# Philip's Music Writer (PMW)

## A Music Typesetting Program

**Philip Hazel**

**Philip's Music Writer (PMW)**

Author: Philip Hazel



Revision 4.23 23 January 2012

# 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. 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. Although a textual input method may not be considered as 'user-friendly' as pointing and dragging on the screen, it can 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.

The output of PMW is a PostScript file. This can be previewed on screen using *GhostScript* or similar software. If you have a printer that understands PostScript, PMW output can be printed directly; otherwise, conversion software such as *GhostScript* is required. *GhostScript* can also be used to convert PostScript files into Portable Document Format (PDF) files. PMW can be requested to produce its output as encapsulated PostScript (see the **-eps** option). This is useful if the music is an illustration that will subsequently be included in another document.

This edition of the manual describes PMW version 4.23. Version 4.00 was the first version for Unix and Unix-like environments. Earlier versions were called *Philip's Music Scribe*, and were run on Acorn's RISC OS operating system in the 1990s. From version 4.10 onwards, PMW interprets text strings as UTF-8 encoded Unicode, giving access to all the available characters in the standard PostScript text fonts. Access to non-ASCII characters is also available using escape sequences. Section 6.14.2 discusses the issues of backwards compatibility with previously-existing PMW input files.

As well as PostScript output, PMW can also write a MIDI file that can be played through the computer's sound system by an application such as *Timidity*. MIDI output is not very sophisticated, and is intended for 'proof-hearing' purposes rather than for performance.

PMW comes with a PostScript font called PMW-Music. This contains all the music shapes (notes, rests, accidentals, bar lines, clefs, etc.) that PMW requires. I acknowledge with gratitude the help of Richard Hallas, who created the original versions of some of the characters in this font and improved many others. The half sharp and half flat characters were contributed by Hosam Adeeb Nashed. Richard also contributed a second font called PMW-Alpha. It contains additional characters that may be useful when printing music (see chapter 13).

The PMW input encoding is designed to be easy for a musician to remember. It makes use of as many familiar music notations as possible within the limitations of the computer's character set. Normally it is 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.

This introduction ends with a short summary of the music and other terminology used in this manual. The following two chapters describe how PMW should be installed and operated. Chapters 4 and 5 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 6 onwards) is reference material; the information in earlier chapters is repeated, with additional specialist information. Finally, there are chapters giving details of text fonts, the PMW music fonts, summaries of the syntax of input files, and an index. Many cross-references are given in a shortened form using a pointing hand symbol, for example, ☞ 3.2.

## 1.1 Terminology

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

The word ‘directive’ is used as the name for instructions in the input file that tell PMW what to do. There are directives that control the overall form of the piece, and others that operate within individual staves.

The word ‘argument’ refers to a data value that is provided on the command line for running PMW, or is coded as part of a directive. For example, the directive to set the page length has to be followed by one number; this is its argument (the usage is taken from mathematics and computer programming).

The word ‘parameter’ refers to a data value that controls the format of the typeset music. For example, there is a parameter whose value is the width of lines of music. All parameters have default values, but these can usually be changed by an appropriate directive.

Some formal music 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 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 horizontal lines on which notes are printed. The normal stave contains five lines, but other numbers of lines are sometimes used, for example, a single line for percussion parts. In this document, the word ‘stave’ is used as the singular of ‘staves’. However, the program itself accepts ‘staff’ as a synonym of ‘stave’ under all circumstances.

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.

# 2. Installing PMW

PMW is developed on a Linux system, but as it is a straightforward C program without any kind of fancy interface, it should run without problems in any Unix-like environment. This includes the Cygwin environment under Microsoft Windows. The author of PMW has no Windows experience, but Neil Killeen, a PMW user, has kindly provided notes on running PMW under Windows. These may be found in the PMW distribution tarball in the file *doc/Cygwin.txt*.

The reader is assumed to be familiar with using shell commands in Unix-like environments. PMW is installed from source in the same way as many other applications. First, download the tarball from the web site **http://www.quercite.com/pmw.html** into a suitable directory. You should end up with a file such as *pmw-4.23.tar.gz*. Uncompress the file with *gunzip* and then unpack the archive:

```
gunzip pmw-4.23.tar.gz
tar -xf pmw-4.23.tar
```

This creates a directory called *pmw-4.23*, 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, make the source directory current, and then issue the usual *configure* and *make* commands:

```
cd pmw-4.23
./configure
make
make check
make install
```

You may need to be *root* to run the installation command. 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 the 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 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 set of PostScript fonts (such as *Times-Roman*) and the PMW music fonts.

Once you have installed PMW, you can use the *pmw* command to generate PostScript from input files, as described in chapter 3 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. You can either cause the fonts to be included in every output file, or configure your printer or viewer so that it knows where to find them. Exactly what you have to do varies between systems. I hope the following instructions will give enough hints to cover most cases.

## 2.1 Including the music fonts in the output file

If you use the **-includefont** option on the *pmw* command line, or put it in your *.pmwrc* file, PMW includes the music fonts in every output file that needs them. This means that the output files are freestanding PostScript files that should be printable or viewable without any special action. However,

the output files are larger by about 30–40K for each of the two fonts. If you do not mind this overhead, this is the easiest approach to take.

## 2.2 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. You can check your version of *GhostScript* by displaying the expected output from the PMW 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. The test files were all processed using PMW's **-includefont** command line option, so the output for each contains a copy of the relevant PMW music fonts.

If you do not use the **-includefont** command line option when processing your own input files, *GhostScript* needs to be told where the PostScript music fonts are before it can display a PMW output file. One easy way of doing this is to set the `GS_FONTPATH` environment variable, for example:

```
export GS_FONTPATH=/usr/local/share/pmw/psfonts
```

However, this may not work when you try to print the music because the setting may not be carried over into the printing environment. An alternative is to install symbolic links from a suitable font directory to PMW's *psfonts* directory. This should then also work for printing. You can find out which directories *GhostScript* searches for its fonts by running the following command:

```
gs -h
```

In many systems */usr/share/fonts/default/ghostscript* is an appropriate directory in which to insert the links, so you might use these commands:

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

In addition, for some older versions of *GhostScript*, you might also need to add this line to the file called *Fontmap.GS* 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*. Newer versions of *GhostScript* do not seem to need this, so first of all, try without.

Here is a hint for when you are creating your own PMW output: 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 the page it was previously displaying.

## 2.3 Problems with displaying staves and bar lines

By default, staves and bar lines are output using characters from PMW's music font. Some PostScript interpreters do not display these correctly on the screen, and sometimes there are also printing problems. To help with these issues, the way PMW works can be modified by command line options (see chapter 3). If your output does not show staves or bar lines correctly, experiment with these options to see if they can resolve the issue. Note that default option settings can be put in your *.pmwrc* file.

### 2.3.1 Missing staves

Staves are normally output using two characters that are 100 points and 10 points wide, respectively, at the default magnification. Some PostScript interpreters cannot handle characters as wide as 100

points, and either display nothing, or give errors. The **-nowidechars** option suppresses the use of the wide characters.

### 2.3.2 Gaps in staves

Sometimes PMW output is displayed with gaps in the staves, even when **-nowidechars** is used to suppress the use of wide stave characters. This is sometimes just a problem with a screen display; the same file often prints correctly. If the option **-drawstavelines** is used, staves are output as PostScript drawing commands instead of as characters. This option overrides **-nowidechars**.

### 2.3.3 Gaps in bar lines

Sometimes PMW output is displayed with gaps in bar lines that extend over several staves. This is sometimes just a problem with a screen display; the same file often prints correctly. If the option **-drawbarlines** is used, bar lines are output as PostScript drawing commands instead of as characters.

## 2.4 Antialiasing and the screen display

When it is interpreting PostScript for display on the screen, *GhostScript* can be run with or without antialiasing, 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 PMW-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 antialiasing would not be needed even if it were possible (which it is not on black-and-white printers).

## 2.5 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 music 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 a Type 3 PostScript font.

## 2.6 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. In many Unix-like systems the CUPS printing system is set up to do this automatically, so if you are using CUPS it may ‘just work’.

## 2.7 Printing PMW output on a PostScript printer

Unless you use the **-includefont** command line option, 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 occur.

If you have full control of a PostScript 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 in new PostScript printers. 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 a 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 may generate a warning message. This is perfectly normal and can be ignored. The message is usually something like this:

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

If you do not have full control over the printer, or do not want to load the fonts permanently, you should always use the **-includefont** option on the PMW command line, or put it in your *.pmwrc* file, so that the music fonts are included in the PMW output.

## 2.8 Uninstalling PMW

If you want to uninstall PMW, you can use the command:

```
make uninstall
```

This removes the files that were installed. It does not remove directories.

# 3. Running PMW

The PMW command has the following form:

pmw [*options*] [*input file*]

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 quartet.pmw
pmw -f viola -o quartet.ps -midi /tmp/quartet.mid quartet.pmw
```

The command line options are as follows:

**-a4ona3**
There are several directives that control the size of the page images PMW produces (☞ 6.5). In the common case, this size is identical to the size of paper that is being used, in which case one image fits exactly onto 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.

**-drawbarlines**
Bar lines are normally output using characters from the music font. This option causes them to be output using PostScript drawing commands. It may produce better output in environments where some PostScript interpreters leave little gaps in bar lines that extend over more than one stave. **-dbl** is a synonym for **-drawbarlines**.

**-drawstavelines**
Staves are normally output using characters from the music font. This option causes them to be output as individual lines, using PostScript drawing commands. It may produce better output in environments where some PostScript interpreters leave gaps in staves. **-dsl** is a synonym for **-drawstavelines**, and this option overrides **-nowidechars**.

**-duplex**
Set the ‘duplex’ option in the generated PostScript output. This should be honoured by PostScript printers that can print on both sides of the paper (see also **-tumble**).

**-eps**
Write the output as encapsulated PostScript. This is useful if the music is an illustration that will subsequently be included in another document. See section 3.5 for details of how this option affects the processing of included PostScript files. For one-off illustrations, combining **-eps** with **-includefont** is useful, so that the Music font is automatically included. For a document with many musical illustrations, including the font in each one is undesirable; it is better to make it available in some other way.

One PMW user reported problems with EPS files when other special fonts were also required. The solution was to pass all the fonts and the EPS file into the open-source Scribus desktop publishing program, convert to Bézier curves, then re-export as EPS. This removes font references, and produces a file that can easily be embedded in any DTP program.

**-F** <*directory*>
Search the given directory for *fontmetrics* files, before searching the default directory that was set up when PMW was built. This option is useful when you want to make use of a non-standard font in text strings without having to copy its fontmetrics file into the default directory. If a relative file name is given, it is taken as relative to the current directory, not to the PMW input file's directory.

**-f** <*name*>
This option specifies a format name, which is useful when the input file is set up to generate output in several different formats. The format name can be tested by the ***if** directive and used to vary the output. 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. See chapter 5 for details of how to set up the input so as to output different headings and so forth when different stave selections or formats are requested.

**-H** <*file*>
Use the given file as the PostScript header file, instead of the default that was set up when PMW was built. If a relative file name is given, it is taken as relative to the current directory, not to the PMW input file's directory. This option is unlikely to be of general use, but is helpful when testing new versions of the header file. See section 3.5 for details of how included PostScript files are processed.

**-help** or **--help**
Output a list of command line options, then stop. No file is read.

**-includefont**
This option causes PMW to include the music font within the PostScript output that it generates. If the PMW-Alpha font is used, that is also included. If you use this option, there is no need to install the font(s) for *GhostScript* (or any other display program), nor do you need to download them separately to a PostScript printer. However, it does mean that each PMW output file is bigger by 37K for PMW-Music and 31K for PMW-Alpha.

**-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. See section 6.6 for more about MIDI output.

**-midibars** <*start*>**-**<*end*>
Limit 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 the 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.

**-MF** <*directory*>
Search the given directory for the PostScript music font files, before searching the default directory that was set up when PMW was built. This option is mainly of use when testing PMW and new versions of the fonts.

**-MP** <*file*>
Use the given file as the *MIDIperc* file, instead of the default that was set up when PMW was built. If a relative file name is given, it is taken as relative to the current directory, not to the PMW input file's directory. 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. If a relative file name is given, it is taken as relative to the current directory, not to the PMW input file's directory. 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
```

**-norc**
If this option is used, it must be the very first option that follows the *pmw* command name. It causes PMW not to read the user's *.pmwrc* file (☞ 3.2).

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

**-nowidechars**
This option stops PMW from using wide stave characters when printing staves. It is provided because it seems that some PostScript interpreters cannot deal correctly with characters whose width is 100 points at the default magnification (compared with 10 points for the narrow versions). A 310-point 5-line stave is normally printed using the string FFFC. (The code numbers of wide and narrow 5-line stave characters in the music font correspond to F and C in text fonts.) With **-nowidechars**, the same stave is printed as CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC. **-nw** is an abbreviation for **-nowidechars**. The **-drawstavelines** option overrides this option.

**-o** <*file*>
Write 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.

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

This 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 by **-a4ona3** or **-a5ona4**. 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 (except when right-to-left printing is enabled (☞ 8.1.108), but that is unusual). The resulting pages, if printed two-sided, can be stacked together and folded in the middle to form a ‘pamphlet’.

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

Outputting all pages at once on a single-sided printer is useful for producing master copies for reproduction elsewhere. If you want to produce a final two-sided copy directly, 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. On a duplex (two-sided) printer, you may need to set the **-tumble** option to get all the pages the right way up.

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 normal two-up printing, PMW centres the page images in the half pages in which they appear, but 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, and negative values move it in the opposite directions.

**-printgutter** <*x*>
This option specifies a distance by which righthand (recto) page images are moved to the right, and lefthand (verso) page images are moved to the left, thus creating a ‘gutter’ for binding when the images are printed doublesided. The **-printgutter** setting applies only when pages are being printed 1-up. It is ignored for any of the 2-up printing styles.

**-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 that 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 pairs of numbers separated by a hyphen, to specify a range. Use commas to separate items in the list.

```
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. See chapter 5 for details of how to set up the input so as to output different headings and so forth for different stave selections.

**-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 section 6.10.

**-tumble**
When **-duplex** is set, **-tumble** sets the PostScript option for 'tumbled' duplex printing.

**-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 (☞ 3.3).

## 3.1 Debugging options

The following options are of interest only to a PMW maintainer. They are listed here for completeness, but no details are included.

**-debug**
Write general debugging information to the standard error file.

**-dsb** <*m*>,<*s*>,<*b*>
Write internal debugging data for the contents of bar <*b*> (an absolute bar number) on stave <*s*> in movement <*m*> to the standard error file. If only one number is given, it is taken as a bar number in stave 1 of the first movement; if only two numbers are given, they are taken as a stave and bar number in the first movement.

**-dtp** <*n*>
During formatting, write internal positioning data for bar <*n*> (an absolute bar number) in any movement (there is usually only one when debugging at this level) to the standard error file. Sometimes a bar may be formatted more than once; there will be output each time. If the number is given as -1, positioning data is output for all bars.

## 3.2 Setting default command-line options

There is a simple facility for specifying options that you always want to be set. When PMW starts up, it looks in the user's home directory for a file called *.pmwrc*. If this file exists, its contents are read and used to modify the PMW command line. White space (spaces, tabs, or newlines) in the file are used to separate items. Each item is added to the command line, before the given arguments. Thus, for example, if you always want to make use of the **-nowidechars** option, all you need to do is to create a *.pmwrc* file that contains:

```
-nowidechars
```

The effect of this is the same as if you type `-nowidechars` immediately after `pmw` every time you run it. If you insert an option that requires data, the data item must also be given in the *.pmwrc* file, otherwise an error occurs. For example, if you always want to create MIDI output and write it to a fixed file name, the file might contain:

```
-midi /usr/tmp/pmw.midi
```

Note that PMW does not allow options to be repeated, so if an option is present in the *.pmwrc* file, it cannot also be given on the command line. There is no way to override individual options that are set in the *.pmwrc* file. However, if the first option on the command line is **-norc**, the *.pmwrc* file is not used.

## 3.3 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 that 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 in 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 parentheses 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 parentheses, 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.

## 3.4 PMW input errors

When PMW detects an error in the input file, it writes a 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 do not 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
** File "K495.pmw", near line 17:
rrf'-g |
       <
```

In this case a minus sign (indicating a quaver) has 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 that do not stop the output from being written. An example is the detection of a bar that is too wide for the page; PMW diagnoses this, and then squashes it to fit. The messages for all these errors start with the word 'warning'.

## 3.5 PostScript inclusions

The output of PMW starts with an included PostScript header file. There are also some directives (for example, **psheading**) that allow you to include custom PostScript code at various points in the output. These features are not normally required, but are provided for those who are familiar with PostScript and who want to do things that PMW cannot normally do.

All PostScript files that are included by PMW are treated in the same way. Lines that start with %EPS followed by a space are included, with the first five characters removed, only when the output is an encapsulated PostScript file (see the **-eps** option above). For non-EPS output, such lines are omitted. Any other line that starts with a single percent sign is omitted, as are blank lines. However, lines that start with two percent characters are copied to the output.

# 4. Getting started with PMW encoding

In this and the next chapter we 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 of this manual, which starts at chapter 6.

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

```
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]
```

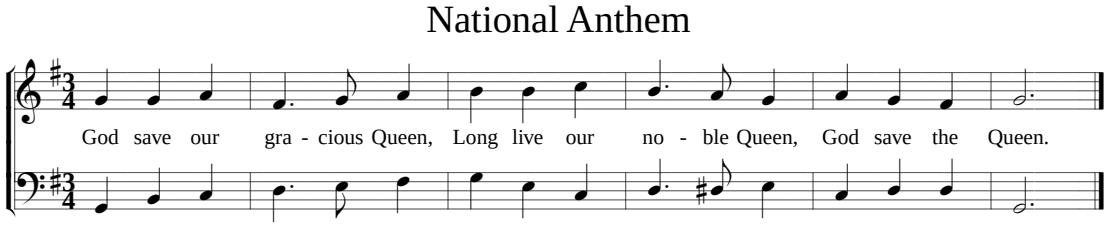
You may use any name you like for the file, and put it in any convenient directory. Let's suppose it's called *natanth*. Process the file with PMW by running this command:

```
pmw -includefont 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 running:

```
gv natanth.ps
```

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




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 vertical bar characters is a common mistake, leading to an overlong musical bar.

We will now 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.

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

The first directive provides a text heading for the piece. 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). In other words, the entire title is centred.

```
breakbarlines
```

The second directive 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 scores 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:

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

This breaks the bar lines after staves 4, 8, and 12.

```
underlaysize 9.5
```

The third directive sets the font size for the underlay text (the sung 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.

**Note**: The music above and in all the following examples in this manual is shown at 0.85 times its normal size, so the type sizes you see here are smaller than they will be if you print the example yourself.

```
notespacing *1.1
```

The fourth directive 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 a factor of between 1.1 and 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.

```
key G
```

The fifth directive 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 that resemble flats or naturals, so instead the two keys that are next to # on some keyboards were chosen: `$` for a flat (think 'Dollar' for 'Down') and `%` for a natural. For example, the key signatures C sharp minor and G flat major are coded as `C#m` and `G$` respectively.

```
time 3/4
```

The sixth directive 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 `C` and `¢` respectively.

The heading ends and the stave data begins with the first line that starts with a square bracket:

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

You will notice that a bit further down there is a line containing just `[endstave]`. This marks the end of the data for the first stave. Each stave's data is always contained between **[stave]** and **[endstave]**.

The data itself consists of a mixture of encoded music, words, bar lines, and so on, and also *stave directives*. To make it clear what is what, the stave directives are enclosed in square brackets, and they are shown in brackets whenever they are mentioned in this manual. 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 in a system. 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 that 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.

The remaining stave directive, `text underlay`, sets the default type for any text strings 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 be 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’, but PMW supplies as many printed hyphens as necessary to fill the space between them when they are printed. Text strings are not restricted to just the characters on the computer keyboard; see section 6.14.3 for details of how to access other characters.

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 (though it is possible to turn this check off). The notes 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, and 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 by adding a full stop, as in the second, fourth, and last bars. Turn 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. PMW also has some basic support for half accidentals (☞ 9.6.2). 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 like this:

```
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. 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 next chapter 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. However, before doing that we introduce a general feature that can be used to simplify and customise PMW input files.

## 4.1 Simple 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 document, the sequence is given a name. Referring to the name later in the input calls up the required characters. There are several advantages in using a macro for a repeated character sequence. 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 that 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. White space that immediately follows the macro name is not included.

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 ***mf***, where the *m* is printed in italic and the ***f*** in bold italic, as is commonly done. (See sections 5.3.2 and 6.14 for explanations of how the above string achieves this.) The example 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, because 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 |
```

The option `/d6` moves the text down by six points. 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 section 6.2.

# 5. Using other PMW features

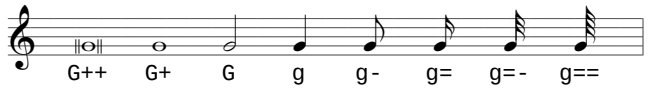
In this chapter we cover most of the major PMW facilities in an introductory manner. All the information is repeated in more detail in the reference chapters that follow.

## 5.1 More about notes

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

### 5.1.1 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:




### 5.1.2 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.

### 5.1.3 Repeated rest bars

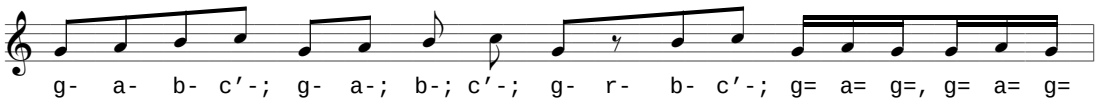
A whole bar rest can be repeated any number of times by putting a number in square brackets before the rest. For example, the code for 24 bars’ rest is:

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

In fact, this kind of repetition is not confined to rest bars; it can be used to repeat any one bar.

### 5.1.4 Beams

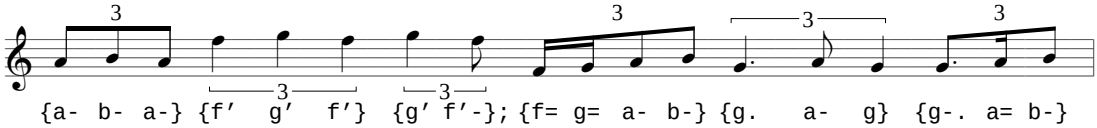
Notes that are shorter than a crotchet are automatically beamed together within a bar, unless they are separated by a beam breaking character. A semicolon breaks the beaming completely, and a comma breaks all but the outermost beam. Beams carry on across rests that are shorter than a crotchet, but they are always broken at the end of a bar, unless a continuation over the bar line is explicitly requested (☞ 9.7.)



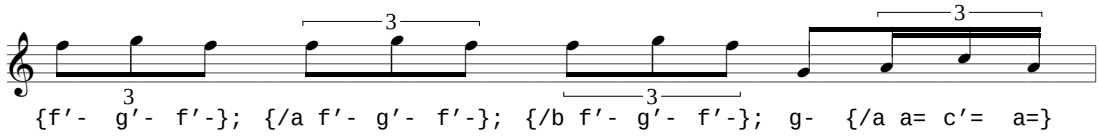

There must not be any space between the last note of a beam and the breaking character (semicolon or comma), but there can be spaces (and other items of data) between the notes themselves.

### 5.1.5 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:



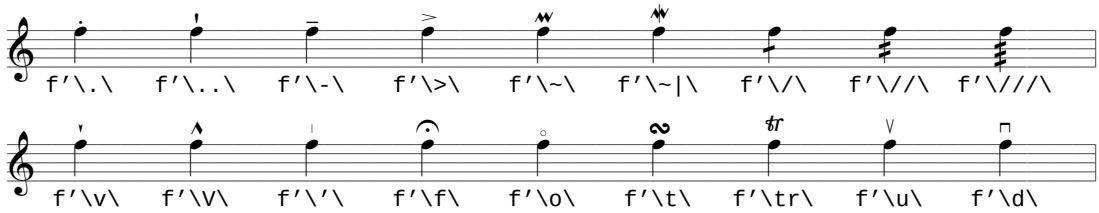
You can change the way triplets are printed by putting options after the opening curly bracket. If you put `/a` the '3' will be put above the notes, whereas `/b` forces it below the notes. In both cases the longways bracket is also drawn.




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 mark (☞ 8.1.137, 8.1.138, 9.6.26, 10.2.113).

### 5.1.6 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:

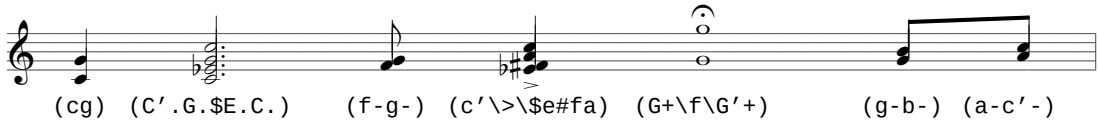



The codes `\v\` and `\V\` are used for small and large 'vertical wedge' accents because of the similarity of shape, though the accents themselves may be the opposite way up to the coding letters, depending on whether they appear above or below the note. The other mark that looks like the letter V is a string 'up bow' mark, 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 9.6.16 onwards.

### 5.1.7 Chords

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




## 5.2 Bar lengths and bar numbers

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**. The length check can also be disabled for an individual bar. 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.

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

If there is more than one stave, PMW checks that each one has the same note length in each bar. If not, it generates a warning message for each mismatch. Output is still generated; the short bars are filled with blank space. However, this check does not apply to whole bar rests that are notated as `R!`, `Q!`, or `S!`, unless the bar ends with something else (such as a clef change) that depends on the bar length. This exception is made so that the whole bar rest notation can conveniently be used in situations where the bars in the other staves are of varying lengths. However, if a MIDI file is generated, whole bar rests always take up the length of time implied by the time signature.

### 5.2.1 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 that control where these appear and in what form.

```
barnumbers line
```

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

```
barnumbers 10
```

This 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.

```
barnumbers boxed line 9
```

This 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:

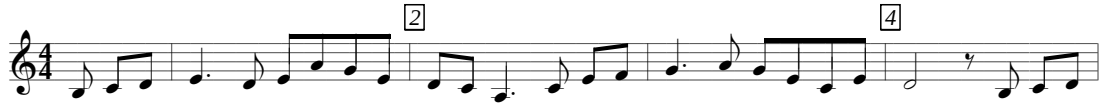
```
barnumbers 5 italic
```

This requests bar numbers in italic, every five bars, using the default font size of 10 points. There are further options for forcing or suppressing individual bar numbers, and for moving them (☞ 10.2.7).

### 5.2.2 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. Section 6.3 explains how PMW identifies uncounted bars if it needs to refer to them, for example, in an error message. The following example shows the use of the **barnumbers**, **[nocheck]**, and **[nocount]** directives:

```
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
```



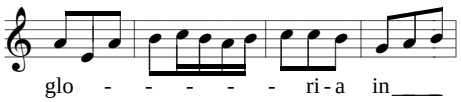

## 5.3 More about underlay (lyrics)

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.

### 5.3.1 Multi-note syllables

In the National Anthem example in chapter 4, 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.

```
"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 follow 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 and is printed underneath.

### 5.3.2 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 these issues, PMW treats the backslash character specially if it is found in a quoted string. (This 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. There are a number of ‘escape sequences’ that 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 |

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*. There is an in-depth discussion of text fonts and character encodings in section 6.14.1. Section 6.14.3 has more details about escape sequences, and there is a list of available text characters and their escape sequences in chapter 11.

### 5.3.3 Spacing

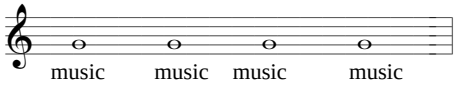
Within a bar, PMW ensures that the syllables of the underlay text do not crash into each other, by spreading out the notes if necessary. **Warning:** If use of the **layout** heading directive (☞ 8.1.57) causes the bars in a system to be horizontally compressed in order to fit them on the line, underlaid syllables may be forced into each other. It’s best to avoid settings of **layout** that cause compression if possible.

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.

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

This coding ensures that the two notes are 7 points (about 0.1 of an inch) 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. Firstly, 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, and putting them at the end moves it left. Secondly, 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. The # and ^ characters are treated specially only in underlay (and overlay) strings. This example shows how the use of # and ^ affects the positioning of syllables:

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

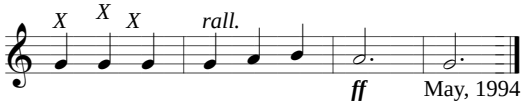



## 5.4 Other kinds of text

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 not underlay, and that it is to be printed above or below the stave, respectively. If **[text underlay]** has not been set for the stave, 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 following 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 |
```




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

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

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

♩ = 45

A rehearsal mark is a special kind of string that is coded by placing it in square brackets:

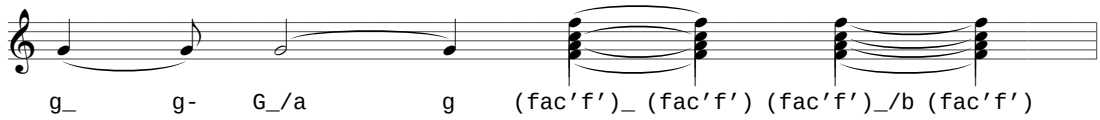
```
["A"]
```

PMW prints such strings in a fairly large font, enclosed in a rectangular box; there are options to change this if necessary (☞ 9.11).

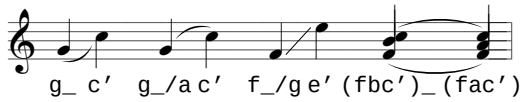
## 5.5 Ties, slurs, and glissandos

Single notes and chords are tied together by entering an underscore character at the end of the first note, or following the closing parenthesis 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.

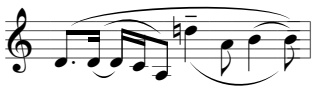



When two single notes of different pitches are connected by a slur, the same notation (an underscore) can be used. However, for chords, the **[slur]** directive (see below) is required to draw slurs, because if two chords are joined by an underscore, the notes in each that are of the same pitch are joined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

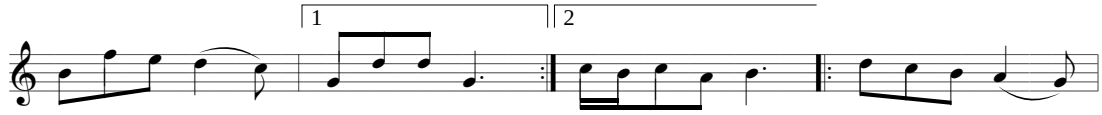


This example shows that slurs can be 'nested' inside one another if necessary, each **[endslur]** directive relating to the most recent **[slur]**. There are options for handling more complicated cases, and there are also options for adjusting the positions and shapes of slurs (☞ 10.2.76).

## 5.6 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:

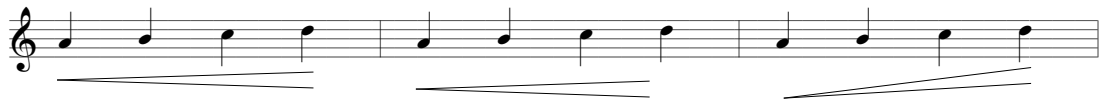
`"\mf\c"` prints 𝄋
`"\mf\d"` prints 𝄌

The escape sequence `\mf\` changes to the music font, full details of which are given in chapter 12.

## 5.7 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.

```
< abc'd' <  |  </d4 abc'd' </l10  |  </r4/d8 abc'd' </u10
```



If the beginning or ending character is followed by `/h`, 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 also some other options for changing the position and form of hairpins (☞ 9.5).

## 5.8 Staves and systems

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

### 5.8.1 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.

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

This example 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 unusually 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.

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

### 5.8.2 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]**.

### 5.8.3 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**, that specify which staves are to be joined with each of these signs. Each of these directives is followed by a list of stave ranges.

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

This example 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, whereas 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.

### 5.8.4 Initial text

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

```
[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.

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

Options are available for changing the form and layout of this text (☞ 10.2.90).

## 5.9 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.

### 5.9.1 Overprinted staves

There are two ways of tackling pieces that have two parts on one stave, with stems pointing in different directions. If most of the piece is like this, 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.

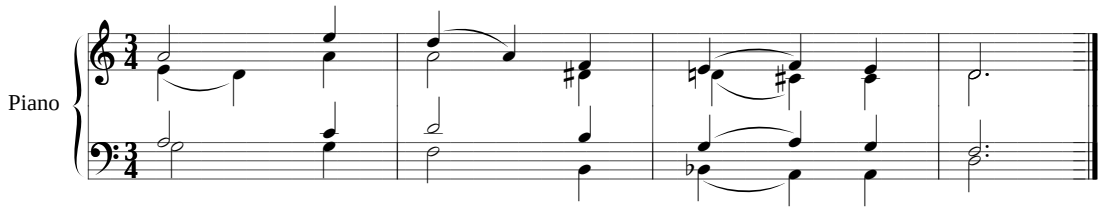
```
time 3/4
bracket
brace 1-4
stavespacing 1/0 2/48 3/0

[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]
```




There are two items in this example that 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 sufficient to get it clear.

### 5.9.2 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.

### 5.9.3 Invisible rests

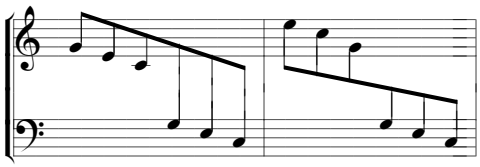
When using overprinting staves for keyboard pieces, it is frequently the case that one ‘part’ does not contain enough notes to fill the bar. The note letter Q can be used to make the bar up to its correct length. This letter (which can be thought of as standing for ‘quiet’) acts exactly like the rest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

### 5.9.4 Coupled staves

Keyboard music sometimes includes sets of beamed notes that 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.

**Warning:** Coupling does not work properly unless the upper stave is using the treble clef and the lower one is using the bass clef, and the distance between them is a multiple of four points.

The second bar of this example shows how to get notes printed on both sides of a beam, a facility that 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 is not enough space 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.

## 5.10 Heads and feet

We introduced the **heading** directive in the discussion of the National Anthem example in chapter 4. You may have as many **heading** directives as you like at the start of a piece. By default, the first two are 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"
```

This example specifies a size of 13 points. 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
```

This example 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.

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

In this example, the first heading consists of centred text, and the second has only left-hand and right-hand parts, with nothing in the middle, so they do not overlap.

The **footing** directive is of exactly the same form as **heading**; it specifies text that 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, and **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
```

This is a typical example:

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

It causes the page number to be printed centrally at the foot of each page (unless there is also a **footing** directive, for printing something different on the first page).

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

This example 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 music 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 12. 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 (for example, 3.3 points for a 10-point font). Consider this directive:

```
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.

```
pageheading "" -10
```

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

## 5.11 Page layout

The horizontal length of music 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:

```
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, as well as some other standard sizes, and 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 music 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 the layout that is required in advance, 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.

```
notespacing *0.92
```

This example 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 changes in the assignment of bars to systems. 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]** stave 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 section.

## 5.12 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.

```
magnification 1.5
magnification 0.75
```

The first example has the effect of increasing the gap between stave lines to 6 points, whereas the second reduces it to 3 points. There is also a directive called **stavesize** (☞ 8.1.117) that can be used to alter the magnification for individual staves. There are no restrictions on the values that can be given for the magnification.

When a magnification is specified, everything that is printed is magnified (or reduced) in proportion, and 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 music data. However, exceptionally, 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.

## 5.13 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 3. 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 PMW's *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 something like this:

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

Because they are preprocessing directives, each ***if**, ***else**, or ***fi** must appear 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 by the **-s** command line option, 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. These conditional preprocessing directives can be used 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.

# 6. PMW reference description

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 that have not yet been covered. The remainder of this document is written in the form of a reference manual. It contains a complete description of PMW input files, repeating in more detail some of what has gone before. 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 bracketed italic words 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. This is an example of the use of **tripletfont**:

```
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 (☞ 6.13).

The following sections describe the format of PMW input files, and then discuss a number of general features, with references to particular directives. Complete descriptions of the directives themselves are not given here; they may be found in *Heading directives* and *Stave directives* (chapters 8 and 10). The chapter in between, *Stave data* (chapter 9), contains the specification of all items other than directives that may appear as part of a stave's data.

## 6.1 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. The input is in free format. Outside quoted strings, there is only one circumstance in which the use of white space is necessary, and that is to delimit an item 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 this:

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

### 6.1.1 Line breaks

Line breaks in a PMW input file 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 (☞ 6.2) always take up a complete line of their own, 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, to indicate that another number follows.

### 6.1.2 Macro insertion

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 6.2.2. If a literal & character is actually required in the input, it must be entered as &&.

### 6.1.3 Case sensitivity

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

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

### 6.1.4 Heading information

A PMW file starts off with a number of items collectively known as the *heading*. These provide information that applies to the whole piece of music, for example, one or more title lines, and they may also change the values of parameters such as the line length that control the final layout on the page. If the title lines 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. The heading is terminated by the first unquoted opening square bracket in the file, and may be completely empty.

### 6.1.5 Stave information

Following the heading there is information for each stave, in this form:

`[stave` *<n> <additional data>*`]`
*<notes and other stave items>*
`[endstave]`

A description of the **[stave]** directive is given in section 10.2.90. There may be up to 63 normal staves, numbered from 1 to 63. Data may also be supplied for a stave numbered 0, which has special properties (☞ 6.15). The normal staves are output in numerical order down the page. If a stave numbered *n* is present, 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.

A PMW input file need not contain any stave data at all; 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.

### 6.1.6 Multiple movements

A PMW file may contain more than one 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.

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. The start of a new movement is indicated by the **[newmovement]** stave directive, which must appear following the information for a stave. 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.

PMW starts a new page at the beginning of a new movement, unless there is enough room on the page for the headings and the first system, or, if the first system contains only one stave, two systems. This can be overridden by options on the **[newmovement]** directive (☞ 10.2.49). In general, most parameters that can be set by heading directives persist from movement to movement, but **doublenotes**, **halvenotes**, **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.

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

In this example, **notespacing** sets absolute note spacings at the start of a movement. Such spacings are reset as the defaults at the start of subsequent movements.

```
notespacing *1.2
```

In this example, **notespacing** is used to multiply the note spacings by a factor. Such a 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**, **righttoleft**, **sheetdepth**, **sheetsize**, **sheetwidth**, **stretchrule**, and **textfont**.

## 6.2 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.

### 6.2.1 *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.

### 6.2.2 *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 that is being defined. That is, a comment is permitted on a ***define** directive, provided there are either no quotes, or only matched pairs of quotes, before the start of the comment. If you use macros to generate partial strings, with unmatched quotes in the defining lines, 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 entered 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 a digit. The semicolon is removed from the text when the substitution takes place. If an actual semicolon is required in the input following a substitution, two semicolons must 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 or not a name has been defined (☞ 6.2.5). An example of the use of a simple macro is given in section 4.1.

### 6.2.3 Macros with arguments

There are times when it is useful to be able to vary the text that 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 that extend to the right of a note but end in mid-air rather than on the next note. The input to achieve this for the note `g'` could be:

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

To shorten this input, a macro with an argument can be defined as follows:

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

The parentheses 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 (parentheses). 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 parentheses and commas form part of the argument; if, for example, there is a space after the opening parenthesis or 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 also times when it is necessary to include commas and parentheses as part of an argument. The following rules make this possible:

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

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

- To include an unmatched opening or closing parenthesis or a comma that is not within parentheses, the character & is used as an escape character. For example, if a note with a bracketted (parenthesized) accidental is used with the `hang` macro, the input is:

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

Without the & preceding it, the accidental's closing parenthesis would be interpreted as terminating the argument list.

- If an argument contains matched double quote characters, commas and parentheses (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 substitutions in the definition of another macro.

If a macro that 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 as an argument list in the macro definition, for example:

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

When the macro is called, empty and missing arguments are replaced by the defaults.

```
&hang()      behaves as  &hang(g',/a)
&hang(B)     behaves as  &hang(B,/a)
&hang(,/b)   behaves as  &hang(g',/b)
```

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 entered.

### 6.2.4 *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:

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

If the name does not start with a slash, it is interpreted relative to the directory containing the current input file, unless the current input is the standard input, in which case 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.

### 6.2.5 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 true, subsequent lines of the input, up to ***else** or ***fi**, are processed. If the condition is not true, these lines are skipped. When ***else** is used to terminate the block of lines after ***if**, 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 in a condition to test whether either one of two (or more) conditions is true:

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

If a condition is preceded by the word ‘not’, the sense of the condition is negated:

```
*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’, 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’, it 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, 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.

Here are some examples of the use of the conditional preprocessing directives:

```
*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** on 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.

## 6.3 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 that are used to verify the layout of systems on the page. This 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 that contain **[nocount]** directives are identified by the number of the previous counted bar, followed by '.1', '.2', etc. as needed. This also applies to uncounted 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 (☞ 3.3). The count is given as the number of bars that do not contain **[nocount]**, followed by the number of bars that do contain **[nocount]**, if any, enclosed in parentheses and preceded by a plus sign. The count for the example above would be '3(+2)'.

## 6.4 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). One millimetre is 2.835 points. Whenever a dimension is required in a PMW directive, its units are always points.

```
linelength 720
```

This example specifies a line length of 720 points, that is, 10 inches. PMW works internally in millipoints (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.

```
barlinespace 3.5
```

This example specifies that the horizontal 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. 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 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 noteheads | 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.

## 6.5 Paper size

By default, PMW assumes that printing is to take place on A4 paper and so it creates a page image appropriate to that size. If a different paper size is required, the **sheetwidth** and **sheetdepth** directives can be used to specify what its dimensions are. For standard paper sizes, it is not normally necessary to use **sheetwidth** and **sheetdepth**, because the **sheetsize** directive, which takes as its argument one of

the words ‘A3’, ‘A4’, ‘A5’, ‘B5’, or ‘letter’, can be used instead. This 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 paper 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. All the **sheet**... directives may appear only in the first movement of a file.

In the most common case, the page image size fits the 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, for certain paper sizes. Details of this are given in chapter 3.

## 6.6 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** (☞ 8.1.65) heading directive is used to specify the allocation of a MIDI voice and/or particular PMW staves to a MIDI channel, and the **[midichannel]** (☞ 10.2.41), **[midivoice]** (☞ 10.2.44), and **[midipitch]** (☞ 10.2.42) 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]** (☞ 10.2.67) 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 **midivolume** (☞ 8.1.71) or **[midivolume]** (☞ 10.2.45) 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.

Notes that are suppressed in printed output by the use of **[notes off]** are by default also omitted from MIDI output. The heading directive **midifornotesoff** can be used to change this behaviour.

## 6.7 Headings and footings

There are three different sets of heading/footing directives:

- **heading** and **footing** specify text that is printed once, on the first page of a piece.
- **pageheading** and **pagefooting** specify text that is printed on the second and subsequent pages of a piece.
- **lastfooting** specifies text that is printed only on the final page of a piece.

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), 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 (in other words, this is still the first page of that movement) and the subsequent movements do not themselves have overriding **footing** directives, 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:

```
[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 does start 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]
```

This causes PMW to use the **lastfooting** setting for this purpose. Its value can then be reset in the new movement.

## 6.8 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, 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 music 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.

## 6.9 Key and time signatures

Key signatures are specified by key letter, followed by a PMW accidental character if necessary, and then 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. All the standard key signatures are supported. See the next section for a discussion of key signatures after transposition, and the **printkey** directive (☞ 8.1.97) for a way of printing non-standard key signatures.

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

Time signatures are specified by separating two numbers with a slash. For example, 3/4 specifies waltz time. PMW imposes no limitations on the values of the numbers used in time signatures. 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 `c`.
- 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 `c`, but expects there to be four minims rather than four crotchets in a bar, and `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 the **printtime** directive 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.

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

## 6.10 Transposition

Octave transposition can be specified for each stave, to simplify the input notation. See the **[octave]** and the various clef directives (**[treble]**, **[bass]**, 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. A piece that 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 that 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:

| | | | | |
|---|---|---|---|---|
| C♭ | major | instead of the default | B | major |
| C♯ | major | " | D♭ | major |
| F♯ | major | " | G♭ | major |
| A♭ | minor | " | G♯ | minor |
| A♯ | minor | " | B♭ | minor |
| D♯ | minor | " | E♭ | minor |

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

If a note is specified with an accidental, 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.

### 6.10.1 Transposition of key and chord names

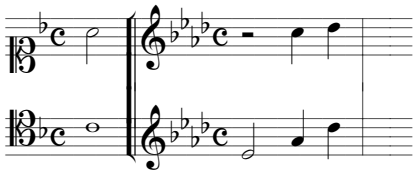
PMW can automatically transpose the names of keys and chords in text strings. This is achieved by means of a special escape sequence `\t`.

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

In this example, the sequence `\tE$` is replaced by E♭ when no transposition is taking place and by F when a transposition of +2 is set. Full details of string escape sequences, including key and chord name transpostion, are given in section 6.14.

## 6.11 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 by one bar. The input is 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, 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.

## 6.12 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 (for example, 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 <*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, and 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, or use the **-F** command line option to specify an additional directory where they may be found.

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

## 6.13 Font sizes, aspect ratios, and shearing

Many PMW directives allow you to specify a size for a font. For example, when defining a heading:

```
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:

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

The first additional number is a horizontal stretching factor that 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 number 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, and a negative one makes it slope to the left. Sheared roman fonts are sometimes used instead of italic fonts:

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

This example prints the heading with a 20° shear.

*This 10-point font is sheared by 20 degrees.*

Stretching and shearing values can be specified in all the places where a text font size can be specified.

## 6.14 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 (single) typographic closing and opening quote characters, respectively, in fonts whose fontmetrics file specifies the Adobe Standard Encoding. This is the case for all the default fonts except the music and symbol fonts. A closing quote character is the same as an apostrophe.
- The backslash character \ is an *escape character* (see below).

There are also some characters that are treated specially in some specific types of string:

- In vocal underlay or overlay strings (☞ 9.12), 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.

### 6.14.1 Unicode and UTF-8 encoding

This section is rather technical. Unless you need to know some of the deep details of character handling, you can probably ignore it and the following section, and skip to section 6.14.3 (*Escaped characters*) below.

The standard PostScript fonts contain many more characters than are available on a computer’s keyboard. The basic computer character set, often referred to as ASCII, comprises 95 characters (including space), whose code values lie between 32 and 126, inclusive. These are the characters you can type on the keyboard. Codes less than 32, together with code 127, are used for control functions such as ‘newline’ and ‘delete’. Codes greater than 127 are not defined in the ASCII character set.

When people needed more than 95 characters, a number of different codes were defined, including several called ISO-8859-*n* (for different values of *n*). These all kept the same meanings for codes 0–127, but added different sets of characters for the values 128–255. The most widely used of these codes is ISO-8859-1 (‘Latin1’), which contains many of the accented characters used in Western European languages.

Before release 4.10, PMW interpreted each byte in a text string as a single character, with a value in the range 0–255. Values less than 128 were interpreted as ASCII, and values in the range 160–255 were taken from ISO-8859-1. Some of the values in the range 128–159 were subverted for additional characters such as en-dash and em-dash that are not defined in ISO-8859-1. In addition, access to non-ASCII characters was available via escape sequences so that a PMW input file could contain only ASCII bytes and still use all the ISO-8859-1 characters, though in practice input files in ISO-8859-1 code were used.

The problem with using many different character codes is that it is hard to switch between them. Even when printing music, where there is not much text, the name of the composer may be in one language, requiring a certain set of accents, and the rest of the text may be in another, requiring different accents. The long-term solution to this problem is Unicode, which is a single encoding for all the world's characters. Unicode character values are no longer constrained to lie in the range 0–255, thus enabling the character sets from many languages to be simultaneously defined. However, this means that no longer can every character fit into one byte of memory.

The Unicode encoding copies ISO-8859-1 for the first 256 characters. Furthermore, there is a way of encoding these characters called UTF-8 which keeps the byte values 0–127 as the encoding for those character values. For character codes greater than 127, a multibyte encoding is defined. If a file consists of bytes containing only the original 127 ASCII values, it is a valid UTF-8 encoded Unicode file. From version 4.10 onwards, PMW treats the bytes that make up quoted strings as UTF-8 encoded Unicode character sequences. For example, the following byte sequence (where each byte is expressed in hexadecimal) encodes two characters:

```
41 C2 A6
```

The first byte, with a decimal value of 65, is less than 128, and is therefore an entire character on its own. The remaining two bytes together encode the value 166. If you have a text editor that can create files using UTF-8 encoding for Unicode characters, you can use these characters directly in PMW strings. If not, you can refer to characters whose values are greater than 127 using escape sequences, as described in section 6.14.3 below.

There is a complete list of the characters in standard PostScript fonts in chapter 11. These are the characters that are accessible using Unicode (or escape sequences) in text strings. There are, of course, many other characters that are defined in Unicode, but which are not present in these fonts. Some of them (for example, Greek letters) exist in the PostScript *Symbol* font, which can be used via PMW escape sequences (☞ 6.14.3). This font, however, does not use Unicode encoding.

### 6.14.2 Backwards compatibility for character strings

Some byte values are invalid in UTF-8 strings. In particular, a single byte with a value greater than 127 that is between two bytes whose values are less than 128 cannot occur. When PMW encounters such a byte in a string, it interprets it as a single-byte encoding of a character in the range 128–255. This is done for backwards compatibility so that input files for PMW releases prior to 4.10 that made use of the ISO-8859-1 encoding directly can still be processed. The output is likely to be correct in most cases; only when there are several high-valued bytes in a row, and they happen to form a valid UTF-8 character, will things go wrong.

- If an existing PMW input file uses only ASCII characters, and does not (by means of escape sequences) refer to characters in the range 128–159 by number, it should continue to work as before.
- If an existing PMW input file contains bytes with values in the range 160–255, but these are always isolated between characters with values less than 128, it should also continue to work as before.
- An existing PMW input file that contains sequences of two or more bytes with values greater than 159 may or may not work, depending on the exact byte values.
- An existing PMW input file that uses characters in the range 128–159 by any method will definitely have to be updated, because the codes for those characters have changed.

For maximum portability of PMW input files, it is recommended that only ASCII characters be used in the file, with escape sequences for high-valued characters.

### 6.14.3 Escaped characters

From PMW release 4.10 onwards, it is possible (as just described) to use UTF-8 encoding to directly represent Unicode character values in text strings. However, it is also possible to use just the set of ASCII characters in PMW input files, without loss of functionality. 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. For example, the following sequences are available to represent some of the commonly accented characters in European languages:

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

Many other accented characters are available, and there are other escape sequences for other special characters:

| | |
|---|---|
| `\c)` | prints as © |
| `\c]` | prints as © (but see below) |
| `\ss` | prints as ß |
| `\?` | prints as ¿ |
| `\\` | prints as \ |
| `\'` | prints as ' (because `'` on its own prints as ’ ) |
| `\`` | prints as ` (because `` ` `` on its own 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.

A complete list of all the available special characters and their escape sequences is given in chapter 11. Other escape sequences are summarized in chapter 14. Characters that are not on the keyboard can be included in strings by giving the character number, in hexadecimal preceded by `x` or in plain decimal, enclosed between two backslashes. For example, `\xb7\` or `\183\` prints a bullet character. Characters that are treated specially in text strings can also be printed by this means. For example, `\x22\` prints a double-quote character, which cannot appear literally in a string.

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 `\x7c\` can be used to print a vertical bar in a heading line, where a literal vertical bar is interpreted as a left-middle-right separator.

Characters from the PostScript *Symbol* font are also available for use in text strings. This font contains some large brackets that are sometimes useful, as well as a number of other special characters that are not present in the ordinary text fonts. Unicode encoding is *not* used for this font. To include a single character from the Symbol font, specify its hexadecimal character number in the font’s default encoding preceded by `sx`, or its decimal character number preceded by `s`, enclosed in backslashes. For example, `\s174\` prints character 174, which is the → right arrow symbol.

### 6.14.4 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 this form:

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

This example prints ‘page *n*’ on the left or right of the page, depending on the value of the page number.

### 6.14.5 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.

### 6.14.6 Transposing key and chord names

A special escape sequence is provided to define the names of keys or 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 accidental characters #, $, or (for completeness) %. Such a sequence has two effects; firstly, the key or chord name is transposed in the same way as its base note would be transposed, and secondly, if the new 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 key or chord names that involve accidentals.

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

When a string that involves a transposable name appears in a heading or footing line, only **transpose** heading directives that are earlier in file are applied to it, because the transposition is performed when the string is read. It is also important to specify the key signature before the transposable heading or footing, in case it affects the result. For example, consider this directive:

```
heading "Sonata in \tC minor"
```

If no key is specified before this line in an input file, and a transposition of +1 is applied, the result is ‘Sonata in D♭ minor’, because PMW assumes the key of C major. However, if the key is set to C minor before the heading line, the result of transposing by +1 is ‘Sonata in C♯ minor’.

### 6.14.7 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, but for other text it is italic, which is appropriate for dynamic marks such as *ff*, though sometimes bold italic is used for this kind of mark. Tempo marks at the start of pieces are normally printed in bold face, as in the following example:

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

The default fonts for each type of text 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.

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.

### 6.14.8 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 that can be requested for any particular item of text (☞ 8.1.126, 9.9).

Whenever the size of a text font is specified, an associated aspect ratio and/or shearing angle may also be specified (☞ 6.13). Stave text strings that are not underlay or overlay can be rotated so that they print at an angle (☞ 9.9). 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 10.2.90.

### 6.14.9 Music characters

An escape mechanism can be used to include single music ‘characters’ in textual output without having to change to the music font and back again. Typical uses of this are for indicating tempo by printing a note followed by an equals sign and its metronome mark, or for printing sharps and flats in the names of instruments. To include a note from the music 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, for example, `\*c.\`. The accidental characters are available as follows:

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

A typical example of a tempo mark that uses this facility might be:

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

This prints as:

**Maestoso ♩ = 60**

Music characters included in character strings with a single asterisk in this way are printed using a music font that is 9/10 the nominal size of the surrounding text characters. This is an appropriate size for items such as tempo marks. Thus, if a 10-point text font is being used, a 9-point music font is used with it. The music font that is used to print the music being typeset is a 10-point font, and it is sometimes useful to be able to print music characters at full size. If two asterisks are present in an escape sequence for a music character, the character is taken from a music font that 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 used for music on the stave.

If more than one escape 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.

There are a number of 'characters' in the music font that do not actually cause any marks to be made on the page. 'Printing' these characters has the effect of moving the current printing position, thus affecting the placing of any subsequent 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 |

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.

In addition to those characters that are available via the escape sequences just described, it is also possible to print *any* character from the music font by specifying its hexadecimal character number preceded by `x` or just its decimal number, preceded by one or two asterisks, between backslashes. A list of the available characters is given in chapter 12. For example, the sequence `\*45\` prints a crotchet rest. If you want to print a long sequence of characters from the music font, it is sometimes more convenient to use a font-changing escape sequence, as described in the previous section, rather than individually escape each character.

### 6.14.10 Guitar chord grids

Guitar chord grids can be printed relatively straightforwardly as a string of characters in the music font, printed above the stave. The grid character itself (character 131) has zero typographic width. If a guitar dot character (116, or 't') is printed immediately following, it is placed on the fourth fret mark of the first guitar string. The typographic width of this character is set so that after it is printed, the current printing point is moved to the next guitar string.

Two of the special characters for moving up and down (119 and 120, that is, 'w' and 'x') can be used to move between frets, and the right moving character 125 ('}') can be used to move to the next string if you do not want to print any symbol on a string. The guitar ring (open string) and cross (silent string) characters (117, or 'u', and 183) behave exactly as the dot. To print them above the first fret you need to move up one and a half times the normal fret distance.

`"\mu\\131\xxxx~\183\|wwwtxtwwtxxxtx~u"` prints 

The sequence `\mu\` switches into the music font, and \131\ prints character 131, the grid. The sequence `xxxx~` moves the printing point up by four and a half frets, which takes it to above the grid, where character \183\, the 'x', is printed. The sequence `|www` moves down by three and a half frets so that `t` prints a dot on the third fret of the second string. And so on...

If you want to print the names of the chords, you can give them as additional strings that can be separately positioned. Section 9.9 discusses text strings in stave data. If you are going to use a lot of guitar chords, it is most convenient to define macros for the text strings.

### 6.14.11 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:

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

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

## 6.15 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. 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, but you can use the **copyzero** heading directive to have copies of stave zero printed over any number of staves. This directive 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.

```
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, its settings are copied from the previous movement. If a stave over which stave zero is being printed is suspended, 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, 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, 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. The default for **copyzero** is:

```
copyzero 1
```

This therefore 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:

```
[copyzero 4]
```

This example 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 marks are specified in stave zero, and there is a need to adjust their height for certain staves, it should be noted that the marks are drawn when the bar in which their end point is determined is processed. Consequently, it is that bar in which **[copyzero]** should appear.

## 6.16 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, so you have to use the **[suspend]** directive in to tell it when to do so (☞ 10.2.96). Resumption of printing is automatic, but there is also a **[resume]** directive for forcing it to happen at a particular bar.

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. 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 makes it possible to suspend individual staves right from the start (☞ 8.1.123).

When a single part is being printed, suspension normally has no effect, because 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 instead of `R!`, it prevents the amalgamation of adjacent bars and may lead to suspendable systems, which are undesirable in single parts. In these cases, therefore, any **[suspend]** directives that are present for use when printing the full score should be skipped (using the ***if** preprocessing directive) when printing the part.

Normally, a stave that 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 (☞ 10.2.60), in which case completely empty bars are not printed at all. This can be useful for printing *ossia* passages. A completely empty bar has no data at all specified for it; a bar containing a rest is not a completely empty bar.

# 7. Drawing facilities

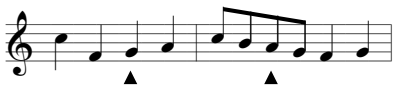
PMW contains a facility for drawing simple shapes, defined by the user, positioned relative to notes, bar lines, headings, stave names, or gaps in slurs and slur-like lines. This makes it possible to print music notation that is not provided explicitly by PMW. For example, the facility can be used to draw piano pedal marks, boxes round notes, vertical brackets between notes, and to print unusual marks 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 that is described below. Whenever the triangle shape is wanted, the stave directive **[draw triangle]** is given before the relevant note.

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



If many 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.

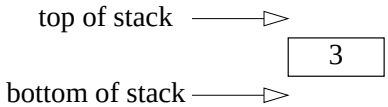
## 7.1 Stack-based operations

A stack is a means of storing items of data such that the last item that 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’.

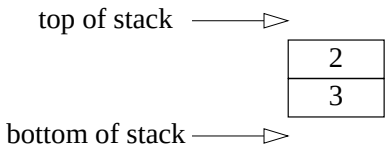
PMW’s drawing stack contains numbers and references to text strings rather than trays. (Discussion of strings is postponed till section 7.13.) When PMW is obeying a set of drawing instructions, if it encounters a number in its input, the number is ‘pushed’ onto the top of the stack. Consider the following fragment of a drawing program:

```
3 2 add
```

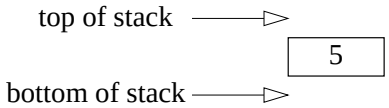
In this example, the first item is the number 3, so the effect of reading it is to put the stack into this state:




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 that 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:




If an operator is encountered that requires more numbers on the stack than there are present, *stack underflow* is said to occur. PMW generates an error message and abandons the drawing. The stack mechanism is very simple, and operates quickly. However, it does make it possible to write very obscure code that is 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 safely. *User variables* (☞ 7.12) must be used instead.

## 7.2 Drawings with arguments

Whenever a drawing function is called, either by the **[draw]** directive or as part of some other directive (for example, **heading**), its name may be preceded by a list of numbers or text strings (☞ 7.13), separated by spaces. These are pushed onto the drawing stack immediately before the function is obeyed, and therefore act as arguments for the function.

```
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 (☞ 6.2.3).

## 7.3 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
```

## 7.4 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.

## 7.5 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
```

leaves the value 1 (*true*) on the stack, and

```
25 4 lt
```

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

## 7.6 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.

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

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

## 7.7 Stack manipulation operators

There are several operators that can be used to manipulate items on the stack.

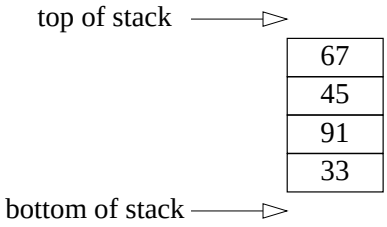
- **copy**: Remove the top item, which must be a number, then duplicate the sequence of that number of items. For example, if the stack contained the numbers 13, 23, and 53, after **2 copy** it would contain 13, 23, 53, 23, 53.
- **dup**: Duplicate the item on the top of the stack. This has the same effect as **copy** with an argument of 1.
- **exch**: Exchange the two top items on the stack.
- **pop**: Remove the topmost item 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 item from the top of the stack, and inserting it below the last item 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 item 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**.

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

When **roll** is obeyed, the three items 45, 67, 91 are rolled upwards one place and the result is:




## 7.8 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:

- A drawing 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. The vertical position of the origin is on the bottom line of the stave if the stave has 3 or more lines. For staves with zero, one, or two lines, the vertical position of the drawing origin is where the bottom line of a 5-line stave would be. The horizontal position of the origin is either at the left-hand edge of the associated note, or at the bar line if there is no following note. 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.
- A drawing can be associated with a heading or footing. 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 (☞ 8.1.47) for more details.
- A drawing can be associated with a gap in a line that is defined by the **[line]** directive, or with a slur. The origin of the coordinate system is in the middle of the gap. For details see the **[linegap]** (☞ 10.2.39) and **[slurgap]** (☞ 10.2.84) directives.
- A drawing 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, at the level of the bottom line of the stave. For details, see the **[stave]** directive (☞ 10.2.90).

## 7.9 Moving the origin

There is an operator called **translate** that 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.

## 7.10 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* 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 normally consists 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 that contains the magnification of the current stave (when the drawing is associated with a stave), so that adjustments can be made explicitly when required.

Whenever a pair of coordinates is required to be on the stack, it is always the x-coordinate that 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 Bézier 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 Bézier curve control points. If you are not familiar with Bézier 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. The default is to fill in black, but this can be changed by the **setgray** operator. After filling, the path is deleted, and a new one can then 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 (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. A single number between 0 (black) and 1 (white) is taken from the stack. For example:

  ```
  0.5 setgray
  ```

  Filling a path is analagous to using opaque paint, so **setgray** 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]** because 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 (☞ 7.13).
- **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.

## 7.11 System variables

In order to set up drawings that are positioned relative 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* that 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 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**: This variable contains distance from the left-hand edge of the leftmost notehead to the left-hand edge of the accidental that is furthest to the left, as a positive number. If there are no accidentals, the value is zero.
- **barnumber**: When used in a bar, this variable 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**: This variable 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**: This variable contains 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, this variable contains the width of this notehead as a positive number; otherwise its value is zero.
- **headright**: This variable contains the width of the notehead, except that, for a chord with an upwards pointing stem in which there is a notehead on the ‘wrong’ side of the stem, the value is twice the notehead width.
- **headtop**: This variable contains the y-coordinate of the top of the notehead; if the drawing function precedes a chord, this refers to the highest notehead.
- **leftbarx**: This variable contains 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 variable contains the value 2 (scaled to the stave size) if the bottom of the lowest notehead is on a stave (or ledger) line (that is, the notehead itself is positioned in a space); otherwise its value is zero.
- **linegapx** and **linegapy**: When a drawing is being obeyed as part of the **[linegap]** directive, these variables contain the coordinates 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, but scaled to the current magnification. For example, with a magnification of 2 and the default pagelength of 480, the value in **linelength** is 240.

- **linetop**: This variable contains the value 2 (scaled to the stave size) if the top of the highest notehead is on a stave (or ledger) line (that is, the notehead itself is positioned in a space); otherwise its value is zero.
- **magnification**: This variable contains the value of the overall magnification. If used on a stave, it does not include the relative magnification (see **stavesize** below).
- **origin**: This variable 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.
- **originx** and **originy**: These variables are for use when more than one note position is participating in a drawing. They contain 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 (☞ 7.23).
- **pagelength**: This variable contains the current page length, as set by the **pagelength** heading directive, but scaled to the current magnification. For example, with a magnification of 2 and the default page length of 720, the value in **pagelength** is 360.
- **pagenumber**: This variable contains the current page number.
- **stavesize**: This variable contains the relative magnification for the current stave, as specified by the **stavesize** heading directive.
- **stavespace**: This variable contains the stave spacing for the current stave, that is, the vertical distance between this stave and the next one.
- **stavestart**: This variable contains the x-coordinate of the left-hand end of the current system, relative to the current origin.
- **stembottom**: This variable contains 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 is the same value as **headbottom**.
- **stemtop**: This variable contains 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 is the same value as **headtop**.
- **systemdepth**: This variable contains the distance from the bottom of the top stave of the current system to the bottom of the bottom stave.
- **topleft**: This variable contains 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 variable 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.

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

The value that is set in a variable may be changed as often as necessary. A variable's name must appear in a definition (preceded by a slash) earlier in the input than its first use as a reference. 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 that is defined but never obeyed can be used for the sole purpose of defining user variable names.

## 7.13 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 can be processed only by special string operators. The most important of these is the **show** operator, which causes the string to be printed at the current point:

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

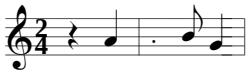
The default font is roman, but the string may contain font changes and other escape sequences, just like any other string. It may be followed by one or more of the following options:

| | |
|---|---|
| `/box` | enclose the string in a rectangular box |
| `/c` | centre the string at the current point |
| `/d`<*n*> | move the string down by <*n*> points |
| `/e` | align the end of the string with the current point |
| `/l`<*n*> | move the string left by <*n*> points |
| `/r`<*n*> | move the string right by <*n*> points |
| `/ring` | enclose the string in a ring (circular if a short string) |
| `/rot`<*n*> | rotate the string by <*n*> degrees (☞ 9.9) |
| `/s`<*n*> | print the string at size <*n*> (☞ 8.1.126) |
| `/u`<*n*> | move the string up by <*n*> points |

The `/u`, `/d`, `/l`, and `/r` options are not particularly useful on strings that are part of drawing definitions, because you can position such strings with **moveto** or **rmoveto**. However, when a string is an argument to a drawing it is sometimes helpful to be able to modify the position that is defined within the drawing. These moving options are ignored when the string is used by some operator other than **show** (for example, **stringwidth**).

When the string is centred or end-aligned, the printing of the string does not change the current point; in the default, left-aligned case, the current point is moved to the end of the string. A path 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.

```
draw dot
  0 headbottom 2 linebottom sub add moveto "\mf\?" show
enddraw
*define bd() [notes off draw dot] &&1-; [notes on]
time 2/4
[stave 1 treble 1]
ra | &bd(a) b-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 ‘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 (a question mark in the music font corresponds to the horizontal dot character). The input could be shortened even further by including the previous note and the bar line inside the macro expansion.

## 7.14 String operators

As well as **show**, there are three other operators that act on strings.

- 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 that provides a place to store the converted number. The string may be followed by any of the usual string options. When **cvs** is obeyed, the two arguments are removed from the stack and a string containing a representation of the number is pushed back. For example, to print the current barnumber at text size 2, centred at the current position:

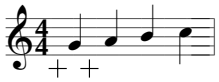
```
barnumber ""/c/s2 cvs show
```

- The **stringwidth** operator 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 (☞ 7.18).
- The **fontsize** operator replaces a string on the stack with the font size associated with it.

## 7.15 Drawing subroutines

One drawing 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 drawing. 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
   -4 -6 moveto draw cross
   10 -6 moveto draw cross
enddraw
[stave 1 treble 1] [draw crosses] gabc' |
```



The subroutine must be defined before the definition of any drawings in which it is called. Subroutines cannot be called recursively, that is, a drawing cannot call itself, and a multi-drawing recursive loop is not possible because a drawing must be defined before it is called.

## 7.16 Blocks

A *block* is a portion of a drawing's coding 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.

## 7.17 Conditional operators

The operator **if** is used to obey a portion of the drawing 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
```

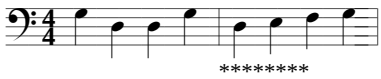
The bar number and the number 5 are pushed onto the stack; the comparison operator **gt** replaces them with 1 if the barnumber is greater than 5, or 0 otherwise. Then a reference to the block is pushed

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 requires two blocks on the stack. The first is obeyed if the condition is true, the second if it is false.

## 7.18 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 (adding 0.5 ensures that the count is rounded to the nearest integer):

```
draw astline
  leftbarx -15 moveto
  leftbarx neg "*" stringwidth pop div
  0.5 add { "*" show } repeat
enddraw
[stave 1 bass 0] gddg | de [draw astline] fg |
```




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.

## 7.19 Drawing in headings and footings

Drawing functions can be obeyed in headings and footings. For example, crop marks, horizontal rules, and borders on title pages can be drawn by this method. For details, see the description of the **heading** directive (☞ 8.1.47).

## 7.20 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 (☞ 10.2.90).

## 7.21 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.

## 7.22 Example of use of system variables

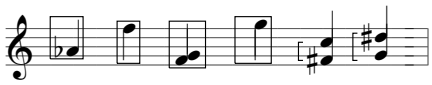
This example illustrates the use of the variables that contain the dimensions of the note that follows the **[draw]** directive:

```
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 that is two points to the left of the leftmost accidental, taking into account any notehead that 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.

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

The second line of the drawing instructions 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.

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

The third line does a computation for the rectangle’s width, which is 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.

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

The remaining lines use these stacked values to complete the rectangle. 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, and 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.

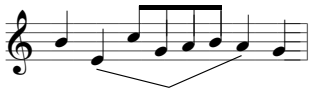
## 7.23 Example of inter-note drawing

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 does not actually do any drawing at all. It just saves on the stack the y-coordinate of the bottom of the next note, and the absolute x-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 (☞ 7.12); 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* x-origin onto the stack, and subtract it from the saved value. 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, and 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 finds 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 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

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 at the mid-point of its notehead, and then cause it to be stroked.

# 8. Heading directives

A heading section in a PMW file contains *heading directives*. Each starts with a keyword, and sometimes the keyword is followed by numerical or other data. It is usual to start a new line for each heading directive, but this is not a requirement. A heading section is terminated by an opening square bracket character that is not in a text string or in a comment (following an @). None of the heading information is mandatory because there are default values for all the parameters; a heading section 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, that is, only in the first movement (☞ 6.1.6). In general, heading directives may appear in any order, but there are some fairly 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.

## 8.1 Alphabetical list of heading directives

The heading directives are now described in alphabetical order. A number of heading directives take a list of numbers as data. In all cases, such numbers can be separated by commas or spaces, or commas and spaces, except when such a list is continued onto another input line, when the final number on the first line must be terminated by a comma, to indicate that more data follows. The values set by most directives persist from movement to movement. When this is not the case, it is noted in the description.

### 8.1.1 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; it just moves 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, that is, nearer the note) or negative (move to the left, that is, further from the note).

```
accadjusts 1.8
```

This example has the effect of moving any accidental that precedes a breve 1.8 points to the right.

### 8.1.2 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.

### 8.1.3 Bar

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

### 8.1.4 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 argument is the maximum number of bars in the current and subsequent movements.

### 8.1.5 Barlinesize

When 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** must be followed by 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.

```
barlinesize 1
```

This example requests that full size barlines be used throughout; they will look somewhat fat on any staves whose size is less than 1, and thin on any whose size is greater than 1.

### 8.1.6 Barlinespace

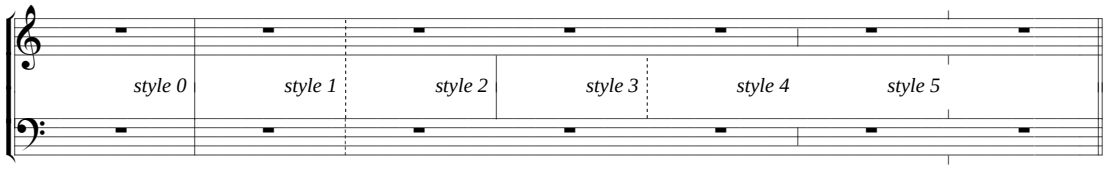
This directive gives control over the amount of space left after bar lines. It must be followed by a single dimension, which may be preceded by a plus or a minus sign, indicating a change to the existing value. If neither of these is present, the number specifies an absolute value. The default value can be reset in a subsequent movement by following the directive with an asterisk instead of a number. 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.

### 8.1.7 Barlinestyle

This directive specifies the way in which bar lines are to be printed on all staves. It takes a numerical argument, with a value in the range 0–5. There is also a **[barlinestyle]** stave directive that 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 specifies 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, to force a particular style. The following example shows the six available styles:




Note that the **breakbarlines** directive can be used to specify breaks in all bar lines at particular staves when style 0 or 1 is used, and an individual bar line can be broken by using **[breakbarline]**.

### 8.1.8 Barnumberlevel

This directive adjusts the level of printed bar numbers. It must be followed by a plus or a minus sign and a dimension.

```
barnumberlevel +4
```

This example prints all bar numbers four points higher up the page than they would otherwise have been.

### 8.1.9 Barnumbers

This directive specifies that bars are to be automatically numbered. **Note**: An incomplete bar at the start of a movement is counted for the purpose of bar numbering, unless it contains the **[nocount]** stave directive (see section 5.2.2 for an example). Automatic bar numbering can be overridden for individual bars by means of the stave directive **[barnumber]** (☞ 10.2.7). Several different numbering options are available, the general form of **barnumbers** being as follows:

`barnumbers` <*enclosure*> <*interval*> <*fontsize*> <*fontname*>

The first argument, 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 argument 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 line of each movement. If a number is given, it specifies the interval between bar numbers.

```
barnumbers boxed 10
```

This example causes a bar number, enclosed in a box, to be printed every 10 bars. The third argument is optional; it specifies the size of the font in which the numbers are printed. The default size is 10 points.

```
barnumbers line 8.5
```

This example numbers the bars at the start of each system, using a font of size 8.5 points. The final argument, which is optional, specifies the font (typeface) for printing the bar numbers. The default is roman.

```
barnumbers 5 9/1.1 italic
```

This example prints bar numbers every 5 bars in a 9-point italic font, horizontally stretched by 1.1.

### 8.1.10 Beamendrests

This directive, which has no arguments, requests PMW to include rests at the ends of beams within the beams (☞ 9.7.3).

### 8.1.11 Beamflaglength

The length of short, straight note flags that are used with beams (for example, 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.

### 8.1.12 Beamthickness

This directive takes a single dimension as its argument; 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.

### 8.1.13 Bottommargin and topmargin

The **bottommargin** and **topmargin** directives make it possible to reserve white space at the top or bottom of a page, within the overall page length, provided that there is room to do this after the systems have been fitted onto the page. These directives give some additional control over the vertical justification action described in section 8.1.51, once the pagination of a piece is determined. In each case, a single dimension is required.

```
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 these 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, 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, 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', 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', 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.

### 8.1.14 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, whereas 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.

```
bracket 1-4,5-7
brace 8-9
```

This example specifies that staves 1–4 and 5–7 are to be joined by brackets, and 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.) 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 following list.

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. 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.

### 8.1.15 Bracestyle

The default brace shape curves a lot at the ends and almost touches the staves. An alternate form that does not curve so much at the ends can be selected by specifying **bracestyle 1**. The default can be reset in a subsequent movement by **bracestyle 0**.

### 8.1.16 Breakbarlines

By default, PMW draws bar lines through all the staves of a system without a break. The **breakbarlines** directive specifies the staves after which there is to be a break in the bar lines. It is followed by a list of stave numbers.

```
breakbarlines 3 6 8
```

This example 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:

```
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.

### 8.1.17 Breakbarlinesx

The **breakbarlinesx** directive acts exactly as **breakbarlines**, except that the final bar line of each system is *not* broken, but is drawn solid right through the system.

### 8.1.18 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.

### 8.1.19 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.), a breve rest sign is used.
- If there are 12 crotchets (6/2 or 12/4 or 2*3/2 etc.), a dotted breve rest sign is used.
- If there are 6 crotchets (3/2 or 2*3/4 etc.), a dotted semibreve rest sign is used.
- Otherwise a semibreve rest is used.

### 8.1.20 Caesurastyle

The default caesura 'character' is two slanting lines through the top line of the stave. This directive specifies an alternative, single line, style if it is followed by the number 1. The default can be reset in a subsequent movement by specifying 0.

### 8.1.21 Check

This directive, which has no arguments, can be used to override an occurrence of **nocheck** in a previous movement.

### 8.1.22 Checkdoublebars

This directive, which has no arguments, can be used to override an occurrence of **nocheckdoublebars** in a previous movement.

### 8.1.23 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.

```
Clefsize 0.7
```

This example specifies that intermediate clefs are to be printed at 0.7 times the normal size.

### 8.1.24 Clefstyle

Some early editions use 𝄡 for F-clefs and 𝄠 for 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.

### 8.1.25 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.

### 8.1.26 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 section 6.15.

### 8.1.27 Cuegracesize

This directive, which takes a single number as an argument, specifies the font size to be used when printing grace notes in bars containing cue notes. See the **[cue]** directive for further details.

### 8.1.28 Cuesize

This directive, which takes a single number as an argument, specifies the font size to be used when printing cue notes. See the **[cue]** directive for further details.

### 8.1.29 Dotspacefactor

This directive specifies the factor by which the horizontal space after a dotted note is extended. The default value is 1.2.

```
dotspacefactor 1.5
```

In this example, the amount of space that 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, because 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.

### 8.1.30 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**.

### 8.1.31 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 7.

### 8.1.32 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.

### 8.1.33 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.

### 8.1.34 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. It takes a positive or negative number as its argument. 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.

```
extenderlevel 1
```

This example moves extender lines up to near the baseline, and larger values can be used to place them nearer the middle of the text characters.

### 8.1.35 Fbsize

By default, text that 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 text.

```
fbsize 8.2
```

This example specifies a somewhat smaller font. Individual figured bass text strings can have an explicit size specified (☞ 9.9).

### 8.1.36 Footing

This directive has the same arguments as **heading**, and they have the same meaning – see **heading** below for a full description. Several footing lines may be specified. **Footing** sets up text to be printed at the foot of the first page only, and setting any footing line for a new movement automatically cancels all the footings that were in force for the previous movement.

```
footing "Copyright \c) 1992 J.S. Bach"
```

Note the use of the escape sequence `\c)` in this example to obtain a copyright symbol. If a type size argument is not given, 8-point type is used. 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.

Footing lines 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 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.

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, text specified by **pagefooting** is printed on the first page as well as on subsequent pages. If the movement is only one page long, **footing** overrides **lastfooting**, but **lastfooting** overrides **pagefooting**.

### 8.1.37 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.

### 8.1.38 Footnotesize

This directive sets the type size used for printing footnotes. The default size is 9 points.

### 8.1.39 Gracesize

The default size of the music 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.

### 8.1.40 Gracespacing

By default, a grace note is printed 6 points to the left of the note that follows it. If there are two or more grace notes, the distance between them is also 6 points by default. This directive allows these values to be changed. It must be followed by either one or two arguments. If only one argument is given, its value is used for both dimensions. If two arguments are given, the first affects the distance between the last grace note and the following main note, and the second affects the distance between multiple grace notes. If the value of either argument is preceded by a plus or a minus sign, it indicates a change to the existing value. If no sign is present, the number specifies an absolute value.

```
gracespacing +2
```

This example increases both dimensions by 2 points.

```
gracespacing -1 8
```

This example reduces the space after the last grace note by one point, and sets the distance between multiple grace notes to 8 points.

### 8.1.41 Gracestyle

When two or more staves are being printed, and a note on one stave is preceded by one or more grace notes, the notes on the other staves that 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, which must be followed by 0 or 1, can be used to make PMW behave differently.

When the style is set to 1, the notes that 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.

### 8.1.42 Hairpinlinewidth

This directive specifies the width of line used to draw crescendo and decrescendo hairpins. Its argument 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.

### 8.1.43 Hairpinwidth

This directive specifies the vertical width of the open end of hairpins. Its argument is the number of points.

```
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.

### 8.1.44 Halfflatstyle

This directive selects which character to print for a half flat. It must be followed by one of the numbers 0 (the default) or 1. There is an illustration of the different styles in section 9.6.2.

### 8.1.45 Halfsharpstyle

This directive selects which character to print for a half sharp. It must be followed by one of the numbers 0 (the default) or 1. There is an illustration of the different styles in section 9.6.2.

### 8.1.46 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**.

### 8.1.47 Heading

The **heading** directive defines a line of text to be printed as a heading to the piece or movement. If no headings are specified, no space is left at the top of the first page. You can specify any number of headings, which may appear in two different forms. In the more common form, the keyword is followed by up to three arguments:

`heading` <*fontsize*> `"`<*text*>`"` <*depth*>

The first argument is a number, and is optional. If present, it defines the font 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 argument 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 argument 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 have to be coded explicitly (☞ 6.14); 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**: heading strings do not need to be input on a single line; line breaks in the string are treated as spaces.

The third argument 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 argument 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; the third one prints nothing, but leaves 20 points of space.

```
Heading "|Partita"
Heading 11 "Moderato||J.S. Bach" 14
Heading "" 20
```

The second form of the **heading** directive causes a drawing subroutine to be obeyed at the next heading position (see chapter 7 for more details of drawings). 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 (a horizontal rule) after a heading:

```
draw rule
  0 0 moveto
  0 linelength rlineto
  1 setlinewidth stroke
enddraw
heading "|Some Text" 0
heading draw rule 20
```

The first heading or footing line on a page is always at the same fixed vertical position, so 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. The file *cropmarks* in the *contrib* directory in the PMW distribution contains

an example of this. See the **pageheading** directive for a means of setting up headings for pages other than the first.

### 8.1.48 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 argument 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 9.12.5 for other facilities that can be used to control exactly what is printed between underlaid syllables.

### 8.1.49 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.

### 8.1.50 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.

```
join 1-2, 3-4
```

This example causes solid lines to be drawn joining staves 1 and 2, and 3 and 4, leaving a gap between staves 2 and 3.

```
join 1,2,3,4
```

This example 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
```

This example 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.

### 8.1.51 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** that 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 music 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 creating examples for inclusion in other documents.
- When 'right' is specified without 'left', each music 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 music system begins at the left-hand side of the page, and is stretched so that it ends at the right-hand side, unless it covers less than half the linelength, in which case the behaviour is as if ‘right’ were not specified.

- When neither ‘left’ nor ‘right’ is specified, each music 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 half the page depth has been used, 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, 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 of 44 points 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 output (☞ 3.3), 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 that 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 movement headings are moved down the page. If a new movement starts in the middle of a page that 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.

### 8.1.52 Key

This directive sets a key signature for a movement. It does not carry over to any subsequent movements. 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 argument to the directive is a key signature in the format described in section 6.9.

```
key A$
key BM
```

If no key signature is given for a movement, the default is C major.

### 8.1.53 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.

### 8.1.54 Keywarn

This directive can be used at the start of a new movement to cancel the effect of **nokeywarn** in the previous movement.

### 8.1.55 Landscape

This directive is permitted only in the first movement. It causes all the output to be in 'landscape' format, that is, with the long side of the page horizontal. The value of **linelength** is interchanged with the value of **pagelength**, and likewise the value of **sheetwidth** is interchanged with **sheetdepth**. Subsequent directives affect the new values. The **landscape** directive should appear after any uses of the **sheetdepth**, **sheetdepth**, or **sheetsize** directives.

### 8.1.56 Lastfooting

This directive specifies footing material for the last page of music, replacing any text specified with **footing** or **pagefooting** for that page. The arguments 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.

### 8.1.57 Layout

The **[newline]** and **[newpage]** directives, together with **notespacing**, are useful for occasional over-riding of PMW's default layout choices. However, if a particular layout for an entire piece is required, achieving it with **[newline]**, **[newpage]** and **notespacing** 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. This directive applies only to the movement in which it appears. If a subsequent movement contains no **layout** directive, PMW fills its systems according to the width of the bars. If you are using **layout**, it is not usually necessary to use **notespacing** as well.

It its simplest form, **layout** 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.

```
layout 4
```

This example specifies that every system (except possibly the last one) is to contain exactly four bars. If page breaks are required at particular points, they are specified by a semicolon in the layout list.

```
layout 4, 3; 5, 4, 4;
```

This example 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 compressed until they fit; if too many are specified the result may be very cramped. If the same item or sequence of items is repeated in a layout list, it can be enclosed in parentheses and preceded by a repeat count. Parentheses can be nested.

```
layout 3(4, 5(6);)
```

This example 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.

### 8.1.58 Ledgerstyle

The **ledgerstyle** directive, which must be followed by 0 or 1, can be used to choose between thinner or a thicker ledger lines. The default is 0, which selects the thinner line; on some printers this may look too insignificant, which is why the thicker style is provided.

### 8.1.59 Leftmargin

Normally, PMW centres page images horizontally on the paper. The width of paper used for this purpose can be specified by **sheetwidth**. The **leftmargin** directive can be used to specify a particular left-hand margin instead of centring.

### 8.1.60 Linelength and pagelength

The **linelength** and **pagelength** directives specify the size of the page images that PMW produces; each takes a single argument 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 default page length is 720 points. These values leave generous margins all round on A4 paper. 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.

### 8.1.61 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.

```
longrestfont 13 bolditalic
```

The default size is 10 points; if this directive is not used, the numbers are printed in the roman font.

### 8.1.62 Magnification

This directive is permitted only in the first movement. It causes all the output to be magnified or reduced by a specified factor, which can be greater or less than 1.0. All 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 1.5
```

This example causes the output to be one and a half times as large as the default size. A magnification of around 1.2 is good for printing individual instrumental parts.

```
magnification 0.5
```

This example reduces the output to half size. Reduction is helpful when printing full scores. Magnification or reduction can sometimes be helpful in fitting a piece onto the paper, though it is more usual to use other directives such as **notespacing** and/or **layout** for this purpose.

### 8.1.63 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. They apply only when PMW is choosing its own slope.

### 8.1.64 Maxvertjustify

This directive is permitted only in the first movement. It 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**.

### 8.1.65 Midichannel

This directive allocates a MIDI voice and/or one or more PMW staves to a MIDI channel, and sets 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. There can be up to four arguments, of which only the first, the channel number, is mandatory. There are also **[midichannel]** and **[midivoice]** stave directives that 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 preceded by a sharp character, or the voice name, is given in quotes after the channel number. If a channel’s voice is specified more than once, the last specification overrides the earlier ones.

```
midichannel 1 "#57"
midichannel 2 "church organ"
```

There are sixteen MIDI channels, numbered 1–16 (but see the next section 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 follow. 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*. You can edit it to change or add names. The version supplied contains voice names taken from the *General MIDI* specification. Because there is some variation in some of the names, you can have more than one name for any given voice number, and there are some duplicates in the supplied file.

All staves are initially allocated to MIDI channel 1. This channel allocation can be changed by giving a list of staves to the **midichannel** directive, with or without a voice name.

```
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 is 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). Having set a voice in one movement, you can request ‘no voice setting’ in a subsequent movement by specifying an empty quoted string.

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 **midivolume** or **[midivolume]** directives for different voices, the resulting volumes do not correspond. To help balance voices, a volume value in the range 0–15 may be given after the voice name, preceded by a slash.

```
midichannel 1 "trumpet"/12 9
```

This example has the effect of reducing the volume of notes played via channel 1 by 12/15. This applies to all staves playing via the channel (in this example, just stave 9). The actual volume used for any MIDI note is 127 multiplied by the channel volume and the stave volume and divided by 225.

### 8.1.66 Midichannel settings for untuned percussion

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 argument of the **midichannel** directive 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 that specifies either the MIDI pitch number, or the instrument name. Note that the other string argument (the instrument name for a 'normal' channel) is placed immediately after the channel number, whereas this string argument comes last.

```
midichannel 10 5 "#60"
midichannel 10 6 "triangle"
```

These examples 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 | ...
```

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 (☞ 10.2.67).

### 8.1.67 Midifornotesoff

Notes that are suppressed in printed output by the use of **[notes off]** are by default also omitted from MIDI output. The heading directive **midifornotesoff** alters PMW's behaviour so that **[notes off]** no longer affects MIDI output.

### 8.1.68 Midistart

The **midistart** directive is followed by a list of numbers in the range 0–255. When a MIDI file is being created, the MIDI data defined by these numbers is written to the file after PMW's normal initialization. Each number defines a byte of MIDI data, and the whole sequence should consist of a sequence of MIDI events that are to be obeyed at time zero when the file is played. You must not attempt to supply any time information. PMW automatically arranges for all these MIDI events to be specified at time zero by inserting a zero byte before any value that is greater than 127. This feature can be used by those familiar with the MIDI coding to do things like changing the stereo position of the channels.

```
midistart 176 10 0  177 10 40  178 10 80  179 10 120
```

This example pans channels 1–4 evenly from full left (0) to nearly full right (120).

### 8.1.69 Miditempo

This directive is used to specify the tempo that is used when PMW creates a MIDI output file. It must have at least one 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 optionally be followed by pairs of numbers separated by slashes, to specify changes of tempo at particular bars.

```
miditempo 100 24/120 60/90
```

This example 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 (☞ 6.3). If no **miditempo** directive is present, a default tempo of 120 is used.

If there is more than one movement, the initial tempo specified in a **miditempo** directive carries over to the next movement (unless it contains its own **miditempo** directive, of course), but tempo changes within a movement do not. However, PMW cannot write more than one movement at a time to a MIDI output file (see the **-midimovement** command line argument in chapter 3).

### 8.1.70 Miditranspose

By default, PMW plays music exactly as written, except for recognizing transposing clefs. If the piece contains a part for a transposing instrument it will not play correctly. The **miditranspose** 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. **Miditranspose** 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.

```
miditranspose 1/-3
```

This example specifies that stave 1 is to be played a minor third lower than written. There is also a **[miditranspose]** stave directive that can be used to change the transposition part-way through a stave.

### 8.1.71 Midivolume

The **midivolume** directive is 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, to specify relative volumes for individual staves.

```
midivolume 6 2/15
```

This example specifies that stave 2 is to be played at maximum volume, whereas all other staves are to be played at volume 6. See also the **[midivolume]** stave directive.

### 8.1.72 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.

### 8.1.73 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 argument. A positive value moves the signature further away from the preceding bar line.

### 8.1.74 Musicfont

This directive is permitted only in the first movement. It specifies, as a string in quotes, the name of the music font to be used by PMW; its argument is a text string. The facility is intended for accessing new or alternative versions of the font. The default music font is PMW-Music.

### 8.1.75 Nobeamendrests

This directive, which has no arguments, can be used to cancel the effect of **beamendrests** in a previous movement.

### 8.1.76 Nocheck

This directive, which has no arguments, 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 (☞ 10.2.51).

### 8.1.77 Nocheckdoublebars

This directive, which has no arguments, instructs PMW not to check that the notes in bars that begin or end with a double bar line agree with the time signature.

### 8.1.78 Nokerning

This directive is permitted only in the first movement. It disables the use of kerning for text strings (☞ 6.14.11).

### 8.1.79 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. Individual occurrences can be suppressed by an option on the **[key]** stave directive that changes the key signature. If there is a change of both time and key at the start of a system, the warning at the end of the previous line is suppressed only if both warning signatures are being suppressed. If suppression of only one has been requested, both are nevertheless printed in these circumstances.

### 8.1.80 Nosluroverwarnings

This directive, which has no arguments, can be used to cancel the effect of **sluroverwarnings** in a previous movement.

### 8.1.81 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.

### 8.1.82 Notespacing

PMW contains a table of the minimum amount of horizontal space that follows 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 that define the space (in points) that must follow breves, semibreves, minims, crotchets, quavers, semiquavers, 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 specifies a multiplicative factor for each value in the table. This is requested by following the directive by an asterisk and a single number, or by two numbers separated by a slash.

```
notespacing *1.5
notespacing *3/2
```

Each of these examples specifies that the values in the note spacing table are to be multiplied by 1.5. If more than one multiplicative **notespacing** is present, their effect is cumulative, but a multiplicative **notespacing** is 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.

Changing the note spacing is one way of controlling the assignment of bars to systems and systems to pages. For example, if in the default state the last page contains only two bars, a small reduction in the note spacing 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, increasing the notespacing by a small amount can cause it to be filled out. You can also make temporary changes to the note spacing table for certain bars of the music only (☞ 10.2.56).

Another way of controlling the assignment of bars to systems is to use the **layout** heading directive (☞ 8.1.57). If you are using **layout**, it is not usually necessary to use **notespacing** as well.

### 8.1.83 Notime

This directive, which has no arguments, specifies that time signatures are not to be printed for the current movement. It does not stop PMW from checking the notes in a bar and complaining if there are too many or too few (see **nocheck** if you want to suppress this). **Notime** does not affect subsequent movements. See also **startnotime**.

### 8.1.84 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.

### 8.1.85 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. Individual occurrences can be suppressed by an option on the **[time]** stave directive that changes the time signature. If there is a change of both time and key at the start of a system, the warning at the end of the previous line is suppressed only if both warning signatures are being suppressed. If suppression of only one has been requested, both are nevertheless printed in these circumstances.

### 8.1.86 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.

### 8.1.87 Overlaydepth

If two or more character strings, all designated as overlay, are attached to the same note, 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.

### 8.1.88 Overlaysize

By default, text that 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.

```
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.

### 8.1.89 Page

This directive is permitted only in the first movement. 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 argument. There is also a second, optional argument that gives the increment by which page numbers are advanced.

```
page 3 2
```

This example 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.

### 8.1.90 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 arguments 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.

### 8.1.91 Pageheading

The **pageheading** directive defines text to be printed at the head of pages other than the first. Its arguments 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 section 6.1.6 and also the **[newmovement]** directive for discussions of headings when there is more than one movement in a file.

### 8.1.92 Pagelength

This directive is permitted only in the first movement. See section 8.1.60 (*Linelength and pagelength*) above.

### 8.1.93 Playtempo

This directive is a synonym for **miditempo**. It dates from the early days of PMW running on Acorn hardware, when playing was possible without using MIDI.

### 8.1.94 Playtranspose

This directive is a synonym for **miditranspose**. It dates from the early days of PMW running on Acorn hardware, when playing was possible without using MIDI.

### 8.1.95 Playvolume

This directive is a synonym for **midivolume**. It dates from the early days of PMW running on Acorn hardware, when playing was possible without using MIDI.

### 8.1.96 PMWversion

This directive checks that a given version of PMW is in use. It must be followed by a version number:

```
pmwversion 4.20
```

If the wrong version is used, a message is output and PMW stops.

### 8.1.97 Printkey

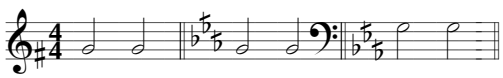
Some old music uses key signatures in which the accidentals are placed differently to the modern convention. For example, for a treble clef G major signature, the sharp is on the bottom space of the stave instead of on the top line. The **printkey** directive can be used to reproduce such usage. It can also be used to specify the appearance of half sharps and half flats in key signatures. The syntax is as follows:

`printkey` *<key>* *<clef>* `"`*<string>*`"`

There may be many occurrences of **printkey**, for different combinations of key signatures and clefs, in a single input file. The use of **printkey** affects only what is printed for the given key/clef combination. It has no other effect on PMW's behaviour. If there are two occurrences of **printkey** that refer to the same key and clef, the later one is used.

The string is normally a sequence of characters in the music font. The size is fixed at 10 points, scaled to the stave's magnification. You can change to other fonts by means of the usual escape sequences, but the size cannot be varied. The starting vertical position is the bottom line of the stave; you can use the up and down moving characters in the music font to position accidentals (or other characters) where you want them.

```
key G
printkey G treble  "\37\"
printkey E$ treble "x~\39\x~\191\ww\191\"
printkey E$ bass   "~\39\x~\191\ww\191\"
[stave 1 treble 1]
GG | [key E$] GG [bass 0] | [key E$] GG ||
[endstave]
```



In the above example, the G major key signature is a single sharp (character 37 in the music font), printed in the initial vertical position. For the E-flat treble clef signature, characters 120 (`x`) and 126 (`~`) are used to move up four and two points, respectively, so that the flat (character 39) prints on the middle line of the stave. Further appearances of `x` and `~`, and also character 119 (`w`), which moves down four points, are used to position the half flats (character 191) that follow. A similar string is used for the bass clef. Details of the music font characters are given in chapter 12.

### 8.1.98 Printtime

Time signatures are occasionally printed in unusual formats. This directive specifies how a given time signature is to be printed. It has the following syntax:

`printtime` *<time signature>* `"`*<top>*`"` `"`*<bottom>*`"`

There may be many occurrences of **printtime**, for different time signatures, in a single input file. 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.

### 8.1.99 Psfooting

This directive makes it possible to include raw PostScript in PMW output at the end of the first page. Unless you are a PostScript expert, this facility is not for you. The directive must be followed by a string in double quotes. If the first character of the string is '<' the rest of the string is taken as a file name from which to copy PostScript into the PMW output at the footing point. See section 3.5 for details of how included PostScript files are processed. 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 it is a literal string and backslashes are required in the PostScript output, they must *not* be doubled in the input.

### 8.1.100 Psheading

This directive makes it possible to include raw PostScript, in the same format as for **psfooting**, at the head of the first page.

### 8.1.101 Pslastfooting

This directive makes it possible to include raw PostScript, in the same format as for **psfooting**, at the end of the final page.

### 8.1.102 Pspagefooting

This directive makes it possible to include raw PostScript, in the same format as for **psfooting**, at the ends of pages other than the first.

### 8.1.103 Pspageheading

This directive makes it possible to include raw PostScript, in the same format as for **psfooting**, at the heads of pages other than the first.

### 8.1.104 Pssetup

This directive is permitted only in the first movement. It is used to include private PostScript, in the same format as for **psfooting**, at the end of the prologue that is output by PMW before the PostScript for the first page.

### 8.1.105 Rehearsalmarks

This directive controls the way rehearsal marks are printed. It has this syntax:

`rehearsalmarks` <*style*> <*size*> <*fontname*>

All three arguments are optional, except that if <*fontname*> is present, one of either <*style*> or <*size*> (or both) must precede it. The style must be one of the words 'boxed' (enclose in a rectangular box), 'ringed' (enclose in a ring), or 'plain' (do not enclose). If no word is given, there is no change of style. The size is the font size, and the third argument specifies the font to be used (☞ 6.12).

```
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.

### 8.1.106 Repeatbarfont

The font used for 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.

```
repeatbarfont 8 extra 4
```

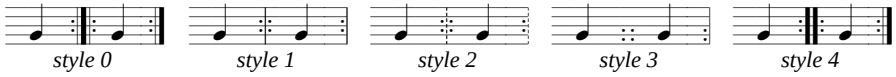
This example specifies the use of the fourth extra font, at an 8-point size. The default size is 10 points, and the default font is roman.

### 8.1.107 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. This 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 (unless at a bar line)
`4` as 0, but alternate amalgamated form

If style 2 is used at the start or end of a bar, you must use an invisible bar line if you want to prevent the dotted line being overprinted by a normal bar line. Style 4 is 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 just two thick lines between the dots, and this is what style 4 gives.

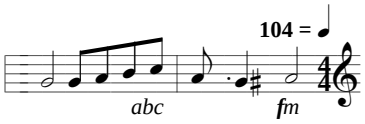



### 8.1.108 Righttoleft

The **righttoleft** directive causes PMW to print music from right to left instead of in the conventional left-to-right manner. This directive must appear in the first movement of a file, and it applies to all the movements that follow. The facility was ''bolted on'' to the existing PMW code, and as a result has some awkwardnesses, in particular in the way in which character strings are handled. It is also somewhat experimental and is likely to give strange results if some of the more complicated features of PMW are used. Nevertheless, **righttoleft** makes it possible to typeset music in a manner that is sometimes used in countries whose language is written from right to left. Although the music runs from right to left, the shapes of the notes, accidentals, and clefs are not altered.

Right-to-left printing is implemented by transforming the PostScript coordinate system so that the x-axis runs to the left and the origin is at the righthand side of the page image. Within the transformed coordinate system, fonts are themselves re-transformed so that their characters appear as normal. This means that characters in text strings still print from left to right; however, the positioning of text strings is adjusted so that they end rather than start at their normal position. There is a further complication if there are font changes within a text string. The substrings, taken in order in the input file, are printed from right to left, but within each substring, the characters run from left to right. Note the way that the strings in this example are rendered:

```
[stave 1 treble 1]
"\*c\\bf\104 = "/a/ts/u2
"m\bi\f" A #g. a- | "abc" c'-b-a-g-; G |
```




The first string, `"\*c\\bf\104 = "` is split into two substrings, the first being a crotchet character from the music font, and the second being the text `"104 = "` from the bold font. The `/ts` option specifies that the string is positioned at the time signature, so this is where the first substring is printed; the second string is then placed to its left. The second string, `"m\bi\f"`, also uses two fonts, and so the italic *m* appears to the right of the bold italic ***f***. The third string has no font changes, so the characters appear in order.

When right-to-left printing is enabled, right and left are interchanged in all the facilities for moving items horizontally, and for the left-centre-right feature of heading lines. For example, using `/r` on a string moves it left instead of right. In shapes defined by the **draw** directive, x-coordinates are inverted, with positive values moving left instead of right.

If any of the two-up printing styles is selected when **righttoleft** is enabled, the order of printing the pages on the sheets is reversed. This makes it possible to print correct right-to-left pamphlet-style pages for folding and binding.

### 8.1.109 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 **-f** command line option, as in this example:

```
*if chorus
  selectstaves 5-8
*fi
*if cello
  selectstave 4
*fi
```

Any tests that rely on a particular stave selection must follow this directive.

### 8.1.110 Sheetdepth, Sheetwidth, and Sheetsize

These three directives are permitted only in the first movement. They 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’, ‘B5’, or ‘letter’. 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 |
| letter | 612 | 792 | 500 | 670 |

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, or A5 pages can be printed two-up by specifying **-a5ona4**.

### 8.1.111 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.

```
shortenstems 4
```

This example allows 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).

### 8.1.112 Sluroverwarnings

When a line ends with a warning time or key signature, and there is a slur or tie that is continued from this line to the next, PMW does not by default draw the slur or tie over the warning. This directive requests it to do so.

### 8.1.113 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 that can be set by this directive. The default value is 0.7.

### 8.1.114 Startbracketbar

This directive applies only to the movement in which it appears; it affects the first system of the movement. It specifies a number of bars by which the joining brackets and/or braces that normally appear at the left-hand end are to be 'indented'. The second and subsequent systems are not affected. 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 section 6.11 for an example of the use of **startbracketbar**.

### 8.1.115 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*>

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 arguments 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), 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.

### 8.1.116 Startnotime

This directive, which has no arguments, applies only to the movement in which it appears. It causes no time signature to be printed at the start of the movement, 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.

### 8.1.117 Stavesize(s)

This directive specifies that certain staves are to be printed at different sizes to the rest. It is followed by pairs of numbers, separated by slashes. The first of each pair is a stave number, and the second is the size of the stave relative to the standard stave size.

```
stavesize 1/0.8
stavesizes 4/1.2 5/1.2 6/1.2
```

The first example specifies that stave 1 is to be printed at 0.8 times the normal size, and the second 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 (the instrument name) is not affected, and neither are bar numbers or

rehearsal marks. A size may be specified for stave zero if required. As no notes are ever printed on this stave, only text items are affected. Bar lines are printed thicker or thinner, as necessary, unless a fixed size has been specified with **barlinesize**. With varying barline thicknesses, it is conventional to break bar lines between staves of different sizes to avoid ugly joins.

### 8.1.118 Stavespacing

This directive controls the amount of vertical white space between staves. The distance between staves is measured from the bottom of one stave to the bottom 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.

```
stavespacing 50 1/54 3/60
```

This example 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 arguments 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.

```
stavespacing 1/36 4/50
```

This example ensures that staves 1 and 2 are nearer together than the default, at 36 points, and staves 4 and 5 are further apart at 50 points (assuming that all these staves are 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 argument to **stavespacing**, the second number specifies a *minimum* space *above* the given stave, and the third specifies the space below it.

```
stavespacing 1/46 2/50 3/50/48
```

This example 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]** (☞ 10.2.89).

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. However, 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.

### 8.1.119 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 need to have their stems lengthened in order to fit in the extra tails. The default setting for this directive is:

```
stemlengths 0 0 0 0 2 4
```

This specifies stem lengthening for demisemiquavers and hemidemisemiquavers of 2 and 4 points, respectively.

### 8.1.120 Stemswap

This directive is 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 section 9.8 (*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
```

The direction of the stem of a note on the stem swap level depends on the stem direction of the note to its left, viewing the part as one long stave (in other words, it depends on the previous note). If the previous note is a breve or semibreve, a notional stem direction is computed as if it were a shorter note.

```
stemswap right
```

This makes the stem direction depend on the next note to the right that is not also on the stem swap level. However, the search for the next note does not extend beyond the end of the bar. If the final note(s) of a bar are on the stem swap level, their stem direction is taken from the preceding note.

### 8.1.121 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** or defaulted. **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.

```
stemswaplevel 1/1 2/-1
```

This example requests that on stave 1, the swap level is moved to the third space instead of the third line, and on the second stave it is moved down to the second space.

### 8.1.122 Stretchrule

In release 4.22 of PMW there was a change to the horizontal spacing algorithm, affecting systems that have to be compressed to fit on the line. This is most commonly caused by the use of the **layout** directive (☞ 8.1.57). When bars are first formatted, notes may be moved apart in order to avoid underlay words crashing into each other; compressing the system horizontally later on, however, could still cause crashes. The new algorithm does some re-spacing when systems are compressed.

Users may have old files in which they have manually moved underlay syllables in order to overcome the problem in the old algorithm. The **stretchrule** directive allows you to turn off the new feature, for backwards compatibility. Its argument is an integer. There are three possible values, harking back to a similar change in the distant past:

0 Use the algorithm prior to release 3.35
1 Use the algorithm prior to release 4.22
2 Use the latest algorithm (default setting)

Values greater than 2 are treated as 2. This directive is permitted only in the first movement.

### 8.1.123 Suspend

This directive affects only the movement in which it appears. It specifies the suspension of certain staves from the beginning of the movement. It must be followed by a list of staves, in the same format as the **bracket** and **brace** directives.

```
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 (☞ 10.2.96).

### 8.1.124 Systemgap

The vertical distance between systems is measured from the bottom of the last stave of one system to the bottom of the first stave of the next system, and this distance can be specified by **systemgap**, which takes a single dimension as its argument. The default is 44 points, the same as the default spacing between staves in a system. Thus, by default, the entire output on a page is on evenly spaced staves when there is no vertical justification. When vertical justification is happening (☞ 8.1.51), the system gap is a minimum distance between systems; once the page layout is determined, the gaps are expanded so that the last stave of the last system on a page is exactly at the bottom of the page. The spacing between systems can be varied in the middle of a piece. See the stave directives **[sghere]** and **[sgnext]** in section 10.2.74.

### 8.1.125 Textfont

This directive is permitted only in the first movement. 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 argument 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 argument is the full name of the font, in double quotes.

```
textfont bold "Palatino-Bold"
```

This example changes the bold face font from the default (which is *Times-Bold*) to *Palatino-Bold*. An example that 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 6.14.7 for details of font-changing escape sequences. The capitalization of font names is important.

### 8.1.126 Textsizes

Text that is specified with 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.

```
textsizes 10.5 11 7.6 9/1.1
```

By default, ordinary text is printed using the first size specified, but 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 (☞ 9.9).

### 8.1.127 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.

### 8.1.128 Time

This directive applies only to the movement in which it appears. It sets a time signature for all the staves of the movement. 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.

### 8.1.129 Timebase

This directive can be used at the start of a new movement to cancel the effect of **notimebase** in the previous movement.

### 8.1.130 Timefont

The **timefont** directive is 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 that 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.

### 8.1.131 Timewarn

This directive can be used at the start of a new movement to cancel the effect of **notimewarn** in the previous movement.

### 8.1.132 Topmargin

See section 8.1.13 (*Bottommargin and topmargin*) above.

### 8.1.133 Transpose

This directive applies only to the movement in which it appears. It sets a transposition for the whole movement, and must be followed by a positive or negative number specifying the number of semi-tones of transposition up or down, respectively. If a transposition is also specified from the command line, the two values are added together. Section 6.10 gives more details about transposition.

### 8.1.134 Transposedacc

By default, PMW always prints an accidental on a transposed note if an accidental was present on the original, thereby preserving cautionary accidentals. If **transposedacc** is followed by the word ‘noforce’, it 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 also possible to force either behaviour for individual notes (☞ 9.6.7).

### 8.1.135 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 relevant key signatures in section 6.10. This list also applies when key or chord names in strings are being transposed. The default can be overridden by specifying:

`transposedkey` <*key1*> `use` <*key2*>

This means ‘if transposing a key signature yields <*key1*>, use <*key2*> instead’.

```
transposedkey G$ use F#
```

This example ensures transposition into F♯ instead of G♭. 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 has other uses when transposing music that 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.

### 8.1.136 Trillstring

When a trill is indicated for a note, the glyph *tr* is printed from the music font. The **trillstring** directive lets you change this for another character or characters.

```
trillstring "\it\tr"
```

This example replaces *tr* by the letters *tr*, printed in italic. The string may be preceded by a number, specifying the size of font to be used.

### 8.1.137 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 such as ‘bolditalic’ (☞ 6.12). It cannot be omitted. When this directive is not used, a 10-point roman font is used for printing triplet numbers.

### 8.1.138 Tripletlinewidth

This directive sets the width of lines used for the horizontal brackets of irregular note groups. The default width is 0.3 points.

### 8.1.139 Underlaydepth

If two or more character strings, all designated as underlay, are attached to the same note, 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 argument can be given to this directive for special effects, such as printing alternative words above a stave. However, this is probably easier to achieve using the overlay facilities. The depth parameters for underlaid and overlaid text are separate and independent.

### 8.1.140 Underlayextenders

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.

### 8.1.141 Underlaysize

By default, text that 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.

```
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.

### 8.1.142 Underlaystyle

By default, PMW centres underlay and overlay syllables under or over each note, respectively. There is a tradition, 'now frequently ignored' (Kurt Stone, *Music Notation in the Twentieth Century*), that 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, and style 1 sets multinote syllables flush left. When operating in style 1, individual multinote syllables can be centred by making use of the ^ character (☞ 5.3.3), 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.

### 8.1.143 Unfinished

This directive, which has no arguments, applies only to the movement in which it appears. It indicates that the music data supplied is not a complete movement. This has the effect of suppressing the solid bar line at the end. It is not necessary to specify **unfinished** if the movement ends with a double bar line.

### 8.1.144 Vertaccsize

The size of accidentals that are printed above or below notes (☞ 9.6.6) is controlled by this heading directive; the default size is 10 points, which causes them to be the same size as normal accidentals.

```
vertaccsize 9
```

This example causes them to be printed slightly smaller than the default.

# 9. Stave data

This is the first of two chapters in which we describe the format of the 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 as follows:

- A few common items that 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 next few sections.
- Textual items, such as ***f***, *a tempo*, etc., are coded as strings enclosed in double-quote characters, and are described in section 9.9.
- Other non-note items take the form of *stave directives*, enclosed in square brackets. There are several different formats for stave directives. They are described in alphabetical order in section 10.2.

Notes, rests and other items may be interspersed freely, as required. Space characters and line breaks can be used to separate items, in order to make the input easier to read, though they are not necessary. PMW makes no attempt to check on the musical sense of what it is asked to print, other than to check bar lengths. When there is more than one stave, the length of the notes in each bar must be the same for all staves. Also, the length of the notes in a bar must agree with the time signature, unless **nocheck** or **[nocheck]** has been used (☞ 5.2).

## 9.1 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 six different styles (☞ 8.1.7). 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.

The amount of horizontal space that is inserted after a bar line is controlled by the **barlinespace** directive. 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 (☞ 9.7.2).

### 9.1.1 Invisible bar lines

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 **barlinespace** directive, which controls the amount of space that is inserted after a bar line, also applies to invisible bar lines. Usually, the bars on either side of an invisible bar line are of abnormal length, so you need to turn off the bar length check 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.

### 9.1.2 Mid-bar 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 does not end the bar.

### 9.1.3 End of movement bar lines

Unless the **unfinished** directive (☞ 8.1.143) has been used, the end of a piece or movement is marked in the traditional manner with a thin bar line followed by a thick bar line. Occasionally it may be

useful to print such a bar line in the middle of a piece. This is notated by three vertical bars in succession.

## 9.2 Repeated bars

A bar that is repeated in the input need only be coded once. The appearance of a number enclosed in square brackets causes those items to the right of it in the bar, including the bar line, to be repeated that number of times. This facility is most commonly used for sequences of rest bars, but it can be used with any bar.

```
[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 all ten bars. There is danger of confusion between repeated bars and rehearsal marks. Accidental omission of the quotes from a numerical rehearsal mark such as `["42"]` can lead to some very strange effects.

**Warning:** Repeated input bars should not be used with multi-syllable underlay texts, because the syllables are apportioned to notes as they are read from the input, and the repeated bars are not re-read.

## 9.3 Repeated sections

The beginnings and ends of repeated sections of music are marked by the following 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. Several different printing styles of repeat mark are provided (☞ 8.1.107). First and second time bars are catered for (☞ 10.2.1). PMW does not normally end lines of music other than at the ends of bars. If a repeat occurs in the middle of a bar and you want to allow that bar to be split over a line break, you have to use an ‘invisible bar line’ (☞ 9.1.1). PMW 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] (:
```

This example causes the time signature to be printed first, followed by the repeat mark.

```
(: [time 4/4]
```

This example 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, the repeat sign is printed first on all staves.

## 9.4 Caesuras

A caesura (pause) in the music is shown in the input in very much the way it is printed, by two successive slashes.

```
c'B // r-c'- |
```

A caesura is normally printed as two sloping strokes through the top of the stave, but the **caesurastyle** directive can be used to obtain a single-stroke version.

## 9.5 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.

```
a b > c d e >
```

This example specifies a diminuendo hairpin that extends under the three notes C, D, and E. 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, 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. 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.

### 9.5.1 Horizontal hairpin positioning

By default, a hairpin starts at the left-hand edge of the first enclosed note, and ends at the right-hand edge of the last note, but there are some options that change this. 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.

```
>/h GAB >/h B
```

This example starts the hairpin halfway between G and A, and ends it halfway between the two Bs. Without `/h`, it would have started just before the G and ended just after the first B. The `/h` option always moves the start or end to the right, never to the left. The halfway distance is just a default; the option can be used more generally by following it with a number indicating the fraction of the distance that is required.

```
< GGG </h0.8 |
```

This example ends the hairpin 0.8 of the way between the last note and the bar line. Another option that adjusts the horizontal position of hairpins is `/bar`. If the character indicating the start of a hairpin is followed by `/bar`, 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`, the hairpin ends at the next bar line. The use of `/bar` overrides `/h`.

### 9.5.2 Horizontal hairpin adjustments

The hairpin characters can be followed by `/l` or `/r`, followed by a number, to move left or right from where the hairpin would otherwise appear. These qualifiers affect only the end of the hairpin at which they are specified.

```
</l5 a b c d </r5
```

This example stretches the hairpin horizontally by 5 points at each end. If `/l` or `/r` are given as well as `/h` or `/bar`, the effect is cumulative.

### 9.5.3 Vertical hairpin positioning

Hairpins are printed under the stave by default, but the **[hairpins]** directive (☞ 10.2.33) can be used to make them print above instead. When a hairpin's vertical position is not explicitly specified, it is determined by the notes under or above which it lies. However, the **[hairpins]** directive 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'. A fixed level above or below the stave can be specified by following `/a` or `/b` by a dimension.

```
>/a10 bc'eb > |  </b12 gfag < |
```

This example prints the hairpins with their ‘sharp end’ 10 points above the top of the stave and twelve points below the bottom of the stave, respectively.

### 9.5.4 Vertical hairpin adjustments

The hairpin characters can be followed by `/u` or `/d`, followed by a number, to move up or down from where the hairpin would otherwise appear. If `/u` or `/d` is given at the start of a hairpin, it causes the whole hairpin to be moved up or down.

```
a b </d4 c d <
```

This example prints a hairpin that is 4 points lower than the default position. If `/u` or `/d` are used at the end of a hairpin, they cause the end to be moved up or down relative to the start.

```
< abc </u10
```

This example specifies a crescendo hairpin that 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.

### 9.5.5 Split hairpins

If a hairpin is split over a line break, 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, ‘sl’ refers to the right-hand end of the first part of a split hairpin, whereas ‘sr’ refers to the left-hand end of the second part. To move the second part of a hairpin down by 10 points, you would use `/srd10` on the starting angle bracket, and `/d10` on the final bracket.

### 9.5.6 Hairpin size and line thickness

The width of the open end of an individual hairpin can be set by following the initial `<` or `>` character with `/w` and a dimension.

```
</w4 ga <
```

This example sets a width of 4 points. The default 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 used for drawing hairpins. The default thickness is 0.2 points.

## 9.6 Notes and rests

The information for a note consists of five parts, of which only the first two are mandatory. The parts 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, though spaces can be inserted to improve the readability. A sequence such as `abcd` is perfectly valid input for four notes. Spaces may not, however, appear within the encoding for a single note, except in the options part as specified below.

### 9.6.1 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.

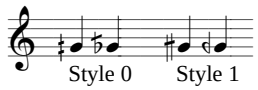
Accidentals are indicated by special characters before the note letter. The sharp character is the obvious one to use for indicating a sharp sign, but there are no obvious candidates for flats or naturals. Therefore two keys that are adjacent on most keyboards, and next to the sharp sign on some, are 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.

### 9.6.2 Half accidentals

PMW has some basic support for half sharps and half flats. Two different symbols for each are provided in the PMW-Music font; which to use are selected by the **halfsharpstyle** and **halfflatstyle** directives. A half sharp is notated as `#-` and a half flat as `$-`.




Using a half sharp or half flat just changes what is printed. MIDI does not support half intervals; if a MIDI file is generated, these accidentals are treated as full sharps or flats. A piece containing half accidentals can be transposed, but the result may be a bit odd.

### 9.6.3 Bracketted and parenthesized accidentals

Cautionary accidentals are sometimes printed in round brackets (parentheses) or square brackets. This is requested by following the accidental with a closing bracket of the appropriate type, as in this example:

```
#)a   $]b   ##)c
```

### 9.6.4 Invisible accidentals

When two or more parts are being overprinted on the same 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 (for example, `#?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 parenthesized.

### 9.6.5 Moved accidentals

Occasionally it is necessary to move an accidental sign to the left of where it would normally print. If the character `<` follows the accidental, it is printed 5 points to the left of its normal position, scaled to the stave size. Two successive `<` characters move 10 points left, and so on. Alternatively, a number may follow the `<` character to specify exactly how far left to move the accidental.

| | |
|---|---|
| `#<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 |

### 9.6.6 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. Instead, the letters `o` and `u` should be used to request that an accidental be printed over or under the note (the letters `a` and `b` cannot be used because they are note names).

```
#oa $ub
```

This example 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 these 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 them up or down by following `o` or `u` with `/u` or `/d` and a number. (In fact, `/l` and `/r` are also available, though unlikely to be useful.)

```
#o/u4c'  %u/d2f
```

If bracketed accidentals (☞ 9.6.3) are required above or below notes, the bracket must follow `o` or `u` and any up/down movement specification.

### 9.6.7 Transposed accidentals

Normally, PMW 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 with `^-`. If an accidental is actually necessary in the transposed music, it is not suppressed. 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 following the accidental with `^+`. 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.

PMW can be forced to print the accidental for a transposed note in a particular way (for example, 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 natural) |
| `^$` | print with a flat (‘black’ notes and B and E natural) |
| `^##` | 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 that is specified as `#^##G` is transposed up by one semitone, and would normally be printed as A-natural, it will now be printed as G-double-sharp. In the absence of any special indication, a subsequent note of the same pitch in the same bar will automatically print in the same way.

### 9.6.8 Rests

There are three ‘note letters’ that 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 rest bars, leading to the printing of ‘long rest’ bars where possible (☞ 9.6.15). When a rest bar is specified using S instead of R, it is always printed as an individual bar and never amalgamated. You can think of S as standing for ‘single’.

### 9.6.9 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 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. 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.

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.

```
[time 5/4] A+.+
```

This example 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 |

### 9.6.10 Chords

PMW can deal with certain kinds of chord, notated by enclosing a number of notes in parentheses. 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.

```
(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.), these must be specified on the first note. Chords consisting of quavers or shorter notes are beamed in the usual way (☞ 9.7); a semicolon after the closing parenthesis breaks all the beaming, whereas a comma breaks secondary beams only. If the chord is tied (☞ 9.6.29), the underline character that indicates a tie must appear after the closing parenthesis. Note that an underline character cannot be used for a short slur when chords are involved (as it can for single notes), because if two chords are joined by an underscore, all the notes in each that are of the same pitch are joined by a tie mark. The **[slur]** directive must be used to obtain just a single slur mark.

PMW automatically positions accidentals on chords unless one or more notes in the chord contains an explicit accidental positioning request (☞ 9.6.5). In this case, no automatic positioning is done; it is assumed that the user has positioned all the accidentals in the chord by hand.

### 9.6.11 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.

```
a>.   g6.2>..
```

In this example, the dot after the A is moved 5 points to the right and the double-dot after the g is moved 6.2 points. 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.

### 9.6.12 Vertical position of augmentation dots

The vertical position of dots for notes on lines can be controlled by the **[dots]** directive and the `\:\` note option (☞ 9.6.16). This option affects only notes that lie on stave lines. Normally dots for such notes are printed in the stave space above, but if the colon option 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.

```
[treble]     e.\:\     @ dot below
[dots below] g..\:\    @ dot above
```

The colon option can be used for 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, if it is on a line, always has its dot below, and the upper note, if it is on a line, always has its dot above. The `\:\` option does not affect notes in spaces, but it is sometimes useful to be able to move their augmentation dots into the space above. The option `\::\` achieves this; it has no effect if used on a note that lies 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 moves the dot on the lowest note down, and 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.

### 9.6.13 Notehead shapes and sizes

The shape of noteheads is controlled by the **[noteheads]** directive (☞ 10.2.54). Smaller than normal noteheads are used for grace notes, and for notes that appear between **[cue]** and **[endcue]**. In these cases, the entire note (head and stem) is printed at a smaller size. You can also request a small (cue-sized) notehead, without affecting any other part of the note, by means of the `\sm\` note option (☞ 9.6.16). This can be useful for indicating optional notes by means of a small notehead within a chord. This option affects only the notehead; the size of the stem, the position of any dots, and all other aspects of the note are not changed.

### 9.6.14 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, whereas 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. Rests that 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.

If a bar contains only whole bars rest on some staves and single notes on others, it sometimes looks better if the notes are also centred in the bar. This can be done by using the `\C\` option for the notes (☞ 9.6.16).

### 9.6.15 Repeated rest bars

When a sequence of bars contains only rests specified using R and Q (but not S) they are amalgamated into a single bar that is printed with a conventional 'long rest' sign and the number of bars printed above. Of course, this happens only if all the staves in the current printing have rest bars, typically 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 an amalgamated sequence, for example, to specify a tempo. If you do not want such a bar to be amalgamated, you must specify its rest using S instead of R.

```
[10]R! | "G.P." S! | [8]R! |
```

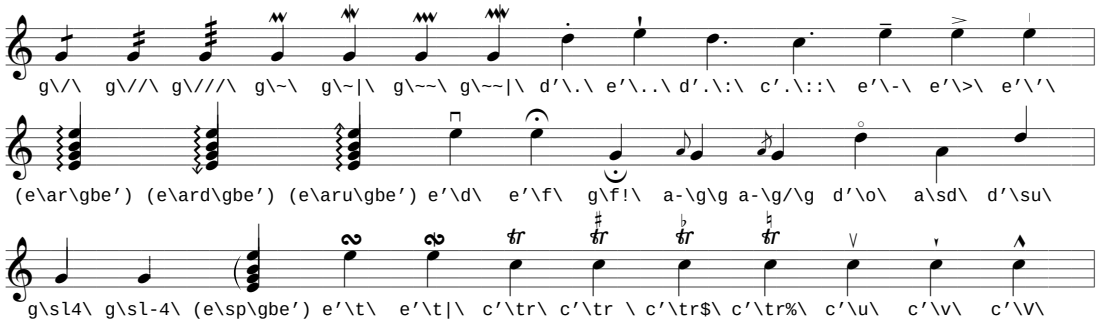
If `R` is used instead of `S` in this example, the last nine bars are printed as a single multi-bar rest when this stave is the only one selected for printing. As it stands, the G.P. bar is printed on its own, followed by an 8-bar multiple rest.

### 9.6.16 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, and so on. For many notes there are no such special marks and this part will not be present. When 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 (☞ 9.6.18) |
| `\.\` | staccato dot |
| `\..\` | staccatissimo mark |
| `\:\` | invert augmentation dot position (notes on lines, ☞ 9.6.12) |
| `\::\` | move augmentation dot up (notes in spaces, ☞ 9.6.12) |
| `\-\` | solid line emphasis mark |
| `\>\` | horizontal wedge emphasis mark |
| `\'\` | 'start of bar' accent |
| `\a`<*n*>`\` | accent number <*n*> (☞ 9.6.17) |
| `\ar\` | arpeggio mark |
| `\ard\` | arpeggio mark with downward arrow |
| `\aru\` | arpeggio mark with upward arrow |
| `\c\` | print on coupled stave (☞ 5.9.4) |
| `\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\ | do not print on coupled stave (☞ 5.9.4) |
| \m\ | masquerade note (☞ 9.6.23) |
| \o\ | small circle over note (harmonic) |
| \sd\ | force note stem down |
| \su\ | force note stem up |
| \sw\ | swap note stem direction in beam (☞ 9.7.5) |
| \sl<*n*>\ | lengthen stem by <*n*> points (☞ 9.6.22) |
| \sl-<*n*>\ | shorten stem by <*n*> points (☞ 9.6.22) |
| \sm\ | print with small (cue sized) notehead (☞ 9.6.13) |
| \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 (☞ 9.6.21) |




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 |

However, staccato and staccatissimo cannot be used together. Notes that are marked as grace notes can be of any length – they do not have to be quavers or semiquavers. 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 \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 sequences \c\ and \h\ are used to override the default note placing when coupled staves are in use (see **[couple]**). The single and double colon options are concerned with the vertical placement of augmentation dots (☞ 9.6.12).

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 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. (Do not confuse \C\ with \c\.)

### 9.6.17 General accent notation

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
10 staccatissimo mark

### 9.6.18 Position of accents and ornaments

By default, accents and the harmonic ring are printed on the opposite side of the notehead to the stem, but fermatas, trill signs, and other ornaments are printed above the note, independent of the stem direction. The addition of `!` to the 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. They are printed above the stave unless the **[bowing]** directive has specified othewise.

### 9.6.19 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.

```
a\./u4\ g\f/u10\
```

This example 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 (for example) a fermata: `g\/f\` is correct, but `g\f/\` causes an error, because it is taken as a fermata with an incomplete movement request.

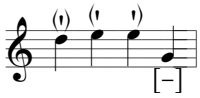
### 9.6.20 Bracketing accents and ornaments

Brackets are sometimes used to indicate editorial accents and ornaments. If there is a sequence of editorially marked notes, the sequence may be bracketed rather than each individual note. The following may be used after the specification of any accent or the specification for a fermata, mordant, trill, or turn, to indicate bracketing:

`/(` precede with an opening parenthesis
`/[` precede with an opening square bracket
`/)` follow with a closing parenthesis
`/]` follow with a closing square bracket
`/b` enclose in parentheses
`/B` enclose in square brackets

Here is a short example:

```
d'\../b\  e'\../(\  e'\../)\  g\-/B\ |
```



### 9.6.21 Repeated expression marks

If a sequence of notes are all to be marked with the same accent, this can be specified by giving the expression syntax for one note inside square brackets.

```
[\.\] a b c d
```

This example causes all the notes to be marked staccato. This feature is limited to accents and a few other expression marks. The only characters that may appear within backslashes in this context are:

| | |
|---|---|
| `.` | staccato |
| `..` | staccatissimo |
| `-` | horizontal bar |
| `>` | 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 |

Note that the movement and bracketing options that are available for expression marks on individual notes cannot be used here. To cancel a setting, two backslashes with nothing between them should be given between square brackets. Cancellation can also be carried out for an individual note by means of the note option letter `x`. In the following example, the note D is printed without a staccato dot.

```
[\.\] a b c d\x\ e f g [\\]
```

Expression/option items are processed from left to right. If there are two or more options being defaulted, `x` cancels them all, but one can be put back again afterwards.

### 9.6.22 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.

```
a\sl3.4\  b\sl-1.2\
```

This example 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 that 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.

### 9.6.23 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 by the form of the `m` in the same way as for normal notes. The only effect of masquerading is to substitute a different note for printing; the position of the note is not affected. 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.

| | |
|---|---|
| `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 |

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.

```
g-\M\ b- d'-
```

In this example, the first notehead is printed as a minim. 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. Note the use of an invisible quaver rest to make the item's length up to a crotchet.

### 9.6.24 Expression items on rests

Accent marks are not supported on rests, but pause marks (fermatas) are permitted. Other ornaments such as turns are allowed on invisible rests only. This gives a way of printing these marks on their own at positions in a bar that are not associated with printed notes.

### 9.6.25 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.

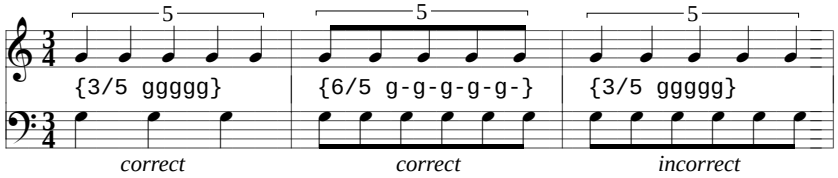
```
R\l4\
```

This example 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.

### 9.6.26 Triplets and other irregular note groups

In his book *Music Notation*, Gardner Read writes: ‘‘Notating unequal groups – triplets against duplets or quadruplets, quintuplets against triplets, and so on – is one of the musician's most perplexing problems.’’ PMW handles simple cases straightforwardly, but also has facilities for dealing with more general groups. One complication is in the choice of note-value to use for the irregular group. Gardner Read says: ‘‘The note-values of the extraordinary group are always determined by the note-values of the ordinary group against which they are set. […] When, however, the number of notes in the irregular group exceeds twice the number of note-values in the regular group, the uncommon group must employ the next smaller note-value.’’

This is not as simple as it sounds. Consider the case of five equal notes in a bar in 3/4 time. If the regular group is three crotchets, the irregular group should use crotchets because five is less than twice three; however, if the regular group is six quavers, the irregular group must use quavers, because an irregular group never uses longer notes than the regular group.




In PMW input, brace characters (curly brackets) are used to enclose a group of notes that is not a standard division of a longer note or group. In simple cases, the opening brace is followed by a single number to indicate the number of notes in the irregular group. If this number is omitted, the group is assumed to be a triplet.

| | |
|---|---|
| `{a b c}` | three crotchets in the time of two |
| `{2 g-a-}` | two quavers in the time of three |
| `{5 c-d-e-f-g-}` | five quavers in the time of four |

By default, PMW makes assumptions about the size of the regular group that is being subdivided, based on the number of subdivisions. However, in order to cope with more complicated cases, another parameter may also be set. The general form of an irregular note group is:

`{`*s*`/`*n* *<notes...>*`}`

The number of notes in the irregular group is *n*, and *s* controls the size of the group that is being divided. If *s* (and its slash) are omitted, as in the simple examples above, a default value is chosen that works well in most common cases:

- If *n* is a power of two (2, 4, 8, 16), *s* defaults to three. Thus, the example `{2 g-a-}` above is equivalent to `{3/2 g-a-}`.
- Otherwise, the default is two, so the example `{5 c-d-e-f-g-}` above is equivalent to `{2/5 c-d-e-f-g-}`.

A note in an irregular group can be longer or shorter than a normal note of the same type. For example, in a duplet, notes are longer, whereas in a triplet they are shorter. PMW modifies the lengths of irregular notes as follows:

(1) When *n* is less than 2**s*, the lengths of the notes in the irregular group are multiplied by *s*/*n*. In a triplet such as `{g-g-g-}`, where *s* and *n* have their default values of two and three, respectively, each quaver is shortened to 2/3 of its normal length, so three of them take up the time of two normal quavers.

(2) When *n* is 2**s* or more, but less than 4**s*, the lengths of the notes in the irregular group are multiplied by (2**s*)/*n*. Thus every note in the group `{2/5 c-d-e-f-g-}` is multiplied by 4/5, giving a total of four regular quavers, that is, two crotchets.

(3) When *n* is 4**s* or more, the lengths of the notes in the irregular group are multiplied by (4**s*)/*n*.

These rules are sufficient to handle most cases. For example, the group `{7 f-g-a-b-f-a-g-}` is a division of two crotchets into seven quavers. However, a division of three crotchets into seven quavers is notated on the music 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 time signature or the rest of the bar, and PMW is not capable of analysing bars in this detail. This is an example of a case where it is necessary to specify *s* explicitly; the code for dividing three crotchets into seven quavers is `{3/7 f-g-a-b-f-a-g-}`. Because of rule (2) above, this means that each note’s length is multipled by 6/7 instead of 4/7.

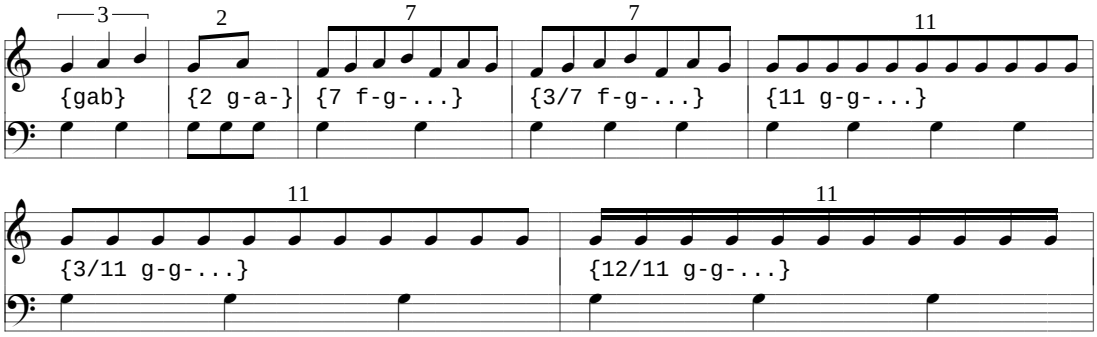
Published music is not always consistent in how some larger groups are notated. PMW can handle some of the alternative requirements. A division of three crotchets into 11 should use quavers, because 11 is less than 12, the number of semiquavers in three crotchets. The normal coding would be:

```
{3/11 g-g-g-g-g-g-g-g-g-g-g-}
```

However, if you want to notate three crotchets divided into eleven, but using semiquavers instead of quavers, you can use this:

```
{12/11 g=g=g=g=g=g=g=g=g=g=g=}
```

The length of each semiquaver is multiplied by 12/11, so the length of the group is 12 semiquavers, that is, three crotchets. The illustration below shows some of the examples discussed in this section, with regular crotchets on a second stave to show how the irregular groups are interpreted.




**Historical Note:** The code for handling irregular note groups was re-written for the 4.20 PMW release because it had got too complicated to easily understand. The new behaviour, as described above, gives the same results in most cases, but there may be differences for some less common irregular groups.

The old code also had an extra feature, which is retained for compatibility, but for which I cannot now dream up a useful example. If a minus sign is present before the slash that follows *s*, the size of the group that is being subdivided is halved. To divide three quavers into eleven using quavers, you could write this:

```
{3-/11 g-g-g-g-g-g-g-g-g-g-g-}
```

However, this is not standard notation; because there are more than 6 notes in the irregular group, semiquavers should normally be used.

### 9.6.27 Options for irregular note groups

By default, PMW prints the number for an irregular note group (for example, the '3' for a triplet) on the same side of the noteheads as the stems. The **tripletfont** directive is used to specify the size and type of font used. If the notes are beamed, just the number is printed; if not, a horizontal 'bracket' is printed as well. The whole mark can be moved, forced to be above or below the stave, and the horizontal bracket can be omitted. The mark may also be totally suppressed. The **[triplets]** directive can be used to set defaults for some of these options. For individual groups, 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 mark in the default state only. If **[triplets]** was used to suppress all irregular note group marks, `/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 move-

ments 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 mark 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.

```
{/ld10 dfa}
```

This example moves the left hand end down by 10 points. 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.

### 9.6.28 Beam breaking in irregular note groups

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.

### 9.6.29 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 (the two notes have the same pitch), the stem direction of the second note defaults to being the same as that of the first note, and if a MIDI file is being generated, the tie is honoured. The stem direction defaulting does not happen in the case of a short slur, when the two notes have different pitches. In both cases, explicit stem directions can be specified if the defaults are not what you want.

In contrast to single notes, when an underscore follows a chord it causes tie lines to be drawn only 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 (and when slurs cover more than two single notes). 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.

```
a_/a | a e'_/b | e'
```

In this example, the tie between the A notes is forced above, and the tie between the E notes is forced below. 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 that follows.

```
(ace)_/1a (ace)   (dfa)_/2b (dfa)
```

These examples show two different ways of specifying that one of the three tie marks is to be drawn above the noteheads, with the other two below.

### 9.6.30 Editorial and intermittent ties

Ties can be marked editorial, or printed as dashed or dotted, by means of the following qualifiers, which are the same options as for slurs:

| | |
|---|---|
| `/e` | editorial – a short line is drawn through the tie |
| `/i` | intermittent – that is, dashed |
| `/ip` | intermittent periods, that is, dotted |

### 9.6.31 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 when this is done. To help with the positioning of the ends of this kind of tie, tie marks are allowed to continue over rests (usually invisible ones).

```
(cde)_ | qq [notes off] (cde) [notes on] |
```

This example extends the ties to the position of the third crotchet in an otherwise empty bar.

### 9.6.32 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`, a glissando line is drawn between the relevant notes. If both a slur and a glissando mark are required, `/s` must be added. If the slur is being forced above or below with `/a` or `/b`, it is not necessary to use `/s`.

| | |
|---|---|
| `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. There are examples of the use of glissandos in sections 5.5 and 9.9.4.

### 9.6.33 Input short cuts

A number of 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, by using the letter `x` instead of a note letter. An optional number may follow `x` to indicate the number of repetitions. A beam break (☞ 9.7) 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 |
  ```

  A rest may intervene between the original note and its copy, but 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. 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`, 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 like this:

```
(d.fa) p-
```

The letter p can be followed by another p, possibly with a different note length, and x may follow p and *vice versa*. In earlier versions of PMW, they had to be in the same bar as the note or chord to which they referred; this is no longer the case.

For both x and p, accidentals are not reprinted within a bar, though they affect the pitch if a MIDI file is being generated. However, when x or p is used at the start of a bar, and the note or chord is not tied to the previous one (in the previous bar), the accidentals are repeated. If such a note or chord is tied to the previous one, no accidentals are printed, but if there is a subsequent use of p or x, the accidentals *are* then repeated, according to the usual notation convention.

```
a b #c_ | P p x
```

In this example, an accidental is not printed before the minim at the start of the second bar, but one is printed before the following crotchet. For technical reasons, x and p are not available after notes that print their accidentals above or below.

## 9.7 Note beaming

PMW makes no beaming breaking decisions based on 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.

### 9.7.1 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:

- `;` break all beams (semicolon)
- `,` break secondary beams only (comma)

PMW automatically beams across rests that are shorter than a crotchet, unless a break in the beam is specified. A beam breaking character must be entered immediately after the end of a note or after the closing parenthesis of a chord, without any intervening space. If the note or chord is tied (with an underscore character), the beam break must follow the underscore. If any other character follows a note or chord, no breaking happens. If the note or chord is the last of an irregular group (for example, triplets), the beam break may appear after the closing curly bracket. Three pairs of beamed quavers could be notated like this:

```
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.

```
g=-a=-b=-,2 c'=-d'=-e'=-
```

This example results in two solid beams, with a third that is broken in the middle. A value that is too large causes no beams to be broken. 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. After a secondary beam break, 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.

### 9.7.2 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.

```
[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

cannot be continued over a second bar line. However, as the example shows, a bar may have carried-over beams at both ends.

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 an explicit stem direction for the first note of the second bar.

### 9.7.3 Beaming across rests at beam ends

A recent innovation in notation is the continuation of beams over rests 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, but does not by default draw beams over rests at the ends of beams. If you want this to happen, you can request it by specifying **beamendrests** in the heading. 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.

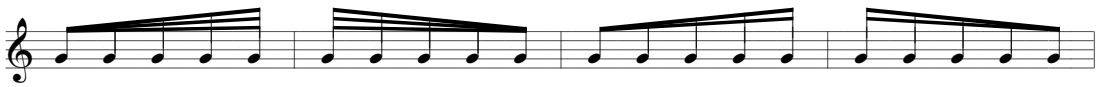
### 9.7.4 Accelerando and ritardando beams

In modern music, accelerandos and ritardandos are sometimes notated by beams that 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. 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.

```
[beamslope 0.1]  [beamacc]   {5/x g-g-g-g-g-} |
[beamslope -0.1] [beamrit]   {5/x g-g-g-g-g-} |
[beamslope 0.1]  [beamacc 2] {5/x g-g-g-g-g-} |
[beamslope -0.1] [beamrit]   {5/x g-g-g-g-g-} |
```

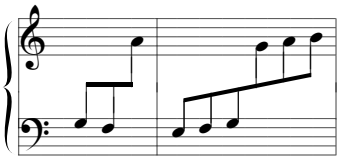


In some cases it is also necessary to move the beams away from the noteheads using the **[beammove]** directive.

### 9.7.5 Beams with notes on both sides

The stem direction that is determined for a beamed group by the rules described in section 9.8 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 when the staves are 'coupled' (see the **[couple]** directive). 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.

```
[stave 1 treble 1 couple down]
g`-f`-a-\sd\ | e`-\su\f`-g`-g-\sw\a-b- |
[endstave]
[stave 2 bass 0]
q-q-q- | Q! |
[endstave]
```



In this example, the default stem direction for the first beam is upwards because of the two low notes, but the third note has its stem forced downwards, so is printed on the other side of the beam. The `\su\` option in the second beam causes the first three notes to have their stems up, and the `\sw\` option forces 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, `\sw\` could not be used on the fourth, because the default direction is not known at the time `\sw\` it is processed (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. The **[beamslope]** and **[beammove]** directives, which adjust the slope and vertical position of a beam, can be used as for any other beam. 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.

## 9.8 Stem directions

This section documents the default rules for choosing a stem direction for a note or a chord. Some variation in the rules can be made by means of the **stemswap** heading directive (☞ 8.1.120). The 'stem swap level' is normally the middle line of the stave, but can be changed by the **stemswaplevel** directive (☞ 8.1.121).

### 9.8.1 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 that is furthest 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.

### 9.8.2 Rules for non-beamed notes and chords

These rules are given in order of priority. ‘The previous note’ includes the last note of a previous beamed group, if relevant. What happens to notes at the stemswap level (rules N5 and N6) can be changed by use of the **stemswap** directive (☞ 8.1.120).

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, that is, the previous note is followed by an underscore and has the same pitch, 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 it is the first in the bar, or if all preceding notes in the bar 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.

### 9.8.3 Rules for beamed groups

B1. If the stem direction of the first note in the group is forced by N1, N2, or N3 above, 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 is no previous note, 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 (☞ 9.7.5).

## 9.9 Text strings in stave data

Section 9.12 gives details of the special facilities that are applicable only to underlay or overlay text, that is, the sung words (lyrics) in a voice part. This section applies to text in general, with some particular features that are relevant only for non-underlay/overlay text. 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 (that is, not underlay, overlay, or figured bass) text. Rehearsal marks are a special form of text and are specified in a slightly different manner (☞ 9.11).

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.

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 (☞ 6.14). In particular, within any text string, the font can be changed by the use of the appropriate escape sequences. 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, midway to the next stave | |
| /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 | |
| /c | centre the text | |
| /e | align end of text | |
| /nc | do not centre | |
| /ne | do not align the end | |
| /box | print enclosed in a box | |
| /ring | print enclosed in a ring | |
| /rot<*n*> | rotate by <*n*> degrees | |
| /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. This feature is useful when a repeated string with a movement option is defined as a macro, but in some instances needs further adjustment. These two examples have exactly the same effect:

```
"allargando"/u6/d2
"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, /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 described in section 9.12.

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 the normal /b position would be. This is useful when printing dynamic marks in keyboard 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 the string is forced to the overlay or underlay level by /ol or /ul, or to an absolute position by /a<*n*> or /b<*n*>.

```
"at underlay level"/ul
"six points above the stave"/a6
"twenty points below the stave"/b20
```

If two or more strings precede the same note, the default vertical position for the second and subsequent strings is an appropriate distance above (for text above the stave) or below (for text below the stave) the previous string. The level of any string can always be adjusted by the use of `/u` or `/d`.

### 9.9.1 Horizontal alignment

The alignment of underlay and overlay strings is described in section 9.12. Any other string is printed by default 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, 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.

### 9.9.2 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.

### 9.9.3 Text sizes

The `/s` option refers to the sizes of text defined by the **textsizes** heading directive (☞ 8.1.126); `/s1` specifies the first size, `/s2` specifies the second size, and so on.

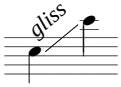
```
"Some text string"/s2
```

This example uses the second size defined by **textsizes**. By default, text is printed in the first size, unless it is underlay, overlay, or figured bass, which have their own default sizes (set by the **underlaysize**, **overlaysize**, and **fbsize** directives). The `/s` option can, however, be used with underlay, overlay, and figured bass text to specify a non-default size for an individual string.

### 9.9.4 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.

```
"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.

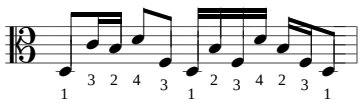
### 9.9.5 PostScript text

If the /ps qualifier appears on a text string, the contents are assumed to be raw PostScript that is to be inserted at the point where a text string would have been output. This facility is provided for PostScript experts; it is not likely to be of interest to most users. 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.

## 9.10 Fingering indications

The small caps feature of text strings is useful for selecting a smaller than normal font for printing fingering indications. Alternatively, a specific font size can be defined, and used with the \s text option. The music font contains the special characters ♀ and ♂ for indication the use of the thumb in cello parts. The use of macros is suggested when setting music with lots of fingering. Note the use of the /c option in this example, to centre the text below each note:

```
*define 1 "\rm\\sc\1"/b/c
*define 2 "\rm\\sc\2"/b/c
*define 3 "\rm\\sc\3"/b/c
*define 4 "\rm\\sc\4"/b/c
[stave 1 alto]
&1 d- &3 c'= &2 b=; &4 d'- &3 f-;
&1 d= &2 b= &3 f= &4 d'=; &2 b= &3 f= &1 d- |
```




## 9.11 Rehearsal marks

Rehearsal marks 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 printed in bold type and enclosed in a rectangular box. 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 at all. If necessary, a rehearsal mark can be moved up, down, left or right, in the same manner as other text.

```
["A"/u2]
```

This example moves the mark two points upwards. Normally, a rehearsal mark 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 mark is given in the middle of a bar, it is aligned horizontally with the next note, exactly as for other text.

Rehearsal marks 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, the rehearsal marks should be specified on stave 0 (☞ 6.15), so that they are always printed above the top stave, whichever staves are selected for printing.

## 9.12 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. 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.

### 9.12.1 Underlay syllables

Underlay can be input one syllable at a time, each syllable preceding the note to which it refers. This gives the maximum possible control, because 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 and/or hyphen characters, it is automatically split up by PMW and allocated to the notes that follow. Rests are excluded from this process (with one exception, which is described in section 9.12.5). As a simple example of this facility, this is an appropriate way to start the British National Anthem:

```
"God save our" g g a |
```

Here, each space delimits a word, and each word is associated with one 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 |
```

PMW prints one or more hyphens, depending on the distance between the syllables. 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.

PMW does not check that the number of syllables matches the number of notes, except that it warns if text is 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 syllables 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. The width of a printed space is half the size of the font.

```
"God# save #our" g g a |
```

If the default, 10-point font is in use, this example prints ‘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, only those characters to the left of it are counted when the string is being centred. The circumflex itself is not printed.

```
"Glory^#be#to#Thee, O God." G+ g #F
```

In this example, 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, 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 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.

```
"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 that 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 (☞ 8.1.34). By default, PMW centres all underlay and overlay syllables at the position of their respective notes. The **underlaystyle** directive (☞ 8.1.142) 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 |
```

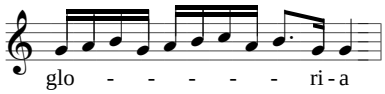
The vertical distance between verses can be altered by means of the **underlaydepth** and **overlaydepth** directives, which control independent values. For overlay, the second verse is printed above the first one, and so on. If any positioning qualifiers are specified on an underlay input string (`/u`, `/d`, `/l`, or `/r`), 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.

The multi-syllable underlay feature in PMW is just 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. |
```

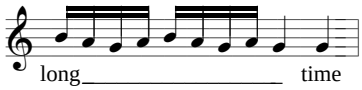
If an underlay string ends with a hyphen, the equals characters can be omitted; PMW automatically prints a sequence of hyphens up to the next underlay syllable. This can be useful when syllables last for many notes, for example:

```
"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.

```
"long=" b=a=g=a=; b=a=g=a=; "= time" g g
```




If there are more notes on the stave, but no more words, a syllable consisting of just a # character can be used to stop PMW drawing an extender line further than is required.

**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 just duplicates the bar in which it appears, without reading any more notes.

### 9.12.2 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.

### 9.12.3 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.

### 9.12.4 Underlay and overlay spreading

PMW spreads 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 treated as printing characters when examining the available space. The minimum space allowed between syllables is one space character in the appropriate font.

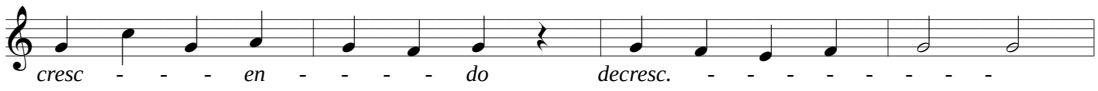
There is a heading directive, **nospreadunderlay**, which disables this facility for both underlay and overlay, and it is recommended that those who place great importance on the spacing of notes should use it. The automatic facility is intended as an insurance for less demanding users against the occasional 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 usually gives better results.

**Warning:** If use of the **layout** heading directive (☞ 8.1.57) causes the bars in a system to be horizontally compressed in order to fit them on the line, underlaid syllables may be forced into each other, even though they were originally separate. Although some re-spacing is done after a sufficiently large compression, in order to mitigate this problem, it is best to avoid settings of **layout** that cause compression if possible.

### 9.12.5 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 |
```



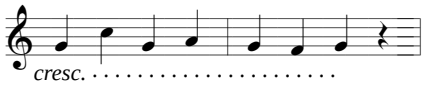

PMW supports multiple verses, so there is no difficulty in mixing this kind of usage with real vocal words, though usually the vocal line would be printed as underlay and the other text as overlay. Underlay and overlay syllables cannot normally be associated with rests, but because a final empty syllable 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 is associated with the rest rather than the following note. This exception applies only to strings consisting of a single # character.

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 in these cases. If a second string is provided as an option to an underlay or overlay string, separated by a slash, it is used instead of hyphens between 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.

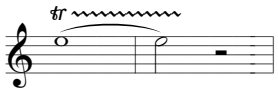
The default font for the second string is the default underlay or overlay font, as appropriate, and the default size is the size of the first string. However, the second string may be followed by `/s` and a number to specify a different size. The second string may also be followed by `/u` or `/d` to specify that it is to be moved up or down, relative to the following syllable. In this example, a space and a full stop is used as the repeating string, and it is moved up so as to be approximately at the middle of the letters.

```
"\it\cresc.-"/ul/" ."/u1.5  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.

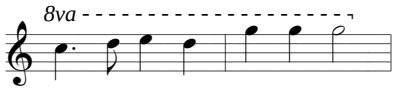
```
"\*136\-"/ol/"\*96\" E'+_ | "#"/ol/r8 E'R |
```




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 wiggly line covers 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 the next example.

The conventional octave mark of *8va* followed by a line of dashes can be printed using an overlay string. 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 is available. 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 the 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 that can be used for printing the ‘jogs’.

```
*define s8 "\it\8va-"/ol/" -| \mf\\159\"/u0.3
*define e8 "#"/ol/r8
&s8 c'.d'-e'd' | g'g' &e8 G' |
```




In this example, the macros **s8** and **e8** contain the strings needed to start and end an *8va* mark, respectively. Notice that **e8** is used before the final note under the mark. The repeated character string is a space and a hyphen (specified before the vertical bar), and at the end, a space followed by character 159 from the music font is printed.

One further feature is available to cope with repeated strings that extend over the end of a music system. If yet 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
```

It is assumed that size 2 is suitably defined using the **textsizes** directive.

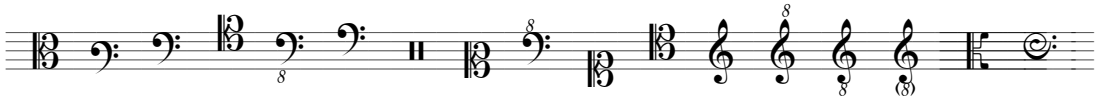
# 10. Stave directives

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. There are three other items that can appear between square brackets: repeated expression marks (☞ 9.6.21), repeated bar counts (☞ 9.2), and rehearsal marks (☞ 9.11).

## 10.1 Clef directives

Clefs are specified by directives that are the names of the clefs. There are two different styles for C and F clefs, controlled by the **clefstyle** directive (☞ 8.1.24). The following clefs are available:

| | |
|---|---|
| `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 |



The last two clefs shown are the alternative forms for C and F clefs. If a clef directive is given without an argument, no change is made to the current default octave. However, the directive name may be followed by a number to indicate a new setting for the current octave.

```
[treble 1]
```

This example sets the default octave to start at middle C. 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.

## 10.2 Alphabetical list of stave directives

The stave directives are now described in alphabetical order.

### 10.2.1 [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 mark for the final one is terminated by the appearance of the directive **[all]**.

```
[1st] g a b g :) | [2nd] g a b c' | [all] b a g f
```

There is an example of the output in section 5.6. More than one bar of music may appear between the items. The **[all]** directive is not used if the piece ends with the second time bar. Often these marks are printed above the top stave of a score only. If parts are to be extracted from a score, first and second

time marks should be specified on stave 0 (☞ 6.15), so that they will be printed above the top stave, whichever staves are selected for printing.

It is possible to specify vertical movements for 1st and 2nd time bar marks, to cope with unusual cases. This is done by entering `/u` or `/d`, followed by a number, in the directive. The left-hand ends of these marks can also be moved left and right by means of `/l` and `/r` qualifiers.

```
[1st/u4]
```

This example specifies a first time bar whose mark is to be 4 points higher than it would be by default. PMW normally puts the second time mark at the same level as the first, unless high notes in the second time bar force it upwards. More than one of these marks can be given in the same bar.

```
[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 number. It is also possible to change the text that is printed by supplying a text string after a slash.

```
[1st/"primero"]
```

If there are several marks in the same 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.

### 10.2.2 [All]

See the immediately preceding section.

### 10.2.3 [Alto]

This specifies a C clef with its centre on the third stave line (☞ 10.1).

### 10.2.4 [Assume]

When an overprinted stave contains a sequence of skipped bars (see **[skip]**), the clef, key signature, or time signature for its partner stave may have changed before the skipping stave resumes. The **[assume]** directive can be used to set these things without causing anything to be printed.

```
[skip 60] [assume bass 0] gbc |
```

This example 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.

### 10.2.5 [Baritone]

This specifies an F clef based on the third stave line (☞ 10.1).

### 10.2.6 [Barlinestyle]

This directive must be followed by a number, and it sets the bar line style for subsequent bar lines in the stave (☞ 8.1.7).

### 10.2.7 [Barnumber]

The heading directive **barnumbers** (☞ 8.1.9) is used to request automatic bar numbering. The stave directive **[barnumber]** is used to control the printing of numbers for individual bars. If **[barnumber]** appears without any arguments, 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.

```
[barnumber/l10/d5]
```

This example 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', no bar number is printed for the current bar, even if the heading directive implies there should be one.

### 10.2.8 [Bass]

This specifies a bass clef, that is, an F clef on the fourth stave line (☞ 10.1).

### 10.2.9 [Beamacc]

This directive causes the next beam to be drawn as an accelerando beam (☞ 9.7.4).

### 10.2.10 [Beammove]

This directive, which takes a single number as its argument, 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, because it is not always clear which notes in the beam are those whose stems control the beam position.

### 10.2.11 [Beamrit]

This directive causes the next beam to be drawn as an ritardando beam (☞ 9.7.4).

### 10.2.12 [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 argument a number specifying the slope of the next beamed group on the current stave.

```
[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, this 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 come into play again and a smaller slope is chosen.

### 10.2.13 [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.

```
[bottommargin 30]
```

This example, which may appear in any bar on the page, sets the margin for that page to 30 points.

### 10.2.14 [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'.

### 10.2.15 [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]**.

### 10.2.16 [Cbaritone]

This specifies a C-clef on the 5-th stave line (☞ 10.1).

### 10.2.17 [Comma]

The **[comma]** directive inserts a comma pause mark above the current stave.

### 10.2.18 [Contrabass]

This specifies a bass clef with a little ‘8’ printed below it (☞ 10.1).

### 10.2.19 [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 (☞ 6.15) is printed at the level of the current stave.

```
[copyzero 4]
```

This example 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 marks are specified in stave zero, and there is a need to adjust their height for certain staves, it should be noted that the marks 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).

### 10.2.20 [Couple]

A single music 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. If **[couple up]** is given on a bass clef stave, it 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 bass clef stave below, and there is also **[couple off]** to terminate 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 5.9.4.

**Warning 1:** Coupling only works properly if the upper stave is using the treble clef and the lower one is using the bass clef.

**Warning 2:** 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.

Occasionally it is desirable to cause individual notes that would not normally be printed on the coupled stave to be moved onto it. A notation for this is provided in the form of the `\c\` note option.

```
[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. When coupling is in force, there is a note option `\h\` (for ‘here’) that prevents a note that would normally move onto the other stave from doing so.

### 10.2.21 [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]  @ cue bar with halved note spacing
  "[flute]"/a      @ print above stave
  g a-g-f-e- e
  | [ns]           @ restore note spacing 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. (Another way of handling optional notes is to use the `\sm\` note option ☞ 9.6.13.) When cue notes are dotted, the dots are spaced horizontally in proportion to the size of the cue notes. However, when printing small optional notes with full-sized notes directly 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]**, which increases the space between the cue notes and their dots.

### 10.2.22 [Deepbass]

This specifies an F clef based on the fifth stave line (☞ 10.1).

### 10.2.23 [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 of an individual note’s dot can be overridden by means of the `\:\` note option (☞ 9.6.16).

### 10.2.24 [Draw]

The **[draw]** directive is described in chapter 7.

### 10.2.25 [Endcue]

See **[Cue]** (☞ 10.2.21) above.

### 10.2.26 [Endline]

See **[line]** (☞ 10.2.38) below.

### 10.2.27 [Endslur]

See **[slur]** (☞ 10.2.76) below.

### 10.2.28 [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.

### 10.2.29 [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 marks that PMW does not know about. The **[ensure]** directive provides this facility. If the requested amount of space is not available between the previous note (or the start of the bar) and the next note (or the end of the bar), a suitable amount of space is inserted.

```
G [ensure 32] G
```

In this example, if this is the only stave, 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 may lie between it and the previous note.

### 10.2.30 [Fbfont]

The default typeface for figured bass text is roman. It can be changed for an individual stave by means of the **[fbfont]** directive. This directive takes as its argument one of the standard font names.

```
[fbfont extra 3]
```

This example assumes that the third extra font has been suitably defined for use in figured bass text. In any given text string it is always possible to change typeface by using the appropriate escape sequence.

### 10.2.31 [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 **textsizes** heading directive. If this directive is not used, the size used is the one set by the **fbsize** heading directive (which is a different parameter from any of the sizes set by **textsizes**). **[Fbtextsize]** is normally needed only if you want different sizes of figured bass text on different staves.

### 10.2.32 [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 (☞ 8.1.47).

```
[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.

### 10.2.33 [Hairpins]

Hairpins (☞ 9.5) are normally printed below the stave. The **[hairpins]** directive is provided for changing this for an individual stave. 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 still. 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.

```
[hairpins above 10]
```

This example specifies that the ‘sharp end’ of hairpins should be positioned 10 points above the top of the stave. Individual hairpins can be moved from this position by the normal `/u` and `/d` qualifiers on the angle bracket characters that specify hairpins. 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.

```
[hairpins below -4]
```

After this example, all hairpins are positioned as if they were followed by `/d4`. Note the distinction between these two directives:

```
[hairpins above 8]
[hairpins above +8]
```

The former causes all hairpins to be printed 8 points above the stave, whereas the latter adds 8 points to whatever position PMW computes from the notes under the hairpin.

### 10.2.34 [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. The default is 7 points.

### 10.2.35 [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 (☞ 10.1).

### 10.2.36 [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, not complete settings. Its effect lasts only until the end of the current movement. The directive name 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 the page depth, and vertical justification is not wanted, **[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.

```
[justify -right -bottom]
```

This example might be used in the last system of a piece if the last line and the last page are both too short to be sensibly justified.

### 10.2.37 [Key]

This directive specifies a change of key signature. If this 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.

```
[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.

```
[key C key A]
```

This example 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] (:
```

This example causes the key signature to be printed first, followed by the repeat mark.

```
(: [key G]
```

This example 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, the repeat sign is printed first on all staves.

### 10.2.38 [Line]

There are a number of situations where it is required to draw a straight line above or below a sequence of notes, with or without small 'jogs' at the ends. The directive **[line]** works exactly like **[slur]** (☞ 10.2.76), except that what is drawn is a straight line with a vertical 'jog' on each end, giving a sort of horizontal or near-horizontal bracket. The angle of the line depends on the 'shape' of the note sequence that is above or below. 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 `/i` and `/ip` qualifiers are available, and cause a dashed or dotted line to be drawn, respectively. The `/co` and `/ci` options affect the length of the 'jogs'; however, the other options starting with `/c`, which for a slur move the Bézier curve control points, are ignored for lines. 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. Unlike slurs, lines are by default always positioned above or below the stave itself, never actually overprinting it. However, like slurs, they 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. The **[linegap]** directive can be used to leave gaps in lines and cause drawing or printing to take place in the gap. For handling complicated overlapping lines, there is an **[xline]** directive that corresponds to **[xslur]**, and lines can also be 'tagged' like slurs (☞ 10.2.83). The `/=` option that is used in **[endslur]** to identify tagged slurs is also available in **[endline]** for tagged lines.

### 10.2.39 [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, unless there is an associated text string (see below). The width is measured along the line. If a gap passes either end of the line, the ending jog is never drawn, even if specified.
- `/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. 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.
- `/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. **Warning**: 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. 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. The drawing code is obeyed with the origin of the coordinate system set to the centre of the gap, and the variables **linegapx** and **linegapy** containing the coordinates of the start of the right-hand portion of the line relative to this origin. For a horizontal line, **linegapx** is half the width of the gap, and **linegapy** is zero.

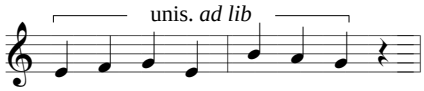
- `/"`*text*`"` associates the text with the gap.

  ```
  [linegap/"\it\ad lib."]
  ```

  This example prints the italic text *ad lib.* in the gap. The default font 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 line. 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.

A drawing or text string can be associated with the start or end of a line by using `/h0` or `/h1`, respectively. To associate a drawing or string with a particular point on a line, but without leaving a gap, `/w0` can be used. 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. Here is an example of a gap that is used to print some text in the middle of a horizontal line:

```
[line/a/h linegap/h/"unis. \it\ad lib"] efge | bag [el] r |
```



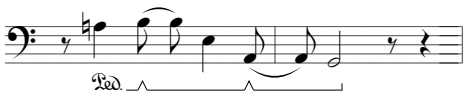

Because the `/h` option is used, the **[linegap]** directive can be given right at the start of the line. Do not confuse `/h` as used with **[line]**, where it means *horizontal*, with `/h` as used with **[linegap]**, where it means *halfway*. This latter usage was chosen to be similar to the `/h` option on hairpins. Another use of the **[linegap]** facility is for drawing conventional piano pedal marks. 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 (that is, 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 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.

### 10.2.40 [Mezzo]

This specifies a C clef with its centre on the second stave line (☞ 10.1).

### 10.2.41 [Midichannel]

The **[midichannel]** directive can be used to change the MIDI channel that the current stave uses, from the next note onwards. It can also be used to change the voice allocation of the new channel at the same time.

```
[midichannel 5]
[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, separated by a slash.

```
[midichannel 1 "trumpet"/12]
```

See the **midichannel** directive for how to set up channels at the start of a piece.

### 10.2.42 [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. See **[printpitch]** for an alternative way of handling MIDI untuned percussion.

### 10.2.43 [Miditranspose]

The **[miditranspose]** directive can be used to change the playing transposition of a stave, which is initially set from the **miditranspose** heading directive. The value given in **[miditranspose]** 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 at the correct pitch. Changes made by **[miditranspose]** do not persist beyond the end of the movement.

### 10.2.44 [Midivoice]

The **[midivoice]** directive can be used to change the MIDI voice without changing the channel.

```
[midivoice "french horn"]
```

Note that this will affect any other staves that are using the same channel. If **[midivoice]** with different arguments appears in multiple staves that are using the same channel, the result is undefined. If you want different staves to play using different voices, you must allocate them to different channels, using either **midichannel** at the start of the piece, or **[midichannel]** on each stave.

### 10.2.45 [Midivolume]

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.

### 10.2.46 [Move]

If the **[move]** directive is followed by a single number, it 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 that number of points, without affecting the position of anything else in the bar, except slurs or ties that are attached to a moved note and any accents or ornaments that a note may have. If the number is positive, movement is to the right; if negative, it is to the left. Certain items can also be moved vertically, by specifying a second number after a comma. The second argument may also 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.

```
[move -2,4 treble]
```

This example prints a mid-stave clef two points to the left and four points higher than normal. Vertical movement does not apply to notes and rests, and is ignored if specified. (See section 9.6.25 for ways of moving rests vertically.) 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]** that scales in both directions. 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, notes, and rests (for the last two, horizontal movement only). If **[move]** directive is present in the first bar of a sequence of rest bars and 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]**.

### 10.2.47 [Name]

The **[name]** directive has two entirely separate functions. If its arguments 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.

```
[stave 1 treble 1]
*if score
[name "Flute" "Fl."]
*fi
```

This example prints the name in the score, but not in the part. In this form, the arguments for **[name]** are exactly the same as the text and drawing arguments of the **[stave]** directive. The second form of **[name]** is used 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*. In this usage, its argument 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, a stave might start with:

```
[stave 2 "Alto" "A" "Alto I"]
```

By default in the first system the stave is labelled ‘Alto’, and in all subsequent systems it is by default labelled ‘A’. However, if **[name 3]** is used at any point, the system in which it appears, and any subsequent ones, 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.

### 10.2.48 [Newline]

The directive **[newline]** can be used to force a new line (system) of music to be started at a particular point. It should always be at the start of a bar, but need appear in only one stave. If a stave is not selected for printing, however, an appearance of **[newline]** within it is ignored. See also the **layout** heading directive.

### 10.2.49 [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 an argument, 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 two systems into the current page, because a single line of music at the bottom of a page does not 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.

Page headings specified in the new movement completely replace those set up in the previous movement, but if nothing is specified, the old page headings continue. This means that, for example, if page numbers are specified in the first movement by a page heading, they are printed on all subsequent pages, including pages that are the start of new movements. Section 6.1.6 contains more details about the interaction of headings and footings with new movements.

When a new movement starts at the top of a page, any page headings that are in force (either carried over or newly specified) are printed, in addition to the headings for the movement. Sometimes, however, it is required to suppress these page headings, 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.

```
[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 by specifying **[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 replaced by a new **lastfooting** directive in the new movement.

Another possible form of the directive is **[newmovement thisline]**, which is useful in some 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.

### 10.2.50 [Newpage]

The directive **[newpage]** can be used to force a new page of music to be started at a particular point. It should always be at the start of a bar, but need appear in only one stave. If a stave is not selected for printing, however, an appearance of **[newpage]** within it is ignored. See also the **layout** heading directive.

### 10.2.51 [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 the heading directive of the same name for suppressing the check globally.) You can omit **[nocheck]** from completely empty bars and bars in which nothing appears other than a whole bar rest indication.

### 10.2.52 [Noclef]

This specifies an invisible clef (☞ 10.1). 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.

### 10.2.53 [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 systems. 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. This directive need appear in only 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. Section 6.3 explains how PMW identifies uncounted bars if it needs to refer to them, for example, in an error message.

### 10.2.54 [Noteheads]

Three alternative notehead shapes are supported: diamond-shaped for string harmonics, cross-shaped, and invisible (that is, no noteheads at all). 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, and it is also available as an exotic fifth form of notehead. It is also possible to print any of the noteheads without stems. 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'.

```
a b [noteheads cross] c d | [noteheads normal] e f
```

This example prints the second two notes in the first bar with cross-shaped noteheads. For printing stemless notes, the directive **[noteheads only]** requests that all stems and beams be suppressed until another occurrence of **[noteheads]** with a argument other than 'only'. 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 **[noteheads direct]** is selected (to print notes as 𝅜), ledger lines are printed as normal, but no stems or beams are printed. When no noteheads are being printed (**[noteheads none]**), breves and semibreves are completely invisible, but stems and beams are still drawn for other notes, though no ledger lines are printed. See **[notes]** for a way of suppressing noteheads *and* stems, but still drawing beams. With cross-shaped noteheads there is no difference between the appearance of a crotchet and a minim. Breves are distinguished from semibreves only when normal noteheads are being printed.

### 10.2.55 [Notes]

The directive **[notes off]** suppresses the printing of notes and their stems (and ledger lines). 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 (☞ 9.6.31). The effect of **[notes off]** is reversed by **[notes on]**. See **[noteheads]** for a way of suppressing noteheads while leaving both stems and beams.

Notes that are suppressed with **[notes off]** are by default omitted from MIDI output as well as from PostScript output. This can be changed by including the heading directive **midifornotesoff**.

### 10.2.56 [Notespacing]

The **[notespacing]** directive (abbreviation **[ns]**) specifies a temporary change in the horizontal distances between notes. Internally, note spacings are held to an accuracy of 0.001 of a point. If the directive is given with no arguments, it resets to the values that were current at the start of the movement. The most commonly used form of **[notespacing]** 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 |

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]
```

This example specifies that minims are to be printed three points further apart and 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 one bar on one stave will have been processed with the altered spacing. It is usually better to use this form:

```
[notespacing *0.8] a-b- g d | [ns]
```

In this case, the subsequent staves are also processed with the reduced spacing. If a change of note spacing is always required, whatever combination of staves is selected for printing, it can be given on stave 0.

### 10.2.57 [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 may be easier to enter if the letters C–B represent the octave starting at middle C rather than the one below it. The **[octave]** directive, which must be followed by a number, requests transposition by the number of octaves given. The octave can also be set at the same time as the clef (☞ 10.1). 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 that is in force.

### 10.2.58 [Olevel] and [olhere]

These directives control the position of the overlay level in exactly the same way as **[ulevel]** and **[ulhere]** for the underlay level (☞ 10.2.114).

### 10.2.59 [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 numbers are set by the **textsizes** heading directive. If this directive is not used, the size set by the **overlaysize** heading directive (whose parameter is different from any of the sizes set by **textsizes**) is used. **[Oltextsize]** is normally needed only if you want different sizes of overlay text on different staves.

### 10.2.60 [Omitempty]

When a stave is about to be suspended (☞ 6.16, 10.2.96), it is sometimes desirable not to print stave lines after the final bar that 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, *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 by omitting them at the end of a stave's data. Note that a bar that 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]
```

The clef specification (possibly made invisible by means of **[assume]**) can then follow. It is not necessary for the suspend mechanism to be used with this feature, though if is not, vertical white space is 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.

```
[omitempty] ggg |? [skip 3] |? aaa |
```

This example specifies that no bar line it to be printed at the end of the bar before the skip, nor at the start of the final bar. Without the question marks, there would be bar lines in both these places. Note that because of the way **[skip]** works, this example contains 4 empty bars, not 3. The gap in this case is quite small, because, in the absence of other staves, PMW has packed them up into a single (invisible) 'rest' bar.

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 cannot be used in the empty bars, because that causes PMW to treat them as not empty.

### 10.2.61 [Overdraw]

When a drawing is associated with a note or bar line by means of the **[draw]** directive, 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.

### 10.2.62 [Overlayfont]

The default typeface for overlay text can be set for an individual stave by means of the **[overlayfont]** directive, which takes as its argument one of the standard font names.

```
[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.

### 10.2.63 [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]** 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. You can specify an absolute page number, or an increment of the page number preceded by a plus sign.

```
[page 45]
[page +1]
```

### 10.2.64 [Percussion]

*The* **[percussion]** *directive is deprecated, having been superseded by the* **[stavelines]** *directive, which should be used instead in all new input files.*

The **[percussion]** directive, which has no arguments, 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 (that is, under the line that 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 noteheads are printed. Although no clef is printed, the vertical positioning of notes is relative to the current clef.

```
[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.

### 10.2.65 [Playtranspose]

This directive is a synonym for **[miditranspose]**. It dates from the early days of PMW running on Acorn hardware, when playing was possible without using MIDI.

### 10.2.66 [Playvolume]

This directive is a synonym for **midivolume**. It dates from the early days of PMW running on Acorn hardware, when playing was possible without using MIDI.

### 10.2.67 [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 that 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, B is 59, and so on.

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), and 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 argument. 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.

### 10.2.68 [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. 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.

```
[stems up] gabc' [reset] [stems down] efga |
```

This example 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]**.

Because PMW processes bars from left to right, **[reset]** must not appear between two notes that 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 that follows, because 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 mark. Also, **[reset]** may not occur in the middle of an irregular note group.

### 10.2.69 [Resume]

This directive forces a resumption of a suspended stave – see **[suspend]** for details.

### 10.2.70 [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 (☞ 9.6.16). The **[rlevel]** directive 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 argument 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.

```
[rlevel -12]
```

This example 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.

### 10.2.71 [Rmove]

This directive operates exactly as **[move]**, except that horizontal movements are scaled to the relative stave size.

### 10.2.72 [Rsmove]

This directive operates exactly as **[smove]**, except that horizontal movements are scaled to the relative stave size.

### 10.2.73 [Rspace]

This directive operates exactly as **[space]**, except that horizontal dimensions are scaled to the relative stave size.

### 10.2.74 [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), whereas **[sgnext]** makes the change for all systems that follow the current one. In each case a single number is required as an argument. 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).

### 10.2.75 [Skip]

When setting vocal or 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 approach 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.

```
[stave 2] gg | [skip 50] aa | [endstave]
```

This example 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, no clef directive should be given, because otherwise the clef directive is taken as part of the resumed bar after the skip. See also the **[assume]** and **[omitempty]** directives.

### 10.2.76 [Slur]

Slurs between adjacent single notes can be input by inserting an underline character after the first note (☞ 9.6.29). When a slur covers chords, or spans several single 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 (☞ 10.2.38).

### 10.2.77 Normal slurs

The **[slur]** and **[endslur]** directives enclose the notes and/or chords that are to be slurred.

```
a [slur] b-a-g-f- [endslur] g
```

This example 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 manual adjustment. Various options are provided for the **[slur]** directive for this purpose. 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 |

Here are some examples of the **[slur]** directive:

```
[slur]
[slur/b]
[slur/u4]
[slur/lu2/co4]
[slur/rr6]
[slur/a/u4/ld2]
[slur/a/lu2/ru4]
```

Repeated movement qualifiers are accumulated. The options `/u` and `/d` are shorthand for specifying an identical vertical adjustment of both ends of the slur. Specifying `/ci` causes the slur to become flatter, and 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 (for example, `/a8`) initializes the vertical positions of both end points, as does the use of the `/ao` or `/bu` options. This results in a horizontal slur by default. The `/h` option is not relevant in these cases, and is ignored if given. However, the options for moving the ends can be applied.

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 marks (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 that actually have associated text or dashes or extender lines. If not, all the notes on the stave are taken into account.

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 provokes an error, because it is an attempt to draw a slur of zero horizontal extent.

```
[slur] a [endslur]
```

This example is incorrect. However, if one end of the slur is moved, all is well.

```
[slur/rr15] a [endslur]
```

This example is acceptable. The slur starts at the note, and extends for 15 points to the right. Slurs may be nested to any depth.

```
a b [slur] c d | [slur/b] e f g [es] a | f e [es] d c |
```

This prints 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 section 5.5.

### 10.2.78 Additional control of slur shapes

Slurs are drawn using Bézier curves, which are described in many books on computer graphics. A Bézier curve is defined by two end points and two control points. 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.

For slurs, 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 independently moved.

| | |
|---|---|
| `/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 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.

### 10.2.79 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' (dashed) slur |
| `/ip` | draw an 'intermittent points' (dotted) slur |
| `/e` | draw an editorial slur |

These qualifiers can be freely mixed with the other slur qualifiers. However, 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.

### 10.2.80 Wiggly slurs

The option `/w` causes the curvature of the slur to change sides in the middle. For example, 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.

### 10.2.81 Split slurs

Slurs are correctly continued if they span a boundary between two systems. 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.

```
[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, even those for section 1. 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.
- The `/u` and `/d` options 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 is 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 that 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.

### 10.2.82 Overlapping nested slurs

Usually slurs are properly nested, that is, if a second slur starts within a slur, the inner slur ends before the outer slur. The slur notation in PMW is naturally nested, and automatically ensures that this

convention is followed. 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 **[endslur]** is obeyed, it terminates the slur whose data is on the top of the stack (and that data is removed). By default, therefore, **[endslur]** always terminates the most recently started slur.

Very occasionally, 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. To make this possible, the **[xslur]** ('crossing slur') directive causes an innermost nested slur cross over the one immediately outside it.

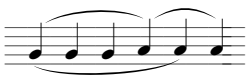
```
[slur] a [xslur] b [es] c [es]
```

This example draws one slur covering the first two notes, and the next slur covering the second and third notes. 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, and in the example above, the first **[es]** is thereby 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.

### 10.2.83 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. 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, that slur 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:

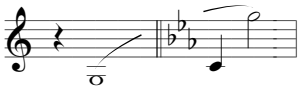
```
[slur/=A] [slur/b/=Z] ggg [slur/=B] a [es/=A] a [es/=Z] a [es] |
```



### 10.2.84 [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/co3/lu2] 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/"   )"]
```

In this example, the `/h0` and `/h1` options specify the start and end of the slur, respectively, and `/w0` specifies a gap of zero width. 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.

### 10.2.85 [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.

### 10.2.86 [Soprabass]

This specifies a bass clef with a little ‘8’ printed above it (☞ 10.1).

### 10.2.87 [Soprano]

This specifies a C clef with its centre on the bottom stave line (☞ 10.1).

### 10.2.88 [Space]

The **[space]** directive, which has a single number as an argument, causes space to be inserted before the next note or rest in the bar, or before the bar line if there are no more notes or rests. The remainder of the bar, including appropriate items on other staves, is adjusted accordingly. The number 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]**. When there are two or more occurrences of **[space]** at the same position in a bar, PMW takes the largest if previous ones specify a positive amount of space, and the smallest if they specify a negative amount. This normally gives the right effect if extra space is accidentally specified in two different staves.

**Note**: unlike **[move]**, **[space]** always affects the position of the next note, rest, or bar line, even if some other item intervenes. Other items are always printed in relation to the note, rest, or bar line that follows them. Therefore, adjusting the position of a note, rest, or bar line with **[space]** affects these items too. The following two examples have exactly the same effect:

```
[comma] [space 6] A
[space 6] [comma] A
```

This is 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.

```
[space 6][move -6][comma] A
```

In this example, **[space]** moves both the note and its attached comma (and everything that follows) to the right; **[move]** then moves the comma back to where it would have been without the inserted space. **[Space]** is obeyed when PMW is figuring out where to position the notes in the bar, whereas **[move]** is obeyed when the bar is output. See also **[ensure]**, **[move]**, **[smove]**, **[rspace]**, **[rmove]**, and **[rsmove]**.

### 10.2.89 [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 appears), whereas **[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. For example, a bar with very low notes might require notating thus:

```
[treble 1] [sshere 60] f` a` c e |
```

This example has the effect of setting the stave spacing to 60 points, for the current stave in the current system only. If the number is preceded by a plus or minus sign, it is interpreted as a change to the existing spacing.

```
[sshere +10]
```

This example adds 10 points to the stave spacing for the current stave in the current system. If, in a single stave, 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]**.

The argument for these directives can also take the form of two numbers separated by a slash, in which case the first is a stave number and the second is a spacing (which may be preceded by a plus or minus sign). More than one pair may be present. This makes it possible to encode all stave spacing changes in the same stave.

```
[ssnext 2/+8 3/-10 4/44]
```

If zero is given as a stave number, the spacing setting is applied to all the staves.

### 10.2.90 [Stave]

The first thing in each stave's data must be the **[stave]** directive. In its most basic form, the name is followed by just a stave number. Further arguments 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.

### 10.2.91 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 arguments. By default, the first one is used in the first system of the movement, and the second one for all other occurrences of this stave. These strings are normally used for the name of the instrument or voice for which the stave is intended.

```
[stave 1 "Soprano" "S"]
```

This example prints 'Soprano' at the start of the first system, and 'S' on all the others. If there is only one string, only the first system has text printed at the start of this stave. The third and subsequent strings in **[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. If a vertical bar appears in one of the strings, it specifies the start of a new line of text.

```
[stave 5 "Trumpet|in G"]
```

This example prints two lines at the start of the stave 5 in the first system. 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), 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. It is also possible to request that text be vertically positioned halfway between two successive staves. This is specified by appending `/m` (for 'middle') to the text on the upper of the two staves.

```
[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. You can specify a size for the 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.

```
[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. 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.

```
[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 for which they appear.

### 10.2.92 Drawings at stave starts

It is possible to cause a drawing function (see chapter 7) to be obeyed at the start of a stave. 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, so a string consisting of blanks can be used to ensure that an appropriate amount of space is left. The **contrib** directory in the PMW distribution contains an example where a drawing function associated with a stave is used to print a special kind of 'clef' for guitar tablature. The full syntax of **[stave]** is as follows:

`[stave` *<n>* *<string>* `draw` *<arguments>* *<drawing name>* `...]`

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.

```
[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 listed in order, following the corresponding strings, if present.

```
[stave 23  "Trumpet" draw 2.5 thing2  "Tr." draw "arg" thing3]
```

There is an ambiguity if an item that 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. There 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]
```

In this example the **stave** directive is terminated by the closing square bracket, so the **draw** directive is taken as part of the stave data and is associated with the following note in the usual way.

### 10.2.93 [Stavelines]

This directive specifies the number of lines to be drawn for the current stave. Its argument is a number in the range 0–6. A stave with no lines is an invisible stave. 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. Four-line and six-line staves are five-line staves with the top line missing or 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 in the PMW distribution contains an example of guitar tablature.

**[Stavelines]** applies to the entire stave, independently of where it appears. It has no implications for the printing of key signatures or clefs. For staves with fewer than five lines, ledger lines are not printed for notes that are off the stave. On one-line staves, whole bar rests are printed under the single

line, 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.

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 one of these examples:

```
[stavelines 1 noclef key C]
[stavelines 1 hclef  key C]
```

### 10.2.94 [Stemlength]

The **[stemlength]** directive is used to set a default value for the stem length adjustment on a stave.

```
[stemlength -2]
```

This example specifies that subsequent notes should have stems that are 2 points shorter than normal. 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. PMW can also be instructed to automatically shorten the stems of notes whose stems point the 'wrong' way. See the **shortenstems** heading directive for details.

### 10.2.95 [Stems]

Normally PMW chooses for itself in which direction to draw note stems. Details of the rules it uses are given in section 9.8; some variation is possible by means of the **stemswap** heading directive (☞ 8.1.120). You can also force note stems to point upwards or downwards, wherever the noteheads are on the stave. For individual notes there are options to do this; the **[stems]** directive sets a default for any notes that are not explicitly marked. It must be followed by one of the words 'up' (or 'above'), 'down' (or 'below'), or 'auto' – the last causing a reversion to the default state.

### 10.2.96 [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 **[suspend]** directive 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 on this stave in that system. The suspension ends automatically when a system is encountered in which there are notes or text to be printed on this stave.

```
[suspend] [108] R! |
```

This example specifies 108 bars rest, which can be suspended where possible. If the **[suspend]** directive appears in the first rest bar, as in this example, at least one rest bar is 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! |
```

Suspension can be ended early by the **[resume]** directive. If at least one rest bar is required when the stave is resumed, an explicit **[resume]** must appear in the last rest bar, because by default the stave may resume with a non-rest bar at the beginning of a system.

```
[suspend] [107] R! | [resume] R! |
```

When a single part is being printed, **[suspend]** has no effect, because a sequence of rest bars is automatically packed up into a single bar with a multiple rest sign. Because **[suspend]** stave directive 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.

### 10.2.97 [Tenor]

This specifies a C clef with its centre on the fourth stave line (☞ 10.1).

### 10.2.98 [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 argument.

| | |
|---|---|
| `[text above]` | print above the stave |
| `[text above` <*n*>`]` | ditto, at a given level |
| `[text below]` | print below the stave |
| `[text below` <*n*>`]` | ditto, at a given level |
| `[text underlay]` | default is underlay |
| `[text overlay]` | default is overlay |
| `[text fb]` | default is figured bass |

To override a default with an absolute position (for example **[text above 15]**), the text options `/a` or `/b` without a following number can be used (as well as `/ul`, `/fb`, or `/m`). Similarly, the appearance of **[text above]** or **[text below]** 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 **textsizes** 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, an italic dynamic mark to be printed above the stave is coded like this:

```
"\it\ff"/a
```

The default text type can be changed many times within one stave.

### 10.2.99 [Textfont]

The default typeface for non-underlay, non-overlay text can be set for an individual stave by means of the **[textfont]** directive, which takes as its argument one of the standard font names.

```
[textfont extra 3]
```

This example supposes that the third extra font has been defined for some special use in the stave's text. The default font for non-underlay, non-overlay text is italic. In any given text string it is always possible to change font by using the appropriate escape sequence.

### 10.2.100 [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 numbers are set by the **textsizes** 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.

### 10.2.101 [Tick]

The **[tick]** directive causes PMW to insert a tick pause mark above the current stave.

### 10.2.102 [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 **[ties]** directive is available for forcing the tie direction for all subsequent ties that are not explicitly marked. The argument 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.

### 10.2.103 [Time]

The time signature for a stave can be changed at the start of a bar by the **[time]** directive. 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.

```
[time 6/8 nowarn]
```

There is also a heading directive, **notimewarn**, for suppressing all cautionary time signatures. PMW does not work sensibly by default if different time signatures are used on different staves, unless they represent the same length 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 (☞ 10.2.104).

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] (:
(: [time 4/4]
```

The first example causes the time signature to be printed first, followed by the repeat mark, whereas the second example 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, the repeat sign is printed first on all staves.

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.

### 10.2.104 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, 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 `->`.

```
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. The notes are stretched or compressed (in position when printing and in time when generating a MIDI file) to make the bar lengths match.

### 10.2.105 [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.

### 10.2.106 [Transpose]

The **[transpose]** directive specifies that the current stave is to be transposed by a number of semitones. A positive number transposes upwards; a negative number transposes downwards. This directive 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 also 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 signa-

ture 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.

```
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 section 6.10, and details of the transposition of chord names are given in section 6.14.6.

### 10.2.107 [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 (☞ 6.10).

### 10.2.108 [Treble]

This specifies a treble clef (☞ 10.1).

### 10.2.109 [Trebledescant]

This specifies a treble clef with a little '8' printed above it (☞ 10.1).

### 10.2.110 [Trebletenor]

This specifies a treble clef with a little '8' printed below it (☞ 10.1).

### 10.2.111 [TrebletenorB]

This specifies a clef that is exactly like the trebletenor clef, except that the little '8' is printed enclosed in parentheses.

### 10.2.112 [Tremolo]

It is possible to print tremolo marks 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, and `/j` followed by a number specifies the number of beams that are to be joined to the note stems. The default is to draw two beams, neither of which is joined to the stems.

```
g [tremolo] b
```

This example prints two crotchets, with two disconnected beams between them.

```
G [tremolo/x3/j3] B
```

This example 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 9.6.16.

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.

### 10.2.113 [Triplets]

This directive is used to control the printing of triplet and other irregular note group marks. 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, or ‘auto’. ‘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 ‘off’ is specified, 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.

```
[triplets above]
```

This example causes all the irregular note marks 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. The words ‘above’ and ‘below’ can be followed by a dimension, to set a fixed vertical position for all subsequent irregular note group marks. If the dimension is preceded by `+` or `-`, this does not set a fixed position, but provides a default vertical adjustment for subsequent irregular groups.

```
[triplets above +4]
```

This example causes subsequent marks 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, 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 marks may be printed above or below the notes, depending on their pitch, and the bracket is omitted 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`.

### 10.2.114 [Ulevel] and [ulhere]

For each stave that it prints, PMW computes a default 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 may be chosen if there are low notes on the stave. There are two different ways of changing this level. The **[ulhere]** directive specifies a temporary change for the current line, and the **[ulevel]** directive sets an absolute level to be used until further notice.

**[Ulhere]** takes a positive or negative number as an argument. This is interpreted as a number of points to add to the automatically computed level for the line in which the current bar appears.

```
[ulhere -2]
```

This example 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 argument is negative and specifies a lower level than the previous one, or if its argument 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.

```
[ulevel -15]
```

This example sets a level fifteen points the bottom of the stave. The **[ulevel]** 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 argument, the underlay level reverts to the value automatically selected by PMW, and any subsequent **[ulhere]** directives are honoured.

### 10.2.115 [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 numbers are set by the **textsizes** 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 **textsizes**) is used. **[Ultextsize]** is normally needed only if you want different sizes of underlay text on different staves.

### 10.2.116 [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]**.

### 10.2.117 [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 argument one of the standard font names.

```
[underlayfont extra 3]
```

This example supposes that the third extra font has been defined for use in underlay text. 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.

### 10.2.118 [Xline]

See **[line]** (☞ 10.2.38) and also section 10.2.82.

### 10.2.119 [Xslur]

See section 10.2.82.

# 11. Characters in text fonts

PostScript text fonts such as *Times-Roman* contain over 300 printing characters. From release 4.10, PMW gives access to all these characters by making use of Unicode encoding, which allows for character codes that are greater than 255. There are several ways in which characters other than the standard ASCII set can be represented in PMW text strings; these are described in section 6.14.3. This chapter lists all the characters in the PostScript standard text fonts, with their Unicode values (in hexadecimal, as is conventional), their PMW escape sequences when defined, and their PostScript character names.

The use of the escape sequence `\fi` for the ‘fi’ ligature is no longer necessary, because PMW now automatically uses the ligature for variable width fonts when it is available. The escape sequence is retained for backwards compatibility. PMW does not use the ‘fl’ ligature automatically.

Printing characters whose code values are less than 007F (127) are ASCII characters that correspond to the keys on the computer keyboard. However, in PMW strings, the literal characters grave accent and single quote (codes 0060 and 0027) are converted into Unicode characters 2018 and 2019 so that they print as opening and closing quotes, respectively. If you want to print a grave accent or an ASCII single quote character, you can use the appropriate escape sequences `\`` and `\'` or a numerical escape sequence such as `\x60\`.

---

| Unicode | Escape | PS name | Character |
|---|---|---|---|
| 0020 | | space | |
| 0021 | | exclam | ! |
| 0022 | | quotedbl | " |
| 0023 | | numbersign | # |
| 0024 | | dollar | $ |
| 0025 | | percent | % |
| 0026 | | ampersand | & |
| 0027 | \' | quotesingle | ' |
| 0028 | | parenleft | ( |
| 0029 | | parenright | ) |
| 002A | | asterisk | * |
| 002B | | plus | + |
| 002C | | comma | , |
| 002D | | hyphen | - |
| 002E | | period | . |
| 002F | | slash | / |
| 0030 | | zero | 0 |
| ↓ | | ↓ | ↓ |
| 0039 | | nine | 9 |
| 003A | | colon | : |
| 003B | | semicolon | ; |
| 003C | | less | < |
| 003D | | equal | = |
| 003E | | greater | > |
| 003F | | question | ? |
| 0040 | | at | @ |
| 0041 | | A | A |
| ↓ | | ↓ | ↓ |
| 005A | | Z | Z |
| 005B | | bracketleft | [ |

| | | | |
|---|---|---|---|
| 005C | \\ | backslash | \ |
| 005D | | bracketright | ] |
| 005E | | asciicircum | ^ |
| 005F | | underscore | _ |
| 0060 | \` | grave | ` |
| 0061 | | a | a |
| ↓ | | ↓ | ↓ |
| 007A | | z | z |
| 007B | | braceleft | { |
| 007C | | bar | \| |
| 007D | | braceright | } |
| 007E | | asciitilde | ~ |
| 00A1 | | exclamdown | ¡ |
| 00A2 | | cent | ¢ |
| 00A3 | | sterling | £ |
| 00A4 | | currency | ¤ |
| 00A5 | | yen | ¥ |
| 00A6 | | brokenbar | ¦ |
| 00A7 | | section | § |
| 00A8 | | dieresis | ¨ |
| 00A9 | \c) | copyright | © |
| 00AA | | ordfeminine | ª |
| 00AB | | guillemotleft | « |
| 00AC | | logicalnot | ¬ |
| 00AE | | registered | ® |
| 00AF | | macron | ¯ |
| 00B0 | | degree | ° |
| 00B1 | | plusminus | ± |
| 00B2 | | twosuperior | ² |
| 00B3 | | threesuperior | ³ |
| 00B4 | | acute | ´ |
| 00B5 | | mu | µ |
| 00B6 | | paragraph | ¶ |
| 00B7 | | bullet | • |
| 00B8 | | cedilla | ¸ |
| 00B9 | | onesuperior | ¹ |
| 00BA | | ordmasculine | º |
| 00BB | | guillemotright | » |
| 00BE | | onequarter | ¼ |
| 00BD | | onehalf | ½ |
| 00BE | | threequarters | ¾ |
| 00BF | \? | questiondown | ¿ |
| 00C0 | \A` | Agrave | À |
| 00C1 | \A' | Aacute | Á |
| 00C2 | \A^ | Acircumflex | Â |
| 00C3 | \A~ | Atilde | Ã |
| 00C4 | \A. | Adieresis | Ä |
| 00C5 | \Ao | Aring | Å |
| 00C6 | | AE | Æ |

| | | | |
|---|---|---|---|
| 00C7 | \C, | Ccedilla | Ç |
| 00C8 | \E` | Egrave | È |
| 00C9 | \E' | Eacute | É |
| 00CA | \E^ | Ecircumflex | Ê |
| 00CB | \E. | Edieresis | Ë |
| 00CC | \I` | Igrave | Ì |
| 00CD | \I' | Iacute | Í |
| 00CE | \I^ | Icircumflex | Î |
| 00CF | \I. | Idieresis | Ï |
| 00D0 | | Eth | Ð |
| 00D1 | \N~ | Ntilde | Ñ |
| 00D2 | \O` | Ograve | Ò |
| 00D3 | \O' | Oacute | Ó |
| 00D4 | \O^ | Ocircumflex | Ô |
| 00D5 | \O~ | Otilde | Õ |
| 00D6 | \O. | Odieresis | Ö |
| 00D7 | | multiply | × |
| 00D8 | \O/ | Oslash | Ø |
| 00D9 | \U` | Ugrave | Ù |
| 00DA | \U' | Uacute | Ú |
| 00DB | \U^ | Ucircumflex | Û |
| 00DC | \U. | Udieresis | Ü |
| 00DD | \Y' | Yacute | Ý |
| 00DE | | Thorn | Þ |
| 00DF | \ss | germandbls | ß |
| 00E0 | \a` | agrave | à |
| 00E1 | \a' | aacute | á |
| 00E2 | \a^ | acircumflex | â |
| 00E3 | \a~ | atilde | ã |
| 00E4 | \a. | adieresis | ä |
| 00E5 | \ao | aring | å |
| 00E6 | | ae | æ |
| 00E7 | \c, | ccedilla | ç |
| 00E8 | \e` | egrave | è |
| 00E9 | \e' | eacute | é |
| 00EA | \e^ | ecircumflex | ê |
| 00EB | \e. | edieresis | ë |
| 00EC | \i` | igrave | ì |
| 00ED | \i' | iacute | í |
| 00EE | \i^ | icircumflex | î |
| 00EF | \i. | idieresis | ï |
| 00F0 | | eth | ð |
| 00F1 | \n~ | ntilde | ñ |
| 00F2 | \o` | ograve | ò |
| 00F3 | \o' | oacute | ó |
| 00F4 | \o^ | ocircumflex | ô |
| 00F5 | \o~ | otilde | õ |
| 00F6 | \o. | odieresis | ö |
| 00F7 | | divide | ÷ |

| | | | |
|---|---|---|---|
| 00F8 | \o/ | oslash | ø |
| 00F9 | \u` | ugrave | ù |
| 00FA | \u' | uacute | ú |
| 00FB | \u^ | ucircumflex | û |
| 00FC | \u. | udieresis | ü |
| 00FD | \y' | yacute | ý |
| 00FE | | thorn | þ |
| 00FF | \y. | ydieresis | ÿ |
| 0100 | \A- | Amacron | Ā |
| 0101 | \a- | amacron | ā |
| 0102 | \Au | Abreve | Ă |
| 0103 | \au | abreve | ă |
| 0104 | | Aogonek | Ą |
| 0105 | | aogonek | ą |
| 0106 | \C' | Cacute | Ć |
| 0107 | \c' | cacute | ć |
| 010C | \Cv | Ccaron | Č |
| 010D | \cv | ccaron | č |
| 010E | \Dv | Dcaron | Ď |
| 010F | \dv | dcaron | ď |
| 0110 | \D- | Dcroat | Đ |
| 0111 | \D- | dcroat | đ |
| 0112 | \E- | Emacron | Ē |
| 0113 | \e- | emacron | ē |
| 0116 | | Edotaccent | Ė |
| 0117 | | edotaccent | ė |
| 0118 | | Eognonek | Ę |
| 0119 | | eogonek | ę |
| 011A | \Ev | Ecaron | Ě |
| 011B | \ev | ecaron | ě |
| 011E | \Gu | Gbreve | Ğ |
| 011F | \gu | gbreve | ğ |
| 0122 | | Gcommaaccent | Ģ |
| 0123 | | gcommaaccent | ģ |
| 012A | \I- | Imacron | Ī |
| 012B | \i- | imacron | ī |
| 012E | | Iogonek | Į |
| 012F | | iogonek | į |
| 0130 | | Idotaccent | İ |
| 0131 | | dotlessi | ı |
| 0136 | | Kcommaaccent | Ķ |
| 0137 | | kcommaaccent | ķ |
| 0139 | \L' | Lacute | Ĺ |
| 013A | \l' | lacute | ĺ |
| 013B | | Lcommaaccent | Ļ |
| 013C | | lcommaaccent | ļ |
| 013D | \Lv | Lcaron | Ľ |
| 013E | \lv | lcaron | ľ |
| 0141 | \l/ | Lslash | Ł |

| | | | |
|---|---|---|---|
| 0142 | \l/ | lslash | ł |
| 0143 | \N' | Nacute | Ń |
| 0144 | \n' | nacute | ń |
| 0145 | | Ncommaaccent | Ņ |
| 0146 | | ncommaaccent | ņ |
| 0147 | \Nv | Ncaron | Ň |
| 0148 | \nv | ncaron | ň |
| 014C | \O- | Omacron | Ō |
| 014D | \o- | omacron | ō |
| 0150 | | Ohungrumlaut | Ő |
| 0151 | | ohungrumlaut | ő |
| 0152 | | OE | Œ |
| 0153 | | oe | œ |
| 0154 | \R' | racute | Ŕ |
| 0156 | | Rcommaaccent | Ŗ |
| 0157 | | rcommaaccent | ŗ |
| 0158 | \Rv | Rcaron | Ř |
| 0159 | \rv | rcaron | ř |
| 015A | \S' | Sacute | Ś |
| 015B | \s' | sacute | ś |
| 015E | \S, | Scedilla | Ş |
| 015F | \s, | scedilla | ş |
| 0160 | \Sv | Scaron | Š |
| 0161 | \sv | scaron | š |
| 0164 | \Tv | Tcaron | Ť |
| 0165 | \tv | tcaron | ť |
| 016A | \U- | Umacron | Ū |
| 016B | \u- | umacron | ū |
| 016E | \Uo | Uring | Ů |
| 016F | \uo | uring | ů |
| 0170 | | Uhungrumlaut | Ű |
| 0171 | | uhungrumlaut | ű |
| 0172 | | Uogonek | Ų |
| 0173 | | uogonek | ų |
| 0178 | \Y. | Ydieresis | Ÿ |
| 0179 | \Z' | Zacute | Ź |
| 017A | \z' | zacute | ź |
| 017B | | Zdotaccent | Ż |
| 017C | | zdotaccent | ż |
| 017D | \Zv | Zcaron | Ž |
| 017E | \zv | zcaron | ž |
| 0192 | | florin | *f* |
| 0218 | | Scommaaccent | Ș |
| 0219 | | scommaaccent | ș |
| 021A | | Tcommaaccent | Ţ |
| 021B | | tcommaaccent | ţ |
| 0302 | | circumflex | ˆ |
| 0303 | | tilde | ˜ |
| 0306 | | breve | ˘ |

| | | | |
|---|---|---|---|
| 0307 | | dotaccent | ˙ |
| 030A | | ring | ˚ |
| 030B | | hungrumlaut | ˝ |
| 030C | | caron | ˇ |
| 0326 | | commaaccent | |
| 0328 | | ogonek | ˛ |
| 0394 | | Delta | Δ |
| 2013 | \-- | endash | – |
| 2014 | \--- | emdash | — |
| 2018 | | quoteleft | ‘ |
| 2019 | | quoteright | ’ |
| 201A | | quotesinglbase | ‚ |
| 201C | | quotedblleft | “ |
| 201D | | quotedblright | ” |
| 201E | | quotedblbase | „ |
| 2020 | | dagger | † |
| 2021 | | daggerdbl | ‡ |
| 2026 | | ellipsis | … |
| 2027 | | periodcentred | · |
| 2031 | | perthousand | ‰ |
| 2039 | | guilsinglleft | ‹ |
| 203A | | guilsinglright | › |
| 2044 | | fraction | ⁄ |
| 20AC | | Euro | € |
| 2122 | | trademark | ™ |
| 2202 | | partialdiff | ∂ |
| 2211 | | summation | ∑ |
| 2212 | | minus | − |
| 221A | | radical | √ |
| 2260 | | notequal | ≠ |
| 2264 | | lessequal | ≤ |
| 2265 | | greaterequal | ≥ |
| 25CA | | lozenge | ◊ |
| FB01 | \fi | fi | fi |
| FB02 | \fl | fl | fl |

# 12. The PMW music font

This chapter contains a list of all the available characters in the music 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"
```

Character 33 in the music font (which corresponds to an exclamation mark in ASCII) is the treble clef. 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 that are useful when these characters are printed as part of a text string.

Here is a list of the characters in the font, giving both their numbers (in decimal) 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.

---

| ASCII | Code | Width | Char | Comment |
|---|---|---|---|---|
| | 32 | 0.75 | | space |
| ! | 33 | 1.5 | 𝄞 | treble clef |
| " | 34 | 1.5 | 𝄢 | bass clef |
| # | 35 | 1.5 | 𝄡 | alto clef |
| $ | 36 | 1.0 | 𝄇 | piano end pedal sign |
| % | 37 | 0.6 | ♯ | sharp |
| & | 38 | 0.6 | 𝄪 | double sharp |
| ' | 39 | 0.5 | ♭ | flat |
| ( | 40 | 0.45 | ♮ | natural |
| ) | 41 | 0.0 | 𝄐 | fermata (over) |
| * | 42 | 0.66 | 𝄻 | breve rest |
| + | 43 | 0.66 | 𝄼 | semibreve rest |
| , | 44 | 0.66 | 𝄽 | minim rest |
| - | 45 | 0.66 | 𝄾 | crotchet rest |
| . | 46 | 0.59 | 𝄿 | quaver rest |
| / | 47 | 0.0 | 𝄑 | fermata (under) |
| 0 | 48 | 3.5 | 𝄃 | many bars rest |
| 1 | 49 | 1.34 | 𝄆 | breve |
| 2 | 50 | 0.84 | 𝅝 | semibreve |
| 3 | 51 | 0.84 | 𝅗𝅥 | up minim |
| 4 | 52 | 0.84 | 𝅗𝅥 | down minim |
| 5 | 53 | 0.84 | ♩ | up crotchet |
| 6 | 54 | 0.84 | ♩ | down crotchet |
| 7 | 55 | 1.2 | ♪ | up quaver |
| 8 | 56 | 0.84 | ♪ | down quaver |
| 9 | 57 | 1.2 | 𝅘𝅥𝅯 | up semiquaver |
| : | 58 | 0.84 | 𝅘𝅥𝅯 | down semiquaver |

| | | | | |
|---|---|---|---|---|
| ; | 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 | | for repeat marks |
| 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 | | pause comma |
| O | 79 | 0.0 | | mordent |
| P | 80 | 0.0 | | double mordent |
| Q | 81 | 0.0 | | inverted mordent |
| R | 82 | 0.0 | | double inverted mordent |
| S | 83 | 0.0 | | turn |
| T | 84 | 0.0 | | horizontal bar accent |
| U | 85 | 0.0 | | accent |
| V | 86 | 1.0 | | caesura |
| W | 87 | 0.0 | | accent |
| X | 88 | 0.0 | | accent |
| Y | 89 | 0.0 | | accent |
| Z | 90 | 0.0 | | accent |
| [ | 91 | 0.6 | | dashed bar line |
| \ | 92 | 1.0 | | single-line caesura |
| ] | 93 | 0.0 | | for use with clefs |
| ^ | 94 | 1.0 | | ‘common’ time |
| _ | 95 | 1.0 | | ‘cut’ time |
| ` | 96 | 0.4 | | suitable for following *tr* |
| a | 97 | 0.0 | | thumb (above) |
| b | 98 | 0.0 | | thumb (below) |
| c | 99 | 1.5 | | dal segno |
| d | 100 | 1.5 | | dal segno |
| e | 101 | 0.0 | | down bow |
| f | 102 | 0.0 | | inverted down bow |
| g | 103 | 0.0 | | up bow |
| h | 104 | 0.0 | | inverted up bow |

| | | | | |
|---|---|---|---|---|
| i | 105 | 0.0 |  | inverted turn |
| j | 106 | 0.55 |  | for figured bass |
| k | 107 | 0.76 |  | for figured bass |
| 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 |  | for figured bass |
| t | 116 | 0.55 |  | dot for guitar grid |
| u | 117 | 0.55 |  | circle for guitar grid |
| v | 118 | 0.0 | | prints nothing; moves down by 0.1 |
| w | 119 | 0.0 | | prints nothing; moves down by 0.4 |
| x | 120 | 0.0 | | prints nothing; moves up by 0.4 |
| y | 121 | -0.1 | | prints nothing; moves left by 0.1 |
| z | 122 | 0.1 | | prints nothing; moves right by 0.1 |
| { | 123 | -0.33 | | prints nothing; moves left by 0.33 |
| \| | 124 | 0.0 | | prints nothing; moves down by 0.2 |
| } | 125 | 0.55 | | prints nothing; moves right by 0.55 |
| ~ | 126 | 0.0 | | prints nothing; moves up 0.2 |
| | 127 | - | | unassigned |
| | 128 | 0.6 |  | tick |
| | 129 | 0.0 |  | accaciatura bar |
| | 130 | 0.0 |  | accaciatura bar |
| | 131 | 0.0 | | grid for guitar chords |
| | 132 | 0.6 |  | short bar line |
| | 133 | 0.0 |  | breath |
| | 134 | 0.0 |  | ring above |
| | 135 | 0.0 |  | cross |
| | 136 | 0.8 |  | trill |
| | 137 | 0.6 |  | short vertical caesura |
| | 138 | 0.6 |  | long vertical caesura |
| | 139 | 0.35 | [ | brackets for accidentals |
| | 140 | 0.35 | ] | |
| | 141 | 0.35 | ( | |
| | 142 | 0.35 | ) | |
| | 143 | 0.5 |  | for bar repetition |
| | 144 | 0.0 |  | for bar repetition |
| | 145 | 0.0 |  | for arpeggios – moves upwards by 0.4 |
| | 146 | 0.0 |  | tremolo bar – moves upwards by 0.4 |
| | 147 | 1.0 |  | old time signature |
| | 148 | 1.0 |  | old time signature |
| | 149 | 0.0 |  | slur |
| | 150 | 0.0 |  | slur |

| | | | |
|---|---|---|---|
| 151 | 0.0 | ↗ | for splitting/joining staves |
| 152 | 0.0 | ↘ | for splitting/joining staves |
| 153 | 1.0 | 𝅝 | inverted ‘common’ time |
| 154 | 1.0 | 𝅜 | inverted ‘cut’ time |
| 155 | 1.58 | 𝅜 | unison breve |
| 156 | 0.0 | ˈ | ‘start of bar’ accent |
| 157 | 0.35 | ( | for bracketing *8* |
| 158 | 0.35 | ) | for bracketing *8* |
| 159 | 0.33 | ⎡ | for *8va* lines etc. |
| 160 | 0.33 | ⎢ | |
| 161 | 0.33 | ⎢ | |
| 162 | 0.33 | ⎣ | |
| 163 | 1.4 | 𝒫𝑒𝒹. | piano pedal |
| 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 angle |
| 168 | 0.333 | ⎣ | alternate bracket angle |
| 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 | for figured bass |
| 180 | 0.75 | △ | major chord sign (jazz notation) |
| 181 | 0.675 | ○ | diminished chord sign |
| 182 | 0.675 | ∅ | ‘half diminished’ chord sign |
| 183 | 0.55 | × | cross for guitar grid |
| 184 | 0.0 | — | thicker ledger line |
| 185 | -0.42 | | prints nothing; moves left 0.42, up 0.4 |
| 186 | -0.76 | | prints nothing; moves left 0.76, down 0.4 |
| 187 | 0.0 | | prints nothing; moves up 1.2 |
| 188 | 0.0 | | prints nothing; moves down 1.2 |
| 189 | 0.424 | 𝄲 | half sharp, Egyptian style |
| 190 | 0.5 | 𝄲 | half sharp, Turkish style |
| 191 | 0.6 | 𝄳 | half flat, Egyptian style |
| 192 | 0.6 | 𝄳 | half flat, Turkish style |
| 193 | 0.6 | ❛ | pause comma, inverted for R-to-L music |
| 194 | 0.0 | ' | staccatissimo |
| 195 | 0.0 | ' | staccatissimo, inverted |
| 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 |

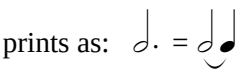
---

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 that 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, 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.

There is a discussion on the use of the special characters for printing guitar chord grids in section 6.14.10. 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 8.1.130). 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*.

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.

```
"\**m.\ = \mf\\149\\51\\53\"
```

This example prints as: 

# 13. The PMW-Alpha font

Richard Hallas contributed an auxiliary font for use with PMW and other programs. It is called PMW-Alpha, because it is designed for printing music 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 marks.
- 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 music 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.

## 13.1 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* mark that prints *f*. The second macro is for printing the fraction ⅗. 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\"
```

## 13.2 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.

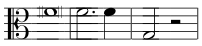
The breves are not supposed to be used with dots, and have a width of 640. All the other notes have a width of 380 and can be followed by either two dots, a dot and a ‘dot space’ (< character) or a ‘double-dot space’ (comma character) to make up a 640-width ‘unit’.

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, 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.’’

## 13.3 Characters in the font

Here is a list 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.

---

| ASCII | Code | Width | Char | Comment |
|---|---|---|---|---|
| | 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 |  | |
| 4 | 52 | 0.64 |  | |
| 5 | 53 | 0.64 |  | |
| 6 | 54 | 0.64 |  | |
| 7 | 55 | 0.64 |  | |
| 8 | 56 | 0.64 |  | |
| 9 | 57 | 0.64 |  | |
| : | 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 |  | beams for downstemmed notes |
| J | 74 | 0.00 |  | |
| 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 |  | beams for upstemmed notes |
| j | 106 | 0.00 |  | |
| k | 107 | 0.00 |  | |

| | | | | |
|---|---|---|---|---|
| m | 109 | 0.88 | 𝒎 | |
| o | 111 | 0.76 | 𝒕𝒓 | |
| p | 112 | 0.69 | 𝒑 | |
| 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 | ⅖ | |
| | 185 | 0.75 | ⅓ | |
| | 186 | 0.75 | ⅑ | |
| | 187 | 0.75 | ⅝ | |
| | 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 |

# 14. Syntax summary

## 14.1 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*"

## 14.2 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> <m>
 Gracestyle <n>
 Hairpinlinewidth <n>
 Hairpinwidth <n>
 Halfflatstyle <n>
 Halfsharpstyle <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> ...
 Ledgerstyle <n>
 Leftmargin <n>
 Linelength <n>
 Longrestfont <fontsize> <font>
*Magnification <n>
 Maxbeamslope <n>
*Maxvertjustify <n>
 Midichannel <n> "<name or number>" <staves>
 Midifornotesoff
 Midistart <n> <n> <n> ...
 Miditempo <n> <n>/<m> ...
 Miditranspose <n>/<m> ...
 Midivolume <n> <n>/<m> ...
 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>
 Printkey <key> <clef> "<string>"
 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>
*Righttoleft
 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> ...
*Stretchrule <n>
†Suspend <n> ...
 Systemgap <n>
*Textfont <fontword> "<font name>"
 Textsizes <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>
```

## 14.3 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 | `# ## $ $$ %` |
| half sharp | `#-` |
| half flat | `$-` |
| invisible | `?` |
| over note | `o` |
| under note | `u` |
| transposed | `^# ^## ^$ ^$$ ^% ^- ^+` |
| 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 |
| `..` | staccatissimo |
| `-` | 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 |
| sm | use small notehead |
| 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 |
| 10 | staccatissimo |

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. All accents, and the fermata, mordant, trill, and turn ornaments can be shown in parentheses or square brackets, by following the code with one of:

| | |
|---|---|
| /( | precede with an opening parenthesis |
| /[ | precede with an opening square bracket |
| /) | follow with a closing parenthesis |
| /] | follow with a closing square bracket |
| /b | enclose in parentheses |
| /B | enclose in square brackets |

## 14.4 Special characters in stave data

These characters, along with text items, occur interspersed in the notes and rests:

| | |
|---|---|
| \| | bar line |
| \|\| | double bar line |
| \|\|\| | end-of-piece 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 |

## 14.5 Stave text item options

These options may occur after any text item, but for a rehearsal mark 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 (☞ 9.12.5).

## 14.6 Character string escapes

The escape sequences that specify accented and other special characters in text fonts are shown in the character list in chapter 11. The remaining escape sequences are summarized here.

| | |
|---|---|
| `\c]` | prints © from the PostScript Symbol font |
| `\p\` | page number |
| `\pe\` | page number, if even |
| `\po\` | page number, if odd |
| `\@` | 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 whose size is 9/10 that of the surrounding text. A double asterisk uses a full size font.

| | |
|---|---|
| \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 |

The character number can be given as a decimal number, or as a hexadecimal number preceded by x, for example *109\ or \x20ac\.

| | |
|---|---|
| \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.

## 14.7 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 |

## 14.8 Bracketed stave directives

These directives occur in square brackets interspersed in among the notes and rests:

| | |
|---|---|
| [\!\] | repeated accent movement |
| [\.\] | repeated staccato |
| [\..\] | repeated staccatissimo |
| [\-\] | repeated accent |

| | |
|---|---|
| [\>\] | repeated horizontal wedge accent |
| [\v\] | repeated small vertical wedge accent |
| [\V\] | repeated large vertical wedge accent |
| [\'\] | repeated ‘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 mark |
| [all] | end 1st/2nd time bars |
| [alto <*octave*>] | select alto clef |
| [assume <*setting*>] | assume key, time, or clef |
| [baritone <*octave*>] | select baritone clef |
| [barlinestyle <*n*>] | select bar line style for stave |
| [barnumber</*options*>] | 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] | bowing marks above |
| [bowing below] | bowing marks below |
| [breakbarline] | break one bar line on one stave |
| [cbaritone <*octave*>] | 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 <*octave*>] | select deep bass clef |
| [dots above] | augmentation dots above |
| [dots below] | augmentation dots below |
| [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]` | put hairpins above |
| `[hairpins below]` | put hairpins below |
| `[hairpins middle]` | centre hairpins between two staves |
| `[hairpinwidth` *<n>*`]` | set hairpinwidth for this stave |
| `[hclef` *<octave>*`]` | 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 |
| `[miditranspose` *<n>*`]` | change MIDI transposition |
| `[midivoice "`*name*`"]` | change MIDI voice |
| `[midivolume` *<n>*`]` | set relative MIDI volume |
| `[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` *<octave>*`]` | select invisible treble clef |
| `[nocount]` | don’t count this bar for numbering |
| `[noteheads` *<style>*`]` | select notehead 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)<br>`[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] and [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*> ...] | 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] |

## 14.9 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 options 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 Bézier 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’ |

## 14.10 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 |
| Caesura style | 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 signature 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

## C



## D

## E



## F

## G



## H



## I



## J



## K



## L

## M



## N

## O



## P

## R



## S

## T

## U



## V



## W



## X