PMW-Alpha"
Timefont extra 1
*Define f    "\x1\f"
*Define fr37 \x1\\203\\222\\217\
```

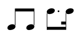
The **textfont** directive sets up PMW-Alpha as the first extra font; the **timefont** directive specifies that numerical time signatures are to be printed using this font.

The first ***define** directive defines a macro for the *forte* marking that prints ***f***. The second macro is for printing the fraction $^3/_7$. It is set up without surrounding quote marks so that it can be included in a longer string, for example

```
"an unusual fraction is &fr37\rm\"
```

49.2 Use of PMW-Alpha in other programs

From within a desktop publishing program, PMW-Alpha can be used as a kind of musical typewriter. Richard Hallas explains:

“The keys Q, W, E, R, T, Y produce notes of descending duration from breve to semiquaver. The lower-case versions of the letters produce notes with up-stems, and the upper-case characters produce down-stem notes. The dot key produces a dot which is suitable to follow any up-stem note, and the > key (shifted dot) produces a suitable dot for the down-stem notes. There are also some simple beams which can be used with the crotchet characters. The keys h, j, and k produce beams for upstemmed notes, while H, J, and K are for use with downstemmed notes. They should be typed after the first note, and all have a width of zero, so do not move the cursor. For example, typing ‘rh,r RK><R’ produces .

The five keys to the left of **return** produce rests: [and] for a semibreve rest 𝄻 and a minim rest 𝄼, semicolon and quote for a quaver rest 𝄾 and a semiquaver rest 𝄿, and / for a crotchet rest 𝄽. Pressing any of these keys in conjunction with **shift** produces a smaller version of each rest, suitable for use in among the notes. The letters G, B and V produce treble, bass, and alto clefs, and are named after the G clef, Bass clef and Viola clef. These are suitable for use with the notes.

These symbols may of course be used on their own, but a stave symbol is also provided so that the symbols can be printed on it. The symbols have all been positioned correctly so that the work with the stave. The stave is obtained by entering the ‘hard’ space character ASCII 160. (How you do this depends on the keystroke conventions of your wordprocessor.) The stave character has a width of zero, so following it with one of the notes etc. will produce a composite symbol. The actual space taken up by the stave character is 640, the same as the normal <space> character moves, so a string of alternating hard and soft spaces produces an empty stave.

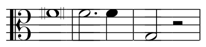
The breves are not supposed to be used with dots, and have a width of 640. All the other notes have a width of 380 and can be followed by either two dots, a dot and a ‘dot space’ (< character) or a ‘double-dot space’ (comma character) to make up a 640-width ‘unit’.

The three clefs each have a 640 ‘unit-width’. The small rests have a width of 260, and can either be followed by a note or a ‘note space’ (\ character). The | character produces a bar line, whose actual width is identical to its printing width so that it can be immediately followed by another stave character.

Example: If ‘Hd’ represents a hard space character, then this sequence

Hd V Hd < Hd Q Hd < | Hd < Hd E > < Hd R , | Hd , Hd e , Hd } \ |

produces:



There are two systems available for fractions. Firstly, the most common fractions ⅓, ⅔, and ⅗ are in the ASCII positions normally occupied by superscript digits ^{1}, ^{2}, and ^{3}. The standard ¼, ½, and ¾ are in their normal ASCII positions, and various other common fractions are available in a fairly sensible order from adjacent characters.

Secondly, any fraction can be made from the individual fraction numbers and character 222. The ready-made fractions are preferable to the custom ones because the relative positions of the numbers and the slash are slightly neater, but nevertheless the custom fractions will usually give a good result. The fractions are in Times-Roman style, and should go well with that font.”

49.3 Characters in the font

Here is alist of the characters in the PMW-Alpha font, in the same format as the list of characters in the PMW-Music font above. The stemless notes (breve and semibreve) appear twice, in both the ‘upstem’ and ‘downstem’ positions, and for technical reasons, the treble clef appears on the lower case g as well as the upper case G.

	32	0.64		space
"	34	0.26	𝄿	small semiquaver rest
#	35	0.46	♯	
$	36	0.38	♭	
%	37	0.36	♮	
&	38	0.7	𝄫	
'	39	0.38	𝄿	big semiquaver rest
(	40	0.924	←	
)	41	0.924	→	
*	42	0.77	❊	
+	43	0.64	+	for time signatures
,	44	0.26		‘double dot’ space
.	46	0.13	.	dot for upstemmed notes
/	47	0.38	𝄽	big crotchet rest
0	48	0.64	**0**	
1	49	0.64	**1**	
2	50	0.64	**2**	
3	51	0.64	**3**	
4	52	0.64	**4**	
5	53	0.64	**5**	
6	54	0.64	**6**	
7	55	0.64	**7**	
8	56	0.64	**8**	
9	57	0.64	**9**	

:	58	0.26	𝄾	small quaver rest
;	59	0.38	𝄾	big quaver rest
<	60	0.13		‘dot’ space
=	61	0.564	=	
>	62	0.13	·	dot for downstemmed notes
?	63	0.26	𝄽	small crotchet rest
B	66	0.64	𝄢	
C	67	0.64	𝄴	
E	69	0.38	𝅮	
F	70	0.80	𝄐	
G	71	0.64	𝄞	
H	72	0.00	▬	
J	74	0.00	═	beams for downstemmed notes
K	75	0.00	▬	
L	76	0.38	●	crotchet notehead
M	77	0.38	○	minim notehead
O	79	0.80	𝄌	
P	80	1.40	𝆮	
Q	81	0.64	𝅜	
R	82	0.38	𝅮	
S	83	0.72	𝄋	
T	84	0.38	𝅯	
V	86	0.64	𝄡	
W	87	0.38	𝅝	semibreve
Y	89	0.38	𝅰	
[	91	0.51	𝄼	minim rest, with ledger
\	92	0.38		‘note’ space
]	93	0.51	𝄻	semibreve rest, with ledger
^	94	0.46	𝄪	double sharp
_	95	0.53	‿	for joining words
`	96	0.70	𝆗	
c	99	0.64	𝄵	
d	100	0.817	↘	
e	101	0.38	𝅗𝅥	
f	102	0.47	𝆑	
g	103	0.64	𝄞	
h	104	0.00	▔	
j	106	0.00	=	beams for upstemmed notes
k	107	0.00	▔	
m	109	0.88	𝆐	
o	111	0.76	𝆖	
p	112	0.69	𝆏	
q	113	0.64	𝅜	
r	114	0.38	𝅘𝅥	
s	115	0.32	*s*	
t	116	0.38	𝅘𝅥𝅮	
u	117	0.817	↗	
w	119	0.38	𝅝	semibreve
y	121	0.38	𝅘𝅥𝅯	
z	122	0.42	*z*	
{	123	0.26	𝄻	small semibreve rest
|	124	0.02	|	bar line

}	125	0.26	-	small minim rest
~	126	0.70	∾	
	160	0.00	≣	stave segment
	178	0.75	⅔	
	179	0.75	⅗	
	180	0.75	⅕	
	181	0.75	⅖	
	182	0.75	⅘	
	183	0.75	⅐	
	184	0.75	2⁄7	
	185	0.75	⅓	
	186	0.75	⅑	
	187	0.75	8⁄9	
	188	0.75	¼	
	189	0.75	½	
	190	0.75	¾	
	200	0.30	^{0}	for fraction numerators
	201	0.30	^{1}	
	202	0.30	^{2}	
	203	0.30	^{3}	
	204	0.30	^{4}	
	205	0.30	^{5}	
	206	0.30	^{6}	
	207	0.30	^{7}	
	208	0.30	^{8}	
	209	0.30	^{9}	
	210	0.30	$_{0}$	for fraction denominators
	211	0.30	$_{1}$	
	212	0.30	$_{2}$	
	213	0.30	$_{3}$	
	214	0.30	$_{4}$	
	215	0.30	$_{5}$	
	216	0.30	$_{6}$	
	217	0.30	$_{7}$	
	218	0.30	$_{8}$	
	219	0.30	$_{9}$	
	222	0.75	⁄	for building fractions

50. Syntax summary

50.1 Preprocessing directives

These may occur at any point in an input file; each one must occupy a line on its own.

`*comment` <*rest of line*>
`*define` <*name*> <*rest of line*>
`*else`
`*fi`
`*if` <*condition*>
`*if not` <*condition*>
`*include "`<*file name*>`"`

50.2 Heading directives

Those marked with an asterisk may appear only at the head of a PMW input file, not at the start of the second or subsequent movements. Those marked with a dagger affect only the movement in which they appear.

`Accadjusts` <*n*> <*n*> `...` <*up to 8 numbers*>
`Accspacing` <*n1*> <*n2*> <*n3*> <*n4*> <*n5*>
`Bar` <*n*>
`Barcount` <*n*>
`Barlinesize` <*n*>
`Barlinespace` <*n*>
`Barlinestyle` <*n*>
`Barnumberlevel` <*sign*><*n*>
`Barnumbers` <*enclosure*> <*interval*> <*fontsize*> <*font*>
`Beamendrests`
`Beamflaglength` <*n*>
`Beamthickness` <*n*>
`Bottommargin` <*n*>
`Brace` <*n-m*> `...`
`Bracestyle` <*n*>
`Bracket` <*n-m*> `...`
`Breakbarlines` <*n1*> <*n2*> `...`
`Breakbarlinesx` <*n1*> <*n2*> `...`
`Breveledgerextra` <*n*>
`Breverests`
`Caesurastyle` <*n*>
`Check`
`Checkdoublebars`
`Clefsize` <*n*>
`Clefstyle` <*n*>
`Clefwidths` <*n1*> `...` <*n5*>
`Copyzero` <*n*>`/`<*m*> `...`
`Cuegracesize` <*n*>
`Cuesize` <*n*>
`Dotspacefactor` <*n*>
`Doublenotes`
`Draw` <*drawing definition*> `enddraw`
`Endlinesluradjust` <*n*>
`Endlineslurstyle` <*n*>
`Endlinetieadjust` <*n*>
`Endlinetiestyle` <*n*>
`Extenderlevel` <*n*>
`Fbsize` <*n*>
`Footing` <*fontsize*> `"`<*string*>`"` <*space*>
`Footing draw` <*name*> <*space*>
`Footnotesep` <*n*>

```
 Footnotesize <n>
 Gracesize <n>
 Gracespacing <n>
 Gracestyle <n>
 Hairpinlinewidth <n>
 Hairpinwidth <n>
 Halvenotes
 Heading <fontsize> "<string>" <space>
 Heading draw <name> <space>
 Hyphenstring "<string>"
 Hyphenthreshold <n>
 Join <n-m> ...
 Joindotted <n-m> ...
 Justify <edges>
†Key <key signature>
 Keydoublebar
 Keysinglebar
 Keywarn
*Landscape
 Lastfooting <fontsize> "<string>" <space>
 Lastfooting draw <name> <space>
†Layout <n1> <n2> ...
 Leftmargin <n>
 Linelength <n>
 Longrestfont <fontsize> <font>
*Magnification <n>
 Maxbeamslope <n>
*Maxvertjustify <n>
 Midichannel <n> "<name or number>" <staves>
 Midkeyspacing <n>
 Midtimespacing <n>
*Musicfont "<font name>"
 Nobeamendrests
 Nocheck
 Nocheckdoublebars
*Nokerning
 Nokeywarn
 Nosluroverwarnings
 Nospreadunderlay
 Notespacing *<factor>
 Notespacing <n1> ... <n8>
†Notime
 Notimebase
 Notimewarn
 Nounderlayextenders
 Overlaydepth <n>
 Overlaysize <fontsize>
*Page <n> <m>
 Pagefooting <fontsize> "<string>" <space>
 Pagefooting draw <name> <space>
 Pageheading <fontsize> "<string>" <space>
 Pageheading draw <name> <space>
*Pagelength <n>
 Playtempo <n> <n/m> ...
 Playtranspose <n/m> ...
 Playvolume <n> <n/m> ...
 Pmwversion <n>
 Printtime <time> "<string 1>" "<string 2>"
 Psfooting "<PostScript string>"
 Psheading "<PostScript string>"
 Pslastfooting "<PostScript string>"
```

 Pspagefooting "<*PostScript string*>"
 Pspageheading "<*PostScript string*>"
*Pssetup "<*PostScript string*>"
 Rehearsalmarks <*style*> <*fontsize*> <*fontname*>
 Repeatbarfont <*fontsize*> <*font*>
 Repeatstyle <*n*>
 Selectstaves <*n-m*> ...
*Sheetdepth <*n*>
*Sheetsize A4 *or* A3
*Sheetwidth <*n*>
 Shortenstems <*n*>
 Sluroverwarnings
 Smallcapsize <*n*>
†Startbracketbar <*n*>
 Startlinespacing <*c*> <*k*> <*t*> <*n*>
†Startnotime
 Stavesize(s) <*n/m*> ...
 Stavespacing <*n*> <*n/b*> ...
 Stavespacing <*n*> <*n/a/b*> ...
 Stemlengths <*n1*> ... <*n6*>
 Stemswap <*direction*>
 Stemswaplevel <*n/m*> ...
†Suspend <*n*> ...
 Systemgap <*n*>
*Textfont <*fontword*> "<*font name*>"
 Textsize(s) <*n*> ...
 Thinbracket <*n-m*> ...
†Time <*time signature*>
 Timebase
 Timefont <*fontsize*> <*name*>
 Timewarn
 Topmargin <*n*>
†Transpose <*n*>
 Transposedacc force
 Transposedacc noforce
 Transposedkey <*key 1*> use <*key 2*>
 Trillstring "<*string*>"
 Tripletfont <*fontsize*> <*name*>
 Tripletlinewidth <*n*>
 Underlaydepth <*n*>
 Underlayextenders
 Underlaysize <*fontsize*>
 Underlaystyle <*n*>
†Unfinished
 Vertaccsize <*n*>

50.3 Note and rest components

The order of the items that go to make up one note is given below. Few notes require all possible components to be present.

accidental	`# ## $ $$ %`
invisible	`?`
over note	`o`
under note	`u`
transposed	`^# ^## ^$ ^$$ ^% ^- ^+`
bracketed	`) ]`
moved	`<` or `<`*number*
note letter	`a-g A-G q Q r R s S`
octave sign	`'` to raise, `` ` `` to lower
note flags	`- = =- ==` following a small letter
	`+ ++` following a capital letter

	`!` following `q`, `Q`, `r`, `R`, `s`, or `S`
move dot	`>` or *number*`>`
dot(s)	`.` or `..` or `.+`
expression/options	`\<`*expression and options indications*`>\`
tie/slur	`_`
above/below	`/a /b`
editorial	`/e`
dashed/dotted	`/i /ip`
full beam break	`;`
partial beam break	`,`

The possible expression/option codes are:

`!`	accent on stem side, trill or fermata below
`:`	augmentation dot other side if note on line
`::`	augmentation dot raised if note in space
`'`	‘start of bar’ accent
`.`	staccato
`-`	accent
`>`	horizontal wedge accent
`~`	upper mordent
`~|`	lower mordent
`~~`	double upper mordent
`~~|`	double lower mordent
`/`	single tremolo
`//`	double tremolo
`///`	triple tremolo
`a<`*n*`>`	accent <*n*>
`ar`	arpeggio
`aru`	arpeggio with up arrow
`ard`	arpeggio with down arrow
`c`	print on coupled stave
`C`	centre if only note in bar
`d`	string down bow
`f`	fermata (pause) above note
`f!`	fermata (pause) below note
`g`	grace note
`g/`	crossed grace note
`h`	don’t print on coupled stave
`l<`*n*`>`	rest level
`m<`*flags*`>`	masquerade
`o`	harmonic
`sl<`*n*`>`	extend stem length
`sl-<`*n*`>`	shorten stem length
`sp`	spread chord
`su`	stem up
`sd`	stem down
`sw`	swap stem direction in beam
`t`	turn
`t|`	inverted turn
`tr`	trill
`tr#`	trill with sharp
`tr$`	trill with flat
`tr%`	trill with natural
`u`	string up bow
`v`	small, closed vertical wedge accent
`V`	large, open vertical wedge accent
`x`	cancel default expression

Accent numbers:

1	staccato dot
2	horizontal bar
3	horizontal wedge
4	small, closed vertical wedge
5	large, open vertical wedge
6	string down bow
7	string up bow
8	ring
9	‘start of bar’ accent

All accents and ornaments can be moved in any direction by following the code with `/u`, `/d`, `/l`, or `/r` and a number. For this reason, the tremolo options must not directly follow an accent or ornament. Use a space to separate, or put the tremolo first.

50.4 Special characters in stave data

These characters, along with text items, occur interspersed in the notes and rests:

`|`	bar line
`||`	double bar line
`|?`	invisible bar line
`|=`	bar line with beam carried over it
`|<`*n*`>`	bar line in style <*n*>
`:`	dotted bar line in middle of bar
`(:`	start repeated section
`:)`	end repeated section
`{`	start triplet
`{<`*n*`>`	start non-standard group
`}`	end non-standard group
`//`	caesura
`>`	decrescendo hairpin
`<`	crescendo hairpin

50.5 Stave text item options

These options may occur after any text item, but for a rehearsal marking inside square brackets only those specifying movement are relevant, and `/bar`, `/c`, `/e`, `/nc`, `/ne`, and `/ts` are ignored on underlay and overlay strings.

`/a`	print above the stave
`/ao`	print above, at the overlay level
`/a<`*n*`>`	print <*n*> points above the stave
`/b`	print below the stave
`/bar`	position at previous bar line
`/box`	print inside a rectangular box
`/bu`	print below, at the underlay level
`/b<`*n*`>`	print <*n*> points below the stave
`/c`	centre the string horizontally
`/d<`*n*`>`	move down <*n*> points
`/e`	align end of string, not start
`/fb`	string is figured bass
`/h`	position halfway between notes
`/l<`*n*`>`	move left <*n*> points
`/m`	print below, halfway to the next stave
`/nc`	cancel a previous `/c`
`/ne`	cancel a previous `/e`
`/ol`	string is overlay
`/ps`	insert PostScript string in output
`/r<`*n*`>`	move right <*n*> points
`/ring`	print inside a ring shape
`/rot<`*n*`>`	rotate <*n*> degrees
`/s<`*n*`>`	print using font size <*n*> (1–12)
`/ts`	position at time signature

`/ul`	string is underlay
`/u<`*n*`>`	move up <*n*> points

Font sizes are defined in the heading by the **textsizes** directive.

For underlay and overlay strings, additional strings may follow as options, to control the printing of hyphens between syllables. See section 45.5 for details.

50.6 Character string escapes

These escape sequences are available in all character strings.

`\a'`	prints á
`` \a` ``	prints à
`\a^`	prints â
`\a.`	prints ä

The same accents are available for the other four vowels, in both upper and lower case forms.

`\c,`	prints ç – also available for ‘C’
`\c)`	prints © from the current font
`\c]`	prints © from the PostScript Symbol font
`\fi`	prints ﬁ
`\fl`	prints ﬂ
`\n~`	prints ñ – also available for ‘N’
`\p\`	page number
`\pe\`	page number, if even
`\po\`	page number, if odd
`\ss`	prints ß
`\?`	prints ¿
`\\`	prints \
`\'`	prints ′ (a plain ′ prints an closing quote/apostrophe)
`` \` ``	prints ` (a plain ` prints a opening quote)
`\--`	prints –
`\---`	prints —
`\@`	start of in-string comment; ends at next \

`\*b\`	breve
`\*s\`	semibreve
`\*m\`	minim
`\*c\`	crotchet
`\*Q\`	quaver
`\*q\`	semiquaver

Any of the above can include a dot after the note letter to print the dotted form of the note.

`\*#\`	sharp
`\*$\`	flat
`\*%\`	natural

`\*u\`	moves up by 0.2 times the music font’s size
`\*d\`	moves down by 0.2 times the music font’s size
`\*l\`	moves left by 0.33 times the music font’s size
`\*r\`	moves right by 0.55 times the music font’s size
`\*<\`	moves left by 0.1 times the music font’s size
`\*>\`	moves right by 0.1 times the music font’s size

Musical escapes with a single asterisk use a font at 9/10 that of the surrounding text. A double asterisk gets the full size.

`\t<`*x*`>`	transpose chord name <*x*> (one of A–G)

The note letter can be followed by # or `$`.

\<*n*>\	character number <*n*> from the current font
\*<*n*>\	character number <*n*> from the 9/10 music font
**<*n*>\	character number <*n*> from the full sized music font
\s<*n*>\	character number <*n*> from the Symbol font

\rm\	change to roman type
\it\	change to italic type
\bf\	change to bold face type
\bi\	change to bold-italic type
\sc\	change to a small caps font size
\sy\	change to the symbol font
\mu\	change to the music font at 9/10 size
\mf\	change to the music font at full size
\x1\	change to the first extra font
...	
\x12\	change to the twelfth extra font

Extra fonts are defined in the heading by the **textfont** directive.

50.7 Underlay strings

These characters are treated specially in underlay strings:

-	end of syllable in mid-word
=	continue syllable over additional note
#	print as space; doesn’t terminate a word
^	centre only characters to the left of this or between two ^ characters

50.8 Bracketed stave directives

These directives occur in square brackets interspersed in among the notes and rests:

[\!\]	repeated accent movement
[\.\]	repeated staccato
[\-\]	repeated accent
[\>\]	repeated horizontal wedge accent
[\v\]	repeated small vertical wedge accent
[\V\]	repeated large vertical wedge accent
[\'\]	repeates ‘start of bar’ accent
[\d\]	repeated string down bow
[\u\]	repeated string up bow
[\o\]	repeated harmonic ring
[\a<*n*>\]	repeated accent <*n*>
[\/\]	repeated single tremolo
[\//\]	repeated double tremolo
[\///\]	repeated triple tremolo
[\\]	no repeated marks
[<*n*>]	specify repeated input bars
[1st]	first time bar
[2nd]	second time bar
[3rd]	third time bar
[<*n*>th]	<*n*>th time bar
["<*text*>"/<*options*>]	rehearsal marking

[all]	end 1st/2nd time bars
[alto <*octave*>]	select alto clef
[assume <*setting*>]	assume key, time, or clef
[baritone]	select baritone clef
[barlinestyle <*n*>]	select bar line style for stave
[barnumber<*/movement*>]	explicit bar number print
[barnumber off]	suppress bar number print
[bass <*octave*>]	select bass clef
[beamacc]	next beam is an accelerando beam

[beammove <*n*>]	move next beam vertically
[beamrit]	next beam is a ritardando beam
[beamslope <*n*>]	force slope of next beam
[bottommargin <*n*>]	bottom margin for this page
[bowing <above|below>]	select bow marks position
[breakbarline]	break one bar line on one stave
[cbaritone]	select cbaritone clef
[comma]	comma pause
[contrabass <*octave*>]	select contrabass clef
[copyzero <*n*>]	move stave zero material
[couple up]	spread music to stave above
[couple down]	spread music to stave below
[couple off]	no coupling
[cue]	specify cue bar
[deepbass]	select deep bass clef
[dots <above|below>]	position of augmentation dots
[draw <*name*>]	obey a drawing definition
[el]	synonym for [endline]
[endcue]	end cue notes before bar end
[endline]	end line
[endslur]	end long slur
[endslur/=<*char*>]	end tagged long slur
[es]	synonym for [endslur]
[endstave]	end of this stave
[ensure <*n*>]	ensure space between notes
[fbfont <*name*>]	set default figured bass font
[fbtextsize <*n*>]	default size for figured bass text
[footnote "<*string*>"]	define footnote
[h]	synonym for [noteheads harmonic]
[hairpins above]	select hairpins position
[hairpins below]	select hairpins position
[hairpins middle]	select hairpins position
[hairpinwidth <*n*>]	set hairpinwidth for this stave
[hclef]	select percussion H-clef
[justify +<*edge*>]	add to justification edges
[justify -<*edge*>]	take away a justification edge
[key <*key signature*>]	set key signature
[line/<*options*>]	line above/below notes
[linegap/<*options*>]	leave gap in line
[mezzo <*octave*>]	select mezzo-soprano clef
[midichannel <*n*>]	change MIDI channel
[midipitch "<*name*>"]	change MIDI percussion pitch
[midivoice "<*name*>"]	change MIDI voice
[move <*n*>]	move next item horizontally
[move <*n*>,<*m*>]	ditto horizontally & vertically
[name "<*string*>" ...]	specify stave start text(s)
[name <*n*>]	select stave start text
[newline]	force new line of music
[newmovement <*option*>]	start new movement
[newpage]	force new page of music
[nocheck]	don't check this bar's length
[noclef]	select invisible treble clef
[nocount]	don't count this bar for numbering
[noteheads <*style*>]	select note head shape
[notes on]	turn on note printing
[notes off]	turn off note printing
[notespacing *<*n*>]	adjust note spacing
[notespacing <*n*> <*n*> ...]	adjust note spacing
[ns]	synonym for [notespacing]
[o]	synonym for [noteheads normal]
[octave <*n*>]	set transposition octave

[olevel <*n*>]	force overlay level
[olevel *]	revert to automatic overlay level
[olhere <*n*>]	adjust overlay level for this system
[oltextsize <*n*>]	set text size for overlay
[omitempty]	print nothing for empty bars
[overdraw ...]	as **[draw]** but done last
[overlayfont <*name*>]	set default overlay font
[page <*n*>]	increase page number to <*n*>
[page +<*n*>]	increase page number by <*n*>
[percussion]	specify percussion stave (old method) [stavelines] is now preferred
[playtranspose <*n*>]	change playing transposition
[playvolume <*n*>]	set relative playing volume
[printpitch <*note*>]	force printing pitch
[reset]	reset position to bar start
[resume]	resume printing stave
[rlevel <*n*>]	set rest level
[rmove ...]	as **[move]** but scale horizontally
[rsmove <*n*>]	as **[smove]** but scale horizontally
[rspace <*n*>]	as **[space]** but scale horizontally
[sghere <*n*>]	set system gap for this system
[sgnext <*n*>]	set system gap for next system
[skip <*n*>]	skip <*n*> bars
[slur/<*options*>]	start long slur
[slurgap/<*options*>]	leave gap in slur
[smove <*n*>]	combined [move] & [space]
[soprabass <*octave*>]	select soprabass clef
[soprano <*octave*>]	select soprano clef
[space <*n*>]	insert space before next note
[sshere <*n*>]	set stave spacing for this system
[ssnext <*n*>]	set stave spacing for next system
[stave <*n*> "<*string*>" ...]	start new stave
[stavelines <*n*>]	set number of stave lines
[stemlength <*n*>]	set default stemlengh adjustment
[stems <*direction*>]	force/unforce stem direction
[suspend]	suspend stave at next system
[tenor <*octave*>]	select tenor clef
[text <*name*>]	select default text type
[textfont <*name*>]	set default text font
[textsize <*n*>]	set default text size
[tick]	tick pause
[ties <*direction*>]	force/unforce tie direction
[time <*time signature*>]	set time signature
[time <*sig1*> -> <*sig2*>]	scale to other signature
[topmargin <*n*>]	set top margin for current page
[transpose <*n*>]	set transposition
[transposedacc force]	print cautionary accidentals
[transposedacc noforce]	don’t print cautionary accidentals
[treble <*octave*>]	select treble clef
[trebledescant <*octave*>]	select trebledescant clef
[trebletenor <*octave*>]	select trebletenor clef
[trebletenorB <*octave*>]	select trebletenorB clef
[tremolo]	print tremolo between notes
[triplets off]	don’t print triplet indications
[triplets on]	print triplet indications
[triplets <*options*>]	control triplet printing
[ulevel <*n*>]	force underlay level
[ulevel *]	revert to automatic underlay level
[ulhere <*n*>]	adjust underlay level for this system
[ultextsize <*n*>]	set text size for underlay
[unbreakbarline]	join one barline to next stave

`[underlayfont <`*name*`>]`	set default underlay font
`[x]`	synonym for `[noteheads cross]`
`[xline]`	crossing line
`[xslur <`*args*`>]`	crossing slur
`[z]`	synonym for `[noteheads none]`

50.9 Slur options

`/=<`*char*`>`	specify tagged slur
`/a`	slur above (default)
`/a<`*n*`>`	above, at fixed position
`/ao`	above, at overlay level
`/b`	slur below
`/b<`*n*`>`	below, at fixed position
`/bu`	below, at underlay level
`/e`	editorial (crossed) slur
`/h`	force horizontal slur
`/i`	intermittent (dashed) slur
`/ip`	intermittent point (dotted) slur
`/<`*n*`>`	following apply only to section <*n*>
`/ll<`*n*`>`	move the left end left by <*n*> points
`/lr<`*n*`>`	move the left end right by <*n*> points
`/rl<`*n*`>`	move the right end left by <*n*> points
`/rr<`*n*`>`	move the right end right by <*n*> points
`/u<`*n*`>`	raise the entire slur by <*n*> points
`/d<`*n*`>`	lower the entire slur by <*n*> points
`/lu<`*n*`>`	raise the left end by <*n*> points
`/ld<`*n*`>`	lower the left end by <*n*> points
`/ru<`*n*`>`	raise the right end by <*n*> points
`/rd<`*n*`>`	lower the right end by <*n*> points
`/ci<`*n*`>`	move the centre in by <*n*> points
`/co<`*n*`>`	move the centre out by <*n*> points
`/clu<`*n*`>`	move left control point up <*n*> points
`/cld<`*n*`>`	move left control point down <*n*> points
`/cll<`*n*`>`	move left control point left <*n*> points
`/clr<`*n*`>`	move left control point right <*n*> points
`/cru<`*n*`>`	move right control point up <*n*> points
`/crd<`*n*`>`	move right control point down <*n*> points
`/crl<`*n*`>`	move right control point left <*n*> points
`/crr<`*n*`>`	move right control point right <*n*> points

Most of the options for slurs also apply to lines over groups of notes, as they are just a different kind of ‘slur’ to PMW. The options for moving the Bezier curve control points are not relevant to lines, but `/co` and `/ci` have the effect of changing the length of the ‘jogs’. In addition, lines can take the following options:

`/ol`	requests that the line be ‘open on the left’
`/or`	requests that the line be ‘open on the right’

50.10 Default values

Bar length check	enabled
Bar lines	solid through system
Beam flag length	5
Beam thickness	1.8
Bottom margin	0
Bracket/brace	bracket whole system
Breve rests	not used
Caesurae	two strokes
Clef	treble
Clef size	1.0

Dot space factor	1.2
Figured base size	10 points
First page number	1
Font family	Times
Footnote separation	4 points
Grace size	7 points
Grace spacing	6 points
Hairpin line width	0.2 points
Hairpin width	7 points
Heading type sizes	
first heading	17, 12, 10, 8
movement heading	12, 10, 8
Hyphen string	one hyphen character
Hyphen threshold	50 points
Justify	top bottom left right
Key	C major
Key warnings	enabled
Left margin	computed for centring
Line length	480
Long rest font size	10
Magnification	1.0
Maximum number of bars	500
Note spacing	30 30 22 16 12 10 10 10
Note stem direction	automatically chosen
Note style	with stems
Overlay depth	11 points
Overlay size	10 points
Page length	720
Repeat bar font size	10
Repeat style	standard
Sheet depth	900 points
Sheet size	A4
Sheet width	608 points
Small cap size	0.7
Stave spacing	44 points
Stave style	five-line
System gap	44 points
Text size	10 points
Time signature	4/4
Time signatures	printed
Time siguature warnings	enabled
Top margin	10
Trill string	*tr*
Triplet font	roman
Triplet size	10 points
Transposition	none
Underlay depth	11 points
Underlay size	10 points

Index