Philip's Music Writer (PMW)

A Music Typesetting Program

Philip Hazel

Philip's Music Writer (PMW)

Author: Philip Hazel

Revision 5.33   09 December 2025

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 aim is the production of high quality sheet music. PMW is not a screen-based program; it is run from the command line. It reads an input file containing an encoded description of some 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, included files, and shorthands for repetitive input. The textual encoding is unique to PMW. If you want to get a flavour of PMW input before deciding whether or not to install PMW, take a look at chapter 5 below.

From release 5.10 PMW can also read and typeset some music encoded as MusicXML, which is a format used for interchanging encoded music between different processing programs. However, this is still very much an experimental feature that is not yet of production quality. Moreover, only a subset of MusicXML features are supported. It is very much 'work in progress' and therefore relegated to appendix A of this document.

Similarly, from release 5.33, PMW can optionally write a copy of the music in MusicXML format. This, too, is experimental and incomplete. It is documented in appendix B.

The main output of PMW is either a PostScript or a PDF (Portable Document Format) file. The former format can be 'encapsulated PostScript', which can be useful if the music is an illustration that will subsequently be included in another document. For a discussion of PostScript versus PDF, see chapter 13. In addition to its normal output, PMW can also write a MIDI file for playing through the computer's sound system. MIDI output is not at all sophisticated, and is intended for 'proof-hearing' purposes rather than for performance.

Early versions of PMW were called *Philip's Music Scribe*, and ran on Acorn's RISC OS operating system in the 1990s. Version 4.00 was the first version for Linux and other Unix-like environments. This edition of the manual describes PMW version 5.33. Version 5.00 was a major re-working of the code to properly integrate half accidentals and custom key signatures, to improve the way character strings are processed, to remove obsolete features, and to generally overhaul what is now quite old code. Chapter 14 has details of the changes; in the majority of cases existing input files should continue to work without change or with only minor adjustments.

PMW comes with a 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 for certain types of music (see chapter 17).

The PMW input encoding is intended 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. However, PMW input could also be be the output of some other computer program that captures (or generates) music.

This introduction ends with a short summary of the musical and other terminology used in this manual. The following two chapters describe how PMW should be installed and operated. Chapters 5 and 6 are an introduction to the PMW input encoding. They cover most of the more common requirements, with examples, in an introductory manner. Chapter 7 describes the 'standard macros' that are supplied with PMW.

The bulk of the manual (from chapter 8 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, ☞ 3.4. These cross-references are clickable when this PDF is displayed by software that supports the facility.

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 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 appears above a stave in a vocal part. Usually, such words are below the stave, and are called *underlay* (see below), but occasionally alternative words appear above.

A *stave* is a single set of horizontal lines on which notes appear. 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 side in some way. For example, the systems of a string quartet score each contain four staves.

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

2. Installing and setting up 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 on any system with a C compiler and runtime system. However, the only provided building scheme uses Unix-style *configure* and *Makefile* files, which should work in any Unix-like environment, including the Cygwin environment under Microsoft Windows and the command line in MacOS. 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*. There are also some contributed notes on running under MacOS in the file *doc/MacOS.txt*.

In a Unix-like environment, PMW is installed from source in the same way as many other applications. First, download the tarball into a suitable directory. You should end up with a file such as *pmw-5.33.tar.gz*. First unpack the archive:

```
tar -xf pmw-5.33.tar.gz
```

This creates a directory called *pmw-5.33*, 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-5.33
./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 line options for PMW.

- *share/pmw/macros* is a directory that contains files of 'standard macros' (☞ 7).

- *share/pmw/MIDIperc* contains translations between names such as 'acoustic bass drum' and MIDI pitches for untuned percussion voices.

- *share/pmw/MIDIvoices* contains translations between names such as 'piano' and MIDI voice numbers.

- *share/pmw/PSheader* is a header file for PMW's PostScript output.

- *share/pmw/psfonts/PMW-Music.pfa* and *share/pmw/psfonts/PMW-Music.otf* are both copies of the PMW-Music font, in two different formats. The first is a PostScript Type 1 font and the second is an OpenType font.

- *share/pmw/psfonts/PMW-Alpha.pfa* and *share/pmw/psfonts/PMW-Alpha.otf* are both copies of an auxiliary music font, in the same two formats as PMW-Music.

- *share/pmw/fontmetrics/* is a directory that contains files giving character widths and kerning information for the standard set of PDF and PostScript fonts and the PMW music fonts. There are also two files with *.utr* extensions; these contain translations from Unicode code points to the default code points in the *Symbol* font and the *PMW-Music* font (☞ 8.17.1).

As well as the usual *configure* options, there are four that are specific to PMW:

- `--disable-pmwrc` cuts out the code that looks for a file called *.pmwrc* in the user's home directory (☞ 3.4). This code uses Unix-specific functions so must be omitted in other environments.

- `--enable-b2pf` causes PMW to be built with support for using the B2PF library, which converts Unicode text strings from their base to their 'presentation' forms, a facility that is useful for scripts such as Arabic. This option requires the B2PF library to be installed. See section 10.1.3 for details of how to use this facility.

- `--enable-musicxml` causes PMW to be built with support for reading and writing MusicXML files. This is experimental code which is far from complete. See appendix A and appendix B for details.

- `--enable-pdf-default` sets PMW's default main output format to be PDF. Otherwise it is PostScript.

Once you have installed PMW, you can use the *pmw* command to generate PostScript or PDF output, as described in the next chapter. The subsequent chapter (☞ 4) discusses what to do with PMW output.

2.1 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. The PMW command

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* (for PostScript output) or *.pdf* (for PDF output) replacing any existing extension, or being added if there is no extension. The default output format (PostScript or PDF) is specified when PMW is built, but can be overridden by the **-ps** or **-pdf** options. The default output destination can be overridden by 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 -pdf -s 3 -o quartet.pdf quartet.pmw
pmw -ps -f viola -o quartet.ps -midi /tmp/quartet.mid quartet.pmw
```

3.1 General command line options

These options apply to both PostScript and PDF output.

-a4ona3
There are several directives that control the size of the page images PMW produces (☞ 8.5). In the common case, this size is identical to the paper size, in which case one image fits exactly onto one piece of paper when printed. However, PMW also supports *two-up* imposition, in which two page images are put next to each other, to be printed on a larger piece of paper. This option specifies that the music images are A4-sized, but are to be arranged two-up, suitable for printing on A3 paper.

-a5ona4
The pages are A5-sized; arrange them two-up, suitable for A4 paper. The **sheetsize** directive should be used to set the image size to A5.

-C *<arg>*
Show the setting of a build-time option and then exit with its value, which is 1 for 'set' and 0 for 'unset'. The possible arguments are b2pf for B2PF support (☞ 10.1.3) and musicxml for MusicXML support (☞ A, B).

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

-drawstavelines
Staves are normally output using characters from the music font. This option causes them to be output as individually drawn lines. It may produce better output in environments where some PostScript/PDF interpreters leave gaps in staves. The default thickness of the drawn stave lines is 0.3 points, scaled according to the relative stave size and any overall magnification. You can change this by giving a number after the option. The units are tenths of a point, so specifying, for example

```
-drawstavelines 4
```

draws stave lines whose thickness is 0.4 points. A value of zero disables the drawing of stave lines. **-dsl** is a synonym for **-drawstavelines**; a non-zero value for this option overrides **-nowidechars**. See also the **drawstavelines** header directive.

-eps

Write the output as encapsulated PostScript. This option is mutually exclusive with **-pdf**. Encapsulated PostScript is useful if the music is an illustration that will subsequently be included in another document. For one-off illustrations, combining **-eps** with **-incPMWfont** is advised so that the PMW-Music font is automatically included. However, 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 (☞ 13). See also the **eps** header directive.

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 list>*

Search the directories in the colon-separated list for *fontmetrics* (*.afm*) files and other font-related files, before searching the default directory that was set up when PMW was installed. This option is useful when you want to make use of a non-standard font in text strings without having to copy files into the default directory. Relative file names are 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 6 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.

-help or **--help**

Output what the default output format is, which optional features are supported, and a list of command line options, then stop. No file is read.

-MP *<file>*

Use the given file as the *MIDIperc* file, instead of the default that was set up when PMW was installed. 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 intalled. 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
```

-midi <*file*>

This option specifies that MIDI output should be written to the given file. This is in addition to the normal PostScript or PDF 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. 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 8.6 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). Fractional bar numbers can be used to refer to uncounted bars. 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.

-musicxml <*file*> or **-xml** <*file*>

When PMW has been built with MusicXML support, this option requests additional output in the form of one or more MusicXML files. See appendix B for more details.

-musicxmlmovement <*number*> or **-xmlmovement** <*number*> or **-xm** <*number*>

Only one movement can be contained in a MusicXML file; this option selects a movement that is then written to a single file. The default is the first movement, unless there are multiple movements and the XML file name contains '%d'. See appendix B for details.

-norc or **-nopmwrc**

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 (☞ 3.4). Note that PMW can be built without support for *.pmwrc* files; when this is the case this option is ignored.

-norepeats or **-nr**

When generating a MIDI output file, do not repeat repeated sections of the music.

-nowidechars or **-nw**

This option stops PMW from using wide characters for staves. It is provided because it seems that some PostScript/PDF 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 output 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 output as CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC. The **-drawstavelines** option overrides this option. See also the **nowidechars** header directive.

-o <*file*>

Write the main output to the given file, or, if a single hyphen is given as the file name, to the standard output. A file name should be given with the required extension (for example, *etude.pdf*, not just *etude*).

-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 layout is enabled (☞ 10.1.110), 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) PostScript 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.

-pdf
 Generate PDF output. This option is mutually exclusive with **-ps** and **-eps**. It is necessary only when PMW has been built with its default output format set to PostScript.

-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 page images by *<n>*. This differs from the PMW **magnification** header directive (☞ 6.12) in that it affects the size of each page, whereas **magnification** applies only to the contents of the page.

-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.

-ps
 Generate PostScript output. This option is mutually exclusive with **-pdf**. It is necessary only when PMW has been built with its default output format set to PDF.

-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 extracted from a score. For example, in a work for choir and orchestra, to create a vocal score from 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 generate an individual instrumental part. See chapter 6 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 details of transposition, see section 8.10.

-V or **--version**

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 (☞ 3.5).

-x‹*selectors*›

Set one or more options for MusicXML output. The selectors are a sequence of words, separated by plus or minus characters. See appendix B for details.

3.2 Options for PostScript output

These options are useful if PostScript output from PMW is sent to a printer that interprets PostScript. They have no effect if the output is viewed on the screen or converted to a PDF, and they give a warning if given when PMW is generating a PDF itself.

-a4sideways

Rotate the page images by 90 degrees. This can be useful for a printer that has a sideways paper feed.

-c *‹number›*

Set the number of copies to be printed as *‹number›*.

-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.

-H *‹file›*

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

-incPMWfont or **-incpmwfont** or **-ipf**

This option causes PMW to include the PostScript Type 1 PMW-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). However, it does mean that each PMW output file is bigger by about 40K for PMW-Music and 31K for PMW-Alpha. Note that this option applies only to PMW's music fonts. See also the **incPMWfont** header directive. The **textfont** directive (☞ 10.1.127) provides a way of including other fonts in PostScript output.

-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.

-tumble

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

3.3 Maintenance and debugging options

The following options are of interest only to a PMW maintainer. They are listed here for completeness, but few details are included. There is further information in a file called *HACKNOTES* in the *doc* directory of the PMW distribution.

-d *<selectors>*

 Write debugging information to the standard error file. The selectors are a sequence of words, separated by plus or minus characters.

-dbd *<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. This option contrains the amount of output that is produced by −d+bar (which it implies).

-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.

-errormaximum *<number>*

 This option sets the maximum number of errors that may occur before PMW gives up. The default is 40. This facility is provided mainly so that a test of errors can generate a large number of them. **-em** is an abbreviation for **-errormaximum**.

-MacOSapp *<directory>*

 This option is provided for use when PMW is called from within a MacOS 'wrapper' app. It specifies the directory in which all 'resources' (such as font information) are to be found. This is currently an experimental feature.

-MF *<directory list>*

 Search the directories in the colon-separated list for the PostScript music font files, before searching the default directory that was set up when PMW was installed.

-SM *<directory>*

 When looking for 'standard macro' files, search the given directory instead of the one that was set up when PMW was installed.

-testing [*<number>*]

 This option is used by the scripts that do automatic testing. It cuts out the time and version from the output, and also does not include the standard PSheader file for PostScript. The version information is also omitted before any error messages. This makes it straightforward to compare the significant output from different versions of the program. A non-zero value can be used for fine control of testing behaviour.

3.4 Setting default command-line options

In Unix-like environments there is a simple facility for specifying options that you always want to be set. This may not be available in other environments. 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.mid
```

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** or **-nopmwrc**, the *.pmwrc* file is not used.

3.5 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:

```
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 follows. 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. It is rounded up to 0.5 points.

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 (see also the **layout** header directive (☞ 10.1.63).

'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.

3.6 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:

```
** Error: incorrect bar length: too long by 1 quaver
   Detected near line 24 of K495.pmw
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, in which case there is a final message '`** No output generated`.' 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'.

3.7 Return codes

PMW gives the C return code `EXIT_SUCCESS` (which is normally zero) when it has successfully generated some output, even if one or more warnings were given. Otherwise, it returns `EXIT_FAILURE` (which is normally 1).

4. Processing PMW output

Any available PDF viewer, including many browsers, should be able to display or print a PDF produced by PMW using the **-pdf** option, because PMW always includes its music font and any additional text fonts within the PDF. For PostScript output, however, it is a bit more complicated because PMW's music font has to be made available to the processing software. If you are using PDF output, you can ignore the next three sections.

4.1 PostScript processors

GhostScript is an application for processing PostScript. It can display it on the screen or convert it to a PDF via its *ps2pdf* command. Such a PDF is uaually smaller than a PDF written directly by PMW because *GhostScript* can disassemble fonts and include only those characters that are actually used, whereas PMW includes entire fonts in its PDF output. Other document viewers such as *evince* make use of *GhostScript* when processing PostScript files.

Any PostScript processor must have access to the PMW-Music font when handling PMW output files. If it cannot find a font, it makes a substitution, which in the case of the music font usually results in nonsense. One easy way to ensure the availability of the music font is to use the **-incPMWfont** option, which causes the music font to be included within the PostScript output (☞ 4.2). Alternatively, you can set up *GhostScript* so that it knows where to find the music font (☞ 4.3).

GhostScript's basic *gs* command writes information about which fonts it is using to the standard output, but it is quite primitive, being controlled via the command line. A more friendly interface is provided by front-end 'wrapper' applications such as *gv* or *evince*, which have the added advantage of showing thumbnails of all the pages.

4.2 Including music fonts in the PostScript file

If you use the **-incPMWfont** option on the *pmw* command line, or put it in your *.pmwrc* file, or use the **incPMWfont** directive in your input file, PMW includes the PostScript Type 1 version of the PMW-Music font and/or the PMW-Alpha font in every output file that needs them. The resulting files are freestanding PostScript files that should be printable or viewable without the need to take any action concerning the music fonts. However, PostScript Type 1 fonts are gradually becoming obsolete, so this facility should be used with caution. Also, the files are larger by about 30–40K for each of the two fonts. It is not possible to embed the OpenType version of PMW-Music within a PostScript program.

4.3 Setting up for GhostScript

If you do not use the **-incPMWfont** option, *GhostScript* needs to be told where the PostScript music fonts are before it can correctly display a PostScript output file or convert it into a PDF. The easiest way of doing this is to set the `GS_FONTPATH` environment variable, for example:

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

A more complicated alternative is to install symbolic links from a suitable font directory to PMW's *psfonts* directory. You can find out which directories *GhostScript* searches for its fonts by running the following command:

```
gs -h
```

For example, if */usr/share/fonts/gsfonts* is in the 'search path' given by the above command, you might use these commands for the PMW-Music font:

```
ln -s /usr/local/share/pmw/psfonts/PMW-Music.pfa \
      /usr/share/fonts/gsfonts/PMW-Music.pfa
ln -s /usr/local/share/pmw/psfonts/PMW-Music.otf \
      /usr/share/fonts/gsfonts/PMW-Music.otf
```

Similarly, if you are going to make use of the PMW-Alpha font:

```
ln -s /usr/local/share/pmw/psfonts/PMW-Alpha.pfa \
      /usr/share/fonts/gsfonts/PMW-Alpha.pfa
ln -s /usr/local/share/pmw/psfonts/PMW-Alpha.otf \
      /usr/share/fonts/gsfonts/PMW-Alpha.otf
```

Using either of these methods should enable *GhostScript* and other viewers such as *evince* that make use of it to process PMW's PostScript output correctly.

4.4 Watching a changing file

GhostScript and *evince* can display both PDF and PostScript output files. The *gv* command has an option called **-watch** that causes the file to be re-displayed whenever it changes. This happens automatically for *evince*. If you leave one of these viewers running, you can edit the input and reprocess it with PMW, and the viewer will automatically display the new output.

4.5 Problems with displaying staves and bar lines

By default, staves and bar lines are output using characters from PMW's music font. Some PostScript/PDF 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 (☞ 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.

4.5.1 Missing staves

Staves are normally output using two characters that are 100 points and 10 points wide, respectively, at the default magnification. Some PostScript/PDF 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. There is also a **nowidechars** directive that can be included in your input file.

4.5.2 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 drawing commands instead of as characters. This option overrides **-nowidechars**. There is also a **drawstavelines** directive that can be included in your input file.

4.5.3 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 drawing commands instead of as characters. There is also a **drawbarlines** directive that can be included in your input file.

4.6 Printing PMW output

In modern Unix-like environments, the CUPS printing system is usually set up to handle PDF and PostScript files automatically, so it you can display a PMW output file on the screen, it should print without problems.

5. 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 8. We start with a file that contains the first few bars of *Happy Birthday*.

```
heading "|Happy Birthday"
breakbarlines
underlaysize 9.5
notespacing *1.1
key F
time 3/4
unfinished

[stave 1 treble 1 text underlay]
"Hap-py birth-day to you, hap-py birth-day to you. Hap-py"
[nocheck nocount] c-.; c= |
dcf | E c-.; c= | dcg | F c-.; c= |
[endstave]

[stave 2 bass 0]
[nocheck] r | F. | C. | c d e | F. |
[endstave]
```

Let's suppose the file is called *bday*. If you have PMW installed, you can process such a file by running this command:

```
pmw -pdf bday
```

If there are no errors in the notation, there will be no output on the screen, but a new file called *bday.pdf* will have been created. The **-pdf** option overrides PMW's default output format. You should be able to open the PDF with any available PDF viewer. If you want to test PostScript output, run

```
pmw -ps -ipf bday
```

This will make a file called *bday.ps* which includes PMW's music font (requested by the **-ipf** option). This output can be viewed using a PostScript viewer such as *GhostScript*, using its *gs* command or a wrapper such as *gv* or *evince*. See chapter 4 for more details about handling PostScript output. In both the PDF and PostScript cases, the output should display this:

Happy Birthday
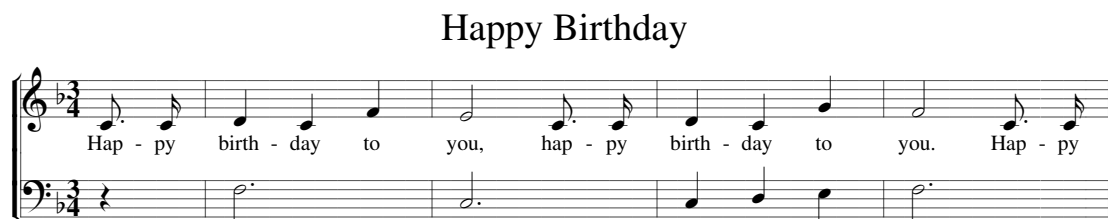
If there are errors in the input file, one or more messages will be written to the standard error file, and should therefore appear on your screen. The messages should be self-explanatory. Correct the error(s), and try again.

We now explain what the different parts of this input file mean to PMW. The data is in two parts: first there is header information, such as the textual heading and key and time signatures for the piece, and then the music for each stave is given separately. The header in this example contains seven *header 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 "|Happy Birthday"
```

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 outputs 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 attached to a stave 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 process 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 F
```

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 output as 𝄴 and 𝄵 respectively.

```
unfinished
```

The final directive tells PMW that this is not a complete piece of music; this stops it from putting a thick bar line at the end. The header 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:

```
"Hap-py birth-day to you, hap-py birth-day to you. Hap-py"
[nocheck nocount] c-. c= |
d c f | E c-. c= | d c g | F c-. c= |
```

The vocal underlay is given as a text string 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 'hap-py' and 'birth-day'. PMW supplies as many hyphens as necessary to fill the space between them on the page. Text strings are not restricted to just the characters on the computer keyboard; see section 8.17.4 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. It is possible to turn this check off, and this has been done for the first bar by including **[nocheck]** (you can also turn it off for the whole piece). The first bar also contains **[nocount]** because it is conventional not to include an 'upbeat' bar at the start of a piece in bar numbering (☞ 6.2.1).

The notes are encoded using their familiar letter names. Because we set the current octave to be octave 1, the letter `c` in the first bar represents middle C. To raise a note by an octave it is followed by a quote character, so `c'` would give the C above middle C. A note can be lowered an octave by following its letter with a grave accent.

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 third 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 quaver from a crotchet. A dotted note is coded by adding a full stop, as in the first, third, and fifth bars. An equals sign (two flags) signifies a semiquaver. Sequences of notes that are shorter than a crotchet are automatically beamed together unless separated by a semicolon (☞ 6.1.4). The convention in vocal parts is not to beam notes that are sung to different syllables, which is why semicolons are used here.

The only new feature in the second stave is the use of the letter `r` for a rest. This example has no accidentals, but if you need one, it is entered before the note letter (just as accidentals are written 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 support for half accidentals (☞ 11.6.2).

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 bars of the second stave could equally well appear like this:

```
[nocheck]r|F.|C.|cde|F.|
```

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 at most places where 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, and some people even like to keep each bar on a separate line like this:

```
[nocheck] r | @0
F. | @1
C. | @2
c d e | @3
F. | @4
```

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.

5.1 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 conveniently used for text strings such as *mf*, or *ff* which may appear many times in a piece. A macro definition can specify exactly how you would like such a string to appear – what size, which font, etc. Several files of 'standard macros' are provided with PMW (☞ 7). These contain basic macros for dynamics, fingering, figured bass text, and more. There are also a number of macros contributed by users in the *contrib* directory of the distribution tarball. As a brief introduction to macros, 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 header 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 in italic and the *f* in bold italic, as is commonly done. (See sections 6.3.2 and 8.17 for explanations of how the above string achieves this.) The example also specifies that the string is to be positioned 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 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 8.2.

6. 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.

6.1 More about notes

This section describes some more common facilities related to notes.

6.1.1 Note types

PMW can handle eight different kinds of note, from breves to hemi-demi-semiquavers. The encoding for some of them was introduced in the previous chapter. For notes longer than a minim the plus character is used to double the duration, and for those shorter than a semiquaver, the minus and equals characters can be combined. The complete set is as follows:

If a stave consists mostly of notes that are shorter than a crotchet, some typing can be saved by the use of the **[halvenotes]** stave directive, which halves the length of subsequent notes. For more detail see section 12.2.37.

6.1.2 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. See section 11.6.8 for details about other special types of rest.

6.1.3 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. It is also possible to repeat sequences of more than one bar (☞ 8.2.5).

6.1.4 Beams

Notes that are shorter than a crotchet are automatically beamed together within a bar, except when a note is followed by a beam breaking character. A semicolon breaks the beaming completely, whereas 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 (☞ 11.7.)

White space is permitted between the last note of a beam and any breaking character (semicolon or comma).

6.1.5 Triplets

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

You can change the way triplets are handled 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 (☞ 10.1.141, 10.1.142, 11.6.31, 12.2.117).

6.1.6 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:
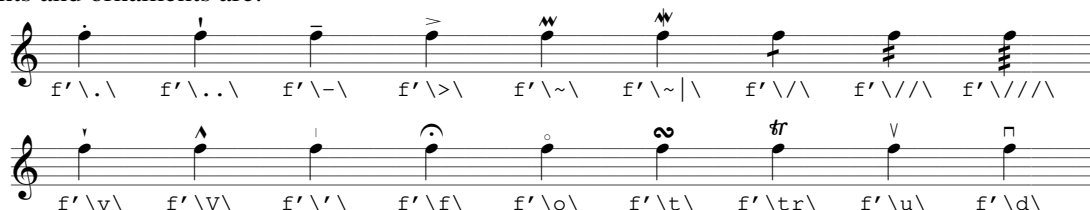
The codes `\v\` and `\V\` are used for small and large 'vertical wedge' accents because of the similarity of shape, though the accents themselves may be the opposite way up to the coding letters, depending on whether they appear above or below the note. The other mark that looks like the letter V is 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 11.6.17 onwards.

6.1.7 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.

6.2 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 typesetting the kind of examples that appear in this manual, the checking can be entirely suppressed by using the header 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.

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 using an exclamation mark (`R!`), 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. If a MIDI file is generated, whole bar rests always take up the length of time implied by the time signature.

6.2.1 Bar numbers

The **barnumbers** header directive is used to request automatic bar numbering above the top stave of systems. There are several options that control where the numbers appear and in what form.

```
barnumbers line
```

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

```
barnumbers 10
```

This causes PMW to output a bar number every 10 bars. You can choose any number you like; specifying 1 causes every bar to be numbered. If the word 'boxed' or 'roundboxed' appears after the directive name, the numbers are enclosed in a rectangular box with mitred or round corners, respectively. 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 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 (☞ 12.2.8).

6.2.2 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 8.3 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
```


6.3 More about underlay (lyrics)

PMW supports overlay (words 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.

6.3.1 Multi-note syllables

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

```
"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 outputs 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 positioned underneath the first string.

6.3.2 Special characters and font changes

The computer keyboard does not contain all the characters that are needed for 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 may not be directly available on the keyboard:

`\a'`	outputs á
`` \a` ``	outputs à
`\a.`	outputs ä
`\a^`	outputs â
`\ss`	outputs ß

For example, the input string `"\it\sch\o.ner"` becomes *schöner*. Using such escapes makes it possible to encode any character using only ASCII codes. However, PMW does also recognize UTF-8 encoding in character strings. 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***

There is an in-depth discussion of text fonts and character encodings in section 8.17.1. Section 8.17.4 has more details about escape sequences, and there is a list of available text characters and their escape sequences in chapter 15.

6.3.3 Spacing

Within a bar, PMW ensures that the syllables of underlay text do not crash into each other, by spreading out the notes if necessary. **Warning:** If use of the **layout** header directive (☞ 10.1.63) 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, is replaced by 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 is not output. 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+
```


6.4 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 placed 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. It is also possible to adjust the horizontal position by reference to the musical offset (☞ 11.9.1). 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\      outputs a minim
\*c\      outputs a crotchet
```

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

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

```
["A"]
```

PMW outputs such strings in a fairly large font, enclosed in a rectangular box; there are options to change these settings (☞ 11.11).

6.5 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.
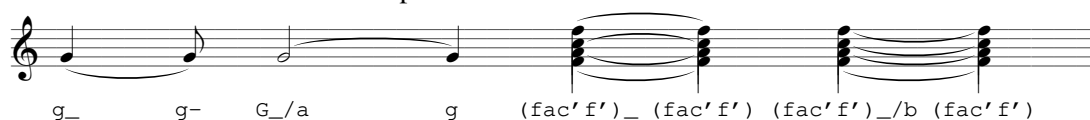
When two single notes of different pitches are connected by a slur, the same notation (an underscore) can be used. However, for chords, the **[slur]** directive (see below) is required to draw slurs, because if two chords are joined by an underscore, the notes in each that are of the same pitch are joined 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, and for introducing gaps (☞ 12.2.76).

6.6 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. A number of different mark styles are available (☞ 10.1.109). 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 included in text strings as follows:

"\mf\c"   gives ⊕
"\mf\d"   gives 𝄋

The escape sequence \mf\ changes to the music font, full details of which are given in chapter 16.

6.7 Hairpins

Crescendo and diminuendo '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 (☞ 11.5).

6.8 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.

6.8.1 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** header directive is used to alter this. In the most simple case it is followed by a single dimension, which sets the same spacing for all staves. This directive may also be followed (optionally after a single dimension) by a list of stave numbers and spacings, each pair being separated by a slash. These set spacings for specific staves; each dimesion is a 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, and 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 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
```

If there is more than one **[sshere]** in the same stave in a single system, the last absolute setting (if any) is used, with any relative changes acting cumulatively. For more details see sections 10.1.120 and 12.2.89, which also describe facilities for adjusting the amount of space above a stave.

6.8.2 System gap

The distance between systems is called the 'system gap', and is set by the **systemgap** header 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]**. For more details see section 12.2.74.

6.8.3 System separators

It is common in scores to have two short thick lines between systems at the lefthand side, to emphasise visually where a new system begins. PMW can be configured to output such lines; see section 10.1.126 for details.

6.8.4 Brackets and braces

By default, PMW joins together the staves that comprise a system with a bracket. 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 header 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.

6.8.5 Initial text

At the start of a piece it is common to show the names of the voices and/or instruments to the left of the first system. This is done by giving a string in quotes as part of the **[stave]** directive.

```
[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 (☞ 12.2.91).

6.9 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, with the name of the instrument mid-way between the two staves. This can by done by adding /m to the relevant string (see the example in the next section).

6.9.1 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 for the first one so that the staves appear 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 visible 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 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.

6.9.2 The [reset] and [backup] directives

If a 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 |
```

There is also a **[backup]** directive, which just backs up by one note.

6.9.3 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 nothing is output. It is often referred to as an 'invisible rest'. See section 11.6.8 for more details about special kinds of rest.

6.9.4 Coupled staves

Keyboard music sometimes includes sets of beamed notes that extend over both staves. These can be handled 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]
```
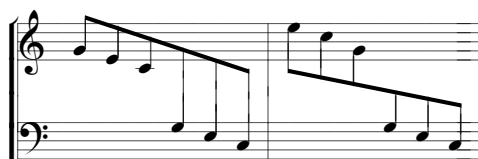

The upper stave has been 'coupled' downwards to the lower one. When this is done, any note in the upper stave that is lower than middle C is put on the lower stave. Notice the use of invisible rests on the lower stave to fill the bars without showing anything. An alternative approach is to use **[couple up]** to get notes from the lower stave that are higher than middle C put 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 position notes 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.

6.10 Heads and feet

We introduced the **heading** directive in chapter 5. You may have as many **heading** directives as you like at the start of a piece. By default, the first two appear 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 outputting 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 be placed 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 is at the same level. This makes it possible to have 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 for the bottom of the first page. The escape sequence \c) is useful in footings; it becomes ©. Both **heading** and **footing** apply to the first page of a piece only. To output 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 outputting page numbers at the top or bottom of each page. There are three escape sequences for inserting a page number into a string:

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

This is a typical example:

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

It puts the page number centrally at the foot of each page (unless there is also a **footing** directive for something different on the first page).

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

This example causes page numbers to appear at the top of each page other than the first, alternately on the left and right. Even numbers are on the left, 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. Page numbers are sometimes needed outside these 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 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 16. 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 current position, so that the even page number will appear 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 appear to the right of the normal margin, so that the subsequent leftwards movement brings the current point back to the margin.

Another common requirement is to put page numbers higher up the page than PMW normally starts. 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.

6.11 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:

```
linelength 480
pagelength 720
```

These are suitable values for 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 positioned centrally on the page, so changing the line length changes both margins symmetrically. PMW defaults to A4 paper, but 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 music systems. However, it does not include the space for footings, which always start 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** header 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. This example reduces all the distances by a factor of 0.92:

```
notespacing *0.92
```

However much you reduce the notespacing, PMW will not allow notes to be output 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.

6.12 Magnification

The standard size of music formatted by PMW has a distance of 4 points between stave lines. The **magnification** header directive can be used to cause it to output 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** (☞ 10.1.119) that can be used to alter the magnification for individual staves, relative to the overall magnification. There are no restrictions on the values that can be given for the magnification.

When a magnification is specified, everything 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 altered.

6.13 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 for a part. At the very least, the headings will probably be different, and you may want to have cue notes in parts but not in the score. You may also want to output 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 start of a file might be 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 output, and if so, sets the magnification to 0.9. Otherwise it sets it to 1.3. PMW considers that a score is being output 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 set up 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 extract 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 (which it overrides), 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 header portion. Here is an example that shows how to notate 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 output at the cue note size.

7. Standard Macros

The use of macros is introduced above. Several files of pre-defined macros are provided with PMW. They contain shorthands for a number of common requirements. Suggestions for additions are welcome.

- **FBass** contains macros for figured bass notation.

- **Fingering** contains macros for fingering numbers.

- **StdMacs** contains macros for a number of common items, including dynamics.

The ***include** preprocessing directive (☞ 8.2.6) gives access to these files, for example:

```
*include "Fingering"
```

However, you must not have a file of your own called **Fingering** because that will take precedence.

7.1 Figured bass macros

The standard macro file called **FigBass** contains a general macro for figured bass numbers, along with some shorthands for common figurings. You can change two aspects of these macros by defining either or both of the macros `FBI` or `FBU` before including the **FigBass** file. `FBI` specifies initial text for the start of each figured bass string. It defaults to `\mf\z` which uses the music font to insert a small amount of white space – this makes for better alignment with the notes. `FBU` specifies the type of text to use. It defaults to `/fbu`, which specifies text at the figured bass size (see the **fbsize** directive), positioned at the underlay level. This ensures that the top line of figured bass numbers are all at the same level. You can change this to `/fb`:

```
*define FBU /fb
*include "FigBass"
```

In this configuration, each figuring's vertical position depends on the note above. The basic figured bass macro is called `FB`, and it takes up to three arguments, which are output one below the other. It is placed immediately before the note to which it applies, as in these examples:

```
&FB(3) g   &FB(3,4) g   &FB(7,2,4) G
```

There are three macros for isolated accidentals, called `FBF` (flat), `FBN` (natural) and `FBS` (sharp). There are also a number of shorthands for common figurings. All of them are shown below:

The names of the macros are intended to suggest their contents, with 'b' for 'below', 'c' for 'cross', 'f' for 'flat', 'n' for 'natural', 's' for 'sharp', and 'u' for 'underscore'. There are two versions of some crossed numbers, the plain 'c' versions using a character in the music font, whereas the 'cp' versions use a normal digit overprinted with a '+', which makes the cross more prominent.

7.2 Fingering macros

The standard macro file called **Fingering** contains definitions of macros whose names are &0, &1, &2, &3, &4, and &5, plus one called &THUMB for the cello thumb sign ♀. Like the figured bass macros, these must appear before the note to which they refer. There is an issue if you want to use both the figured bass and the fingering macros at the same time, because there is a name clash. For this reason, a prefix can be set for the fingering macros. If your input is like this:

```
*define FINGERPREFIX f
*include "Fingering"
```

the names of the fingering macros become &f0, &f1, etc. You can also change the size of the numbers by setting FINGERSIZE for the numbers and THUMBSIZE for the thumb indicator. Finally, you can set FINGERTYPE to adjust the position. The default is /a/c, which puts the fingering above the note, centred. An individual number can be forced below the note by following the macro call with /b, but if you want them all below, setting FINGERTYPE may be more convenient. There is more discussion of fingering and an example below (☞ 11.10).

7.3 Miscellaneous macros

The standard macro file **StdMacs** contains definitions for a number of common musical notations.

7.3.1 Dynamics and common instructions

The meaning of the following macro names should be obvious:

```
f ff fff fp fz mf mp p pp ppp sf sfz fsubito fsub psubito psub
ppsubito ppsub animato cantabile atempo cresc decresc semprecresc
dim pocorall pocorit ponticallo naturale legno rall rit simile
sonore sotto tempo1 ten tranq
```

7.3.2 Slur shorthands

The macros sa and sb are shorthands for starting a slur above or below the notes respectively. Additional options for the slur may be given as an argument to the macro call. For example, to move the start of a slur up and to the right one might have &sa(/u4/r2). There is also a macro called es to end a slur, where again options may be given as an argument.

7.3.3 Octavo marks

The macros S8a and S8b notate the start of an '8va' marking above or below the stave, respectively, with E8a and E8b marking the end. There a discussion of how these work below (☞ 11.12.5).

7.3.4 Piano pedal marks

The macro ped puts a conventional 🎹 sign below the stave, and pedstar outputs ❋. There also three macros for pedal markings with horizontal lines: &pedline follows 🎹 with a horizontal line that continues till &pedend is reached. Intermediate 'blips' can be created with &pedblip. Here is a simple example:

```
&ped C e &pedstar g | &pedline Cdg | &pedblip geC &pedend |
```

An optional argument for `&pedline` is a downward movement (default 6) and an optional argument for `&pedblip` is the width of the blip (default 4). There is discussion of how these macros work in the description of the **[linegap]** directive (☞ 12.2.42).

8. PMW reference description

Earlier 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. PMW has many in-built features for common music notation. More unusual requirements can be met by making use of PMWs drawing features, described in chapter 9. Some examples can be found in the *contrib* directory in the PMW distribution.

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

```
tupletfont <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 **tupletfont**:

```
tupletfont 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 (☞ 8.16).

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 *Header directives* and *Stave directives* (chapters 10 and 12). The chapter in between, *Stave data* (chapter 11), contains the specification of all items other than directives that may appear as part of a stave's data.

8.1 Format of PMW files

A file containing input for PMW is a text file in 8-bit UTF-8 Unicode format. However, non-ASCII characters (those with code points greater than 127) are recognized only within text strings. All other notation uses only ASCII characters. You can use any available text editor or wordprocessor to create such files. 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
```

8.1.1 Line breaks

Line breaks in a PMW input file are equivalent to spaces, except in these circumstances:

- When a line contains a comment (see above), all characters from the introductory @ character to the end of the line are ignored.

- Line breaks are not permitted within the options that follow a note between two backslash characters, whereas other white space is allowed.

- When a directive name is followed by a list of stave numbers, these can be separated by commas and/or white space, but must all be on one line.

- Line breaks are not permitted within the arguments of a macro call (☞ 8.2.2), nor within the argument of a string repetition (☞ 8.2.5).

- Preprocessing directives (☞ 8.2) always take up a complete line of their own, and may not continue onto subsequent lines.

Line breaks inside quoted strings are converted to spaces.

8.1.2 Ignoring line breaks

The restrictions on line breaks just described may occasionally cause a very long line to be required. For example, a macro that generates a number of bars may have quite a long replacement string. To aid readability in such cases, a feature is provided for joining two or more input lines into one long line. If an input line ends with three ampersands (&&&), the ampersands and the newline are removed, and the next line is concatenated. This happens before any other processing. For example,

```
*define &&&
cresc &&&
"Crescendo"/b
```

is converted into the single line

```
*define cresc "Crescendo"/b
```

before being processed.

8.1.3 Macro insertion and repetition

In addition to its use in line concatenation, the character & is an insert character that is recognized at any point in the file. It must be followed either by an asterisk and a number, or by the name of a previously-defined macro. The former syntax is used for repeating a sequence of characters (☞ 8.2.5). If a macro name follows, the macro's contents are inserted at that point – for details, see the description of the ***define** preprocessing directive in section 8.2.2. If a literal & character is actually required in the input, it must be entered as &&.

8.1.4 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.

8.1.5 Header information

A PMW file starts with a number of directives collectively known as the *header*. 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 file contains multiple movements (☞ 8.1.7) there can be a set of header directives for each movement, though some parameters (for example, the magnification) can be set only at the start.

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 moved onto the next page. This gives a way of producing a title page followed by pages of music, all from a single input file. The header is terminated by the first unquoted opening square bracket in the file, and may be completely empty.

8.1.6 Stave information

Following the header 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 12.2.90. There may be up to 63 normal staves, numbered from 1 to 63. They can be defined in any order, though they are always output in numerical order down the page. Data may also be supplied for a stave numbered 0, which has special properties (☞ 8.18). 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 creating concert posters.

8.1.7 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. If you know that each movement will always start on a new page, you might prefer to keep each movement in a separate file. (See the ***include** preprocessing directive (☞ 8.2.6) for a way of including a common header when processing each 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]** directive, which must appear within a header or following the information for a stave. After this there may appear a new set of header directives, followed by the staves for the new movement. The general format of a complete PMW input file is therefore as follows:

Header information
First stave of first movement
Second stave of first movement
…
Last stave of first movement
```
[newmovement]
```
Supplementary header 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 current 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 (☞ 12.2.53). In general, most parameters that can be set by header 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 current note spacings by the given factor. Such a change does not persist into the next movement. Of the parameters whose values persist, most may be changed by header directives at the start of the new movement. However, the following directives

may appear at the start of the first movement only: **b2pffont**, **drawbarlines**, **drawstavelines**, **eps**, **incPMWfont**, **landscape**, **magnification**, **maxvertjustify**, **midifornotesoff**, **musicfont**, **nokerning**, **nowidechars**, **output**, **page**, **pagelength**, **righttoleft**, **sheetdepth**, **sheetsize**, **sheetwidth**, and **textfont**.

8.2 Input preprocessing

Each logical input line (possibly after concatenation of multiple physical lines using the `&&&` facility) is *preprocessed* before being interpreted. Preprocessing allows the input to be modified in various ways, and is controlled by special preprocessing directives. These may occur at any point in an input file; in the header, in the middle of a stave's data, or between staves. A preprocessing directive must be at the start of a line, preceded by an asterisk (spaces before the asterisk are permitted). It takes up the rest of the line; it may not continue onto the next line. Very long preprocessing directives can be input on multiple physical lines by making use of the input concatenation feature (☞ 8.1.2).

8.2.1 *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.

8.2.2 *Define

The ***define** directive is used to define *macros*. A macro is a name for a string of characters, usually for something that is needed more than once. Using a macro ensures consistency, and a short name saves typing. 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, up to but not including the newline, is remembered and associated with *name*, which must consist of a sequence of letters and digits and be no longer than 256 characters. 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.

8.2.3 Macro calls

When an input line is preprocessed, the character `&` is used as a flag character to trigger the substitution of text wherever it appears (except when it follows the @ comment character). There are three cases:

- The sequence `&`*name* in the input is replaced by the remembered text of the macro called *name*, which must have been previously defined.

- The sequence `&*` followed by a number and a parenthesized argument is an input sequence repetition, details of which are given below (☞ 8.2.5).

- If a real ampersand character is required in the input, it must be entered as `&&`.

Any other character following `&` causes an error message to be output. The ampersand is then ignored.

To avoid ambiguity, a semicolon character can optionally be used to terminate the name in a simple macro 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 such 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 (☞ 8.2.7). An example of the use of a simple macro is given in section 5.1.

8.2.4 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 argument values for a macro call are passed in a pair of parentheses that immediately follow the macro name without intervening white space. A semicolon that follows the closing parenthesis is not treated specially and is interpreted as the next input item.

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 parentheses.

Note: When a macro is defined without parentheses, as in the previous section, an opening parenthesis following the name in a macro call is not treated as introducing arguments.

Up to 20 arguments may be used. The example macro above 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 an argument, and an internal closing parenthesis does not terminate the macro call. For example:

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

• To include an unmatched opening or closing parenthesis or a comma that is not within nested 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. Each default argument must be no longer than 256 characters (arguments in macro calls can be longer). If a macro is defined or called with more arguments than are referenced in its replacement string, the additional arguments, either explicit or defaulted, are ignored. Here is an example of the use of default arguments:

```
*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           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.

8.2.5 Input repetition

A single bar may be repeated by starting it with a number in square brackets (☞ 11.2) and there is an input shortcut facility for repeating individual notes (☞ 11.6.36). There is also a way of repeating an arbitrary string of input characters during preprocessing. This is coded as an ampersand followed by an asterisk, a number, and then a single argument in parentheses, in the same format as a macro argument. This example generates the same sequence of bars three times in succession:

```
&*3(abc|def|[2]R!|)
```

This example repeats a group of semiquavers four times, breaking the beam after each group:

```
&*4(g=a=b=c'=;)
```

An error occurs if more than one argument is given. This construction may be nested and may contain ordinary macro calls. Like a macro call, the entire argument must be on a single logical input line; if it is very long, line concatenation (☞ 8.1.2) can be used to spread it over multiple lines in the input file.

8.2.6 *Include

This directive can be used to include one file within another. For example, the same standard header 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. If a file whose name contains no slashes cannot be found, it is

sought in the 'standard macros' directory (☞ 7). Included files may be nested. That is, an included file may contain further ***include** directives.

8.2.7 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', 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.

- 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 'transpose' and no number follows, the condition is true if any transposition, including zero transposition, is in force. If a positive or negative number follows 'transpose', the condition is true if that particular transposition is in force.

- 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 is 'PDF', the condition is true if PMW output is set to be a PDF. Otherwise (for PostScript output) it is false. This condition is mainly for testing so that the same input can be used for both kinds of output.

- 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            @ full score reduced
  magnification 0.8
*else                @ 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 transpose
  heading "│(Transposed version)"
*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.

8.3 Identification and counting of bars

PMW identifies bars in its messages using the same number as would be output 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 (☞ 3.5). 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)'.

8.4 Dimensions

The unit of length used by PMW is the printer's *point*. Historically the size of this has varied, but a value of 1/72 of an inch is now established as the standard in digital publishing. 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 positioned 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.

8.5 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. **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 put next to each other on a larger piece of paper, for certain paper sizes. Details of this are given in chapter 3.

8.6 MIDI output

MIDI output is intended for 'proof-hearing' a piece, to check that the input is correct. It does not try to reproduce all the nuances of a performance. In particular, most accents and ornaments on notes have no effect, with just a few exceptions that work in common cases. These are: trills on single notes (for example, e\tr\), notes or chords marked with tremolo lines (for example, g\//\), and **[tremolo]** between two notes or chords, all of which, by default, are 'played' as a number of shorter notes/chords. However, this is not always satisfactory for tremolos, so there are directives to disable it (☞ 10.1.78).

A number of other directives whose names all start with 'midi' are available for controlling the allocation of MIDI voices and channels to staves. The **midichannel** (☞ 10.1.72) header directive is used to specify the allocation of a MIDI voice and/or particular PMW staves to a MIDI channel, and the **[midichannel]** (☞ 12.2.44), **[midivoice]** (☞ 12.2.48), and **[midipitch]** (☞ 12.2.45) 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]** (☞ 12.2.67) directive can be used to force the visible 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** (☞ 10.1.79) or **[midivolume]** (☞ 12.2.49) 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 by the use of **[notes off]** are by default also omitted from MIDI output. The header directive **midifornotesoff** can be used to change this behaviour.

8.7 Headings and footings

There are three different sets of heading/footing directives:

- **heading** and **footing** specify text that appears once, on the first page of a piece or at the start of a new movement.

- **pageheading** and **pagefooting** specify text for the second and subsequent pages of a piece or movement.

- **lastfooting** specifies text for the final page of a piece or (by request) at the end of a movement.

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 appears 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 used (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 appears on the page. If, for example, a copyright footing is defined at the start of the first movement, it will appear 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 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 appear by default on all subsequent pages. Sometimes page headings need to be suppressed at the start of a new movement, for example if they are being used to show 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, and it applies only to the one movement. When a new movement does start at the top of a page there is sometimes a requirement for a special footing 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 and also applies to just the one movement. The value of **lastfooting** can be reset in the new movement if necessary.

8.8 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.

8.9 Key signatures

Standard 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.

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

There is in addition a pseudo-key N, which behaves like C major except that it does not transpose. See the next section for a discussion of key signatures after transposition, and the **printkey** directive (☞ 10.1.105) for how to output key signatures in non-standard ways.

PMW also supports up to 10 custom key signatures called X1 to X10 which are defined by the **makekey** directive (☞ 10.1.69). This can specify arbitrary accidentals, including half-accidentals and double accidentals, at any position on the stave.

8.10 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 any number of semitones less than 61 (5 octaves). A transposition for the whole piece can be specified externally, via the **-t** command line option, or within the input file by the **transpose** header 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, but there is a special pseudo-key N that does not transpose. This is described below. Custom key signatures (X1 to X10) can be transposed, but additional information, supplied by one or more **keytranspose** directives (☞ 10.1.59), must be supplied for this to work.

A piece that is to be transposed should be input with its original key signature(s) specified in the usual way. When **[transpose]** is used to transpose a single stave, only those key signatures that follow the directive in the input are transposed. When transposing a non-custom key signature, the key 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.

Except when using the pseudo-key N, 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 action for individual notes, and the **[transposedacc]** and **transposedacc** directives can be used to suppress it within a part or throughout a piece, respectively.

8.10.1 The non-transposing pseudo-key N

There are some instruments (transposing brass for example) whose parts are often set with no key signatures. This can, of course, be typeset by specifying C major, but if such a part is transposed, it may acquire an unwanted key signature. To avoid this, PMW supports a pseudo-key N, which behaves like C major, except that it does not transpose. It can only be specified on its own, that is, not with a following accidental or `m`.

When key N is in force, notes transpose as usual. The choice of accidental for the new note is influenced by what C major would require. However, there are some special rules that apply only to this kind of transposition.

- If the new note's accidental is redundant, discard it, except in one specific circumstance: when a note that is notated with a natural is the first non-tied note in the bar. The reasoning behind this is that the original natural must be 'cautionary'. Apart from this special case, the behaviour is as if **[transposedacc noforce]** has been applied. However, an accidental can be forced for individual notes by following the original accidental with `^+`, for example `#^+a`.

- Do not use double sharps or double flats unless the original note had a double sharp or double flat, respectively.

- Convert new notes E♯ and B♯ into F and C, respectively.

- Convert new notes C♭ and F♭ into B and E respectively.

8.10.2 Transposition of key and chord names

PMW can automatically transpose the names of standard 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 8.17.

8.11 Time signatures

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 output using the conventional character 𝄴.

- The letter `A` specifies 'alla breve' – equivalent to 2/2 but output 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 output as given. Thus, for example, the time signature `2*C` is shown as 𝄴, but expects there to be four minims rather than four crotchets in a bar, and `2*3/4` is shown as 3/4 but expects three minims in a bar.

There are options for suppressing time signatures at various places, and the **printtime** directive can be used to specify exactly how certain time signatures are to be shown. For example, 8/8 could appear as 3+3+2/8, or only a single, large number could be used.

By default, numerical time signatures use the bold font. However, the **timefont** header directive can specify an alternative. In addition, if **printtime** is used, the normal font-changing escape sequences can be used in the strings.

PMW can handle 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.

8.12 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. It also uses the masquerade facility (☞ 11.6.24) to output a semibreve in the place of a minim, an invisible rest q (☞ 11.6.8) and an invisible bar line │? (☞ 11.1.1) to generate extensions to the stave lines, and **[nocheck]** to disable bar length checks where necessary. The input is as follows:

```
startbracketbar 1
unfinished
[stave 1 soprano 1 key F time C nocheck]
A │ [treble 1 key a$ time c] Rc'd' │ [nocheck] q │?
[endstave]
[stave 2 tenor 1 key F time C nocheck]
C\M+\ │ [treble 1 key a$ time c] Ead' │ [nocheck] q │?
[endstave]
```

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

```
[stave 2 omitempty]
│ @ An empty bar
[treble 1 key a$ time c] Ead' │ [nocheck] q │?
[endstave]
```


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 in the first system.

8.13 Text fonts

PMW supports the use of a number of different fonts, or typefaces, for use in text strings. As well as the standard four (roman, italic, bold face, and bold italic), the use of a symbol font and of the music font 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 **textfont** header directive (☞ 10.1.127) is used to define exactly which fonts correspond to these names:

```
roman          the roman font
italic         the italic font
bold           the bold face font
bolditalic     the bold italic font
symbol         the symbol font
extra <n>      the <n>th extra font
```

The music font can be changed by **musicfont**, but this is likely to be of use only to PMW maintainers. By default, the *Times* series of fonts are used for text, and the *Symbol* font for symbols.

PMW needs access to the 'fontmetrics' (`.afm`) file of every text font that it uses. Fontmetrics files for the standardly available PostScript and PDF 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 (see below). You may also want to supply Unicode translation files (☞ 8.17.2). These can alter a font's encoding and define translations from Unicode code points to code points within the font. The encoding defines which of a font's characters are accessible via numerical code points. PMW supports up to 512 characters per font.

If the font you want to use does not have a fontmetrics file, you can create one in various ways. For example, there is a utility called *ttf2afm* that generates AFM files for TrueType fonts. The *FontForge* open source font editor can create AFM files for any font that it can load. Some fonts contain thousands of characters, resulting in large AFM files. These are readable text files, and you can reduce the amount of work PMW has to do by cutting out any unwanted characters.

PMW's **-F** command line option can be used to provide a list of additional directories to be searched for any files relating to fonts: AFM files (`.afm`), Unicode translation files (`.utr`), and font files (`.otf`, `.pfa`, `.pfb`). These directories are searched first, before those that are installed with PMW.

8.14 Font handling for PDF output

There are fourteen 'standard' fonts that every PDF processor must make available: the Times, Helvetica, and Courier families, plus Symbol and ZapfDingbats. PMW assumes that these are available, and you can use them without any special action. If you want to use a non-standard font, not only must you make available its AFM file (as mentioned above), but the font itself must be made known to PMW so that it can be included in the output. When generating PDF output, only OpenType (`.otf`) fonts are supported. Other kinds of font can be converted to OpenType by tools such as those found in Adobe's AFDKO font development kit for OpenType.

8.15 Font handling for PostScript output

PostScript has 35 'standard' fonts that you can use without the need to take special action. If you want to use a non-standard font, you must make its AFM file available to PMW (as discussed above). By default, PMW will not include the font itself in its PostScript output – you will need to make it available to the PostScript processor that you are using in the same way as for PMW's music font (☞ 4.3). You can, however, cause PMW to include the font in the output by specifying 'include' in the **textfont** header directive (☞ 10.1.127). If you do this, the font must be made known to PMW via the **-F** option.

8.16 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 a 15-point font. There are further parameters 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"
```

In this example, the heading font has 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.

8.17 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. There is no limit to the length of a string. Three characters are treated specially in all strings:

- The single quote character ' and the grave accent character ` are converted into 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*. Amongst other things, it can be used to include specially treated characters, including itself, as literals in a string (☞ 8.17.4).

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

- In vocal underlay or overlay strings, # (sharp), – (hyphen), = (equals), and ^ (circumflex) are treated specially (☞ 11.12).

- 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.

A backslash can be used to remove the special meaning from any of these characters.

In chapter 15 there is a list of the characters in fonts that use the Adobe standard encoding. These characters are accessible using UTF-8 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 *Symbol* font.

For fonts that do not use the Adobe standard encoding, you can use Unicode code points only if a file that translates from Unicode to the font's own encoding is available (☞ 8.17.2). If not, you are restricted to the font's 256 code points.

8.17.1 Unicode and UTF-8 encoding

This section is rather technical. Unless you need to know some of the deep details of character handling or font encodings, you can ignore it and the following two sections, and skip to section 8.17.4 *Escaped characters* below.

Basic fonts in PostScript programs and PDF files can display up to 256 characters, many more than are available on a computer's keyboard. The original 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.

The problem with using many different character codes is that it is hard to switch between them. Even for 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 8.17.4 below.

Historical Note: 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.

8.17.2 Font encoding and Unicode translation files

A font may contain any number of characters, identified by name, but only 256 are accessible by code point. The mapping between code points and character names is the font's *encoding*. Each font has a default encoding for up to 256 characters, but this can be changed. PMW actually sets up two fonts for each text font, so as to make 512 code points available for each PMW font. From the user's point of view, each text font can handle characters whose code points are in the range 0–511.

Fonts whose fontmetrics files specify Adobe standard encoding contain a known set of around 350 characters (they may contain others as well). For such fonts PMW sets up an encoding that makes all these standard characters accessible via Unicode code points (☞ 15). The first 384 code points

(0–383) in the PMW *virtual font* are directly encoded to match Unicode. The remaining 44 standard characters have Unicode code points that are greater than 511 (U+01FF). PMW handles these by encoding them starting at code point 384 and arranging an automatic translation from their high-valued Unicode code points. The remaining code points in the virtual font are undefined.

Fonts whose fontmetrics files do not specify Adobe standard encoding contain an unknown set of characters. Without additional data the only characters that are accessible are those in the font's default encoding, using code points in the range 0–255. Code points in the range 256–511 default to duplicates of the lower range.

Unicode translation files can be used to modify the encoding arrangements just described, and can also be used to set up custom character translations. When PMW loads a font, it looks for an optional `.utr` file in the same directories that are searched for fontmetrics (`.afm`) files (see the **-F** command line option). A `.utr` file can do three things, and its format is very simple:

- In each line, leading white space is ignored.

- Empty lines and lines that start with # are ignored.

- A line that begins with a slash defines an encoding value that modifies the default encoding. The format is:

    /*<character name> <code point> <optional comment>*

The code point can be specified in decimal, hexadecimal preceded by `0x`, or octal starting with `0`. It must be in the range 0–511. These examples come from an Arabic font:

```
/uni0627 65   ALEF
/uni0622 336  ALEF with MADDA ABOVE
```

In this font, character names are based on Unicode code points. U+0627 is the ALEF character; here it is being encoded as character 65, with an accented version as character 336. Code points that are not mentioned in the `.utr` file are unchanged.

- A line that starts with a question mark specifies the Unicode character to substitute when an unsupported Unicode code point is encountered. The rest of the line is an optional comment. For example:

```
?  35  Use code 35 for unknowns
```

- Other lines specify a translation from a Unicode code point to a code point within the font. They must start with a space-terminated hexadecimal number, optionally preceded by `U+` or `0x`, followed by the corresponding font code point, for example:

```
0627 65   ALEF
fe8d 65   ALEF ISOLATED FORM
0622 336  ALEF with MADDA ABOVE
```

As this example shows, more than one Unicode code point can be mapped to the same font code point.

If a Unicode code point greater than U+01FF is encountered that is neither handled by the standard encoding mechanism described above nor handled by a translation file, it is replaced by the font's 'unknown character' code, and a warning is issued.

Unicode translation files with the extension `.utr` are provided for the *Symbol* font and also for the PMW-Music font, though only a small number of characters in the latter correspond to Unicode code points.

8.17.3 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. From release 5.00 a warning is issued

if any such character is encountered. 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.

8.17.4 Escaped characters

From PMW release 4.10 onwards you can 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 on the computer keyboard, 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'      codes for  á
\a`      codes for  à
\a^      codes for  â
\a.      codes for  ä
```

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

```
\c)      codes for  ©
\c]      codes for  ©    (but see below)
\ss      codes for  ß
\?       codes for  ¿
\\       codes for  \
\"       codes for  "    (because " on its own terminates the string)
\'       codes for  '    (because ' on its own codes for ’ )
\`       codes for  `    (because ` on its own codes for ‘ )
\<<      codes for  "
\>>      codes for  "
\--      codes for  –
\---     codes for  —
```

The normal way to include a copyright symbol in a string is to use \c) because this obtains it from 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 use the copyright symbol from the *Symbol* font.

A complete list of all the available special characters and their escape sequences is given in chapter 15. Other escape sequences are summarized in chapter 18. Characters that are not on the keyboard can also be included in strings by giving the character number, in hexadecimal preceded by x or in decimal, enclosed between two backslashes. For example, \xb7\ or \183\ is a bullet character.

The interpretation of string escape sequences happens when a string is first read. This means that characters that are interpreted in underlay text and elsewhere are always recognized as special, however they are coded. For example, if a vertical bar in a heading is entered as \124\ it is still recognized as a left/centre/right separator. If you need one of these characters to appear without interpretation, you can escape it with a backslash. For example, \| encodes a literal vertical bar.

Note: This is a change from the way strings were handled in releases prior to 5.00. In earlier releases escapes were processed later, after checking for special characters.

Characters from the *Symbol* font are 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. Since release 4.35, Unicode encoding can be used for this font because a translation file from Unicode code points to the font's own code points is now provided. Code points less that 256 that are not in the translation file are output unchanged.

To include a single character from the Symbol font without affecting the font of subsequent characters, specify its hexadecimal character number preceded by sx, or its decimal character number

preceded by s, enclosed in backslashes. For example, \s174\ is the → right arrow symbol. This could also be given as \sx2192\, which is the Unicode code point.

8.17.5 Page numbers

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

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

If the page number is even, \po\ inserts nothing, and if it is odd, \pe\ inserts nothing. These are made available for use in heading and footing lines, to enable page numbers to be placed 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 outputs 'page *n*' on the left or right of the page, depending on the value of the page number. The effect of \so\ followed by \se\ or *vice versa* is undefined.

8.17.6 Numbering repeated bars

There is a facility for requesting the same bar to be repeated multiple times. In text strings in such bars, the escape sequence \r\ is replaced by the repetition number. For more details, see section 11.2.

8.17.7 Transposing key and chord names

A special escape sequence is provided to define the names of keys or chords that should be changed by transposition. This makes it straightforward to transpose pieces that show chord names above a line of music or include a key name in the title. The feature applies only to the names of standard keys; there is no support for custom keys or half accidentals.

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 %. 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 a 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 external transposition specified by the **-t** command line option plus any **transpose** header 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'.

8.17.8 The transposition setting

Another use for the \t escape sequence is to insert the transposition value into a string. In this case, \t must be followed by a backslash. For example:

```
heading "(Transposed by \t\ semitones)"
```

The insertion happens when the string is read.

8.17.9 Font changes

Roman, italic, bold and bold italic fonts are available for all text. By default, these use the *Times* series of fonts but can be changed by the **textfont** header 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 *crescendo*. 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
\mf\      change to the music font at full size
\mu\      change to the music font at 0.9 size
\xx1\     change to the first extra font
…
\xx12\    change to the twelfth extra font
```

For example:

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

Note that the letters in the escape sequences are always in lower case. A change of typeface does not persist beyond the end of the text string in which it appears. A quick way of accessing a smaller music font is provided because, when mixed with normal text, music characters at full size are too large.

Changing to SMALL CAPS does not 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** header 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. If applied to the music font, \sc\ has the same effect as using \mu\.

8.17.10 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.

8.17.11 Sizes of text strings

The header directives that specify page headings and footings allow arbitrary font sizes to be given for those texts. Text within a stave uses 10-point fonts by default, but there are various ways of changing this.

The **textsizes** header directive defines up to 20 settable text sizes, which can be accessed by any text string within a stave by means of the /s option. The default for all these sizes is 10 points. Whenever the size of a text font is specified, an associated aspect ratio and/or shearing angle may also be specified (☞ 8.16). In addition there are 10 fixed sizes that are accessed by /S (☞ 11.9.3).

Underlay, overlay, and figured bass have their own separate default sizes, which are set up by header directives. All other text defaults to a settable size that can specified by the **[textsize]** directive, with the ultimate default being settable size 1.

Stave text strings that are not underlay or overlay can be rotated so that they appear at an angle. Text at the start of a stave can be rotated so as to run vertically instead of horizontally – see the description of **[stave]** in section 12.2.90. More detail about text items in stave data is given below (☞ 11.9).

8.17.12 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 outputting a note followed by an equals sign and its metronome mark, or for including sharps and flats in the names of instruments. To include a note from the music font, the following special sequences can be used:

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

Any of the above can include a dot after the note letter to request the dotted form of the note, for example, *c.\. The accidental characters are available as follows:

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

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

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

This is output as:

Maestoso ♩ = 60

Music characters included in character strings with a single asterisk in this way use 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 for the music being typeset is a 10-point font, and it is sometimes useful to be able to access 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 output. Instead they just move the current 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

In addition to those characters that are available via the escape sequences just described, it is also possible to access 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 16. For example, the sequence `\*45\` outputs a crotchet rest. This could also be specified as the Unicode code point `\*x1d13d\`.

If you want to include a long sequence of characters from the music font in a string, it is sometimes more convenient to use a font-changing escape sequence, as described in the previous section, rather than individually escape each character.

8.17.13 Guitar chord grids

Guitar chord grids can be notated relatively straightforwardly as a string of characters in the music font. The grid character itself (character 131) has zero typographic width. If a guitar dot character (116, or 't') follows, 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 output, the current 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 output 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 output them above the first fret you need to move up one and a half times the normal fret distance. This can be achieved by making use of two other special moving characters, '|' and '~' (124 and 126), which move down and up half a fret distance, respectively.

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

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

If you want to add the names of the chords, you can give them as additional text strings that can be separately positioned. Section 11.9 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.

8.17.14 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 header of the first movement. If you just want to prevent kerning between a particular pair of characters, you must insert something between them that has no noticeable effect, for example, a position movement and its opposite.

    `"\rm\Y\*u*d\orkshire"`

In this example is the o is not moved nearer to the Y.

8.18 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 output for stave 0, and any notes that are specified are treated as

'invisible'. However, text items are shown. If the only stave in a movement is stave 0, the only output is the text items.

The intended use of stave 0 is for setting up text items such as tempo marks that appear above the topmost stave, whatever combination of staves is selected. 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 as follows (/ts aligns the text with the time signature):

```
[stave 0 text above]
"\bf\Allegro"/ts 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, but you can use the **copyzero** header directive to have copies of stave zero duplicated 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 appear 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 required is suspended, stave zero is output 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 more than once at any given level, the spacing parameter for the highest numbered stave is used. Selection of a subset of staves is equivalent to the suspension of all others. The default for **copyzero** is:

```
copyzero 1
```

This therefore has the desired effect of outputting stave zero 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 being output 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** header 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.

8.19 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 output. PMW does not suspend staves automatically, so you have to use the **[suspend]** directive in to tell it when to do so (☞ 12.2.98). Resumption is automatic when a non-rest bar is reached, 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 show all the staves in the first system of a piece, even if some of them contain only rest bars. However, there is a header directive called **suspend** that makes it possible to suspend individual staves right from the start (☞ 10.1.124).

When a single stave is being output, suspension normally has no effect, because multiple rest bars are packed up into a single bar with a count 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 potentially suspendable systems, which are undesirable for a single stave. In this case, therefore, **[suspend]** directives are ignored.

Normally, a stave that is not suspended will be output right across the system, with rest bars as appropriate. However, a stave can be tagged with the **omitempty** option (☞ 12.2.90), in which case completely empty bars generate no output. This can be useful for *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.

9. 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 create music notation that is not provided explicitly by PMW. For example, the facility can be used to draw boxes round notes, vertical brackets between notes, and other unusual marks above or below the stave. It can be used with headings or footings to rule lines across the page or to output crop marks.

This chapter is placed here because there are references to the drawing facilities in what follows. However, unless you plan to make use of these features, you can safely skip to the next chapter.

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** header 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.

9.1 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 9.14.) 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:
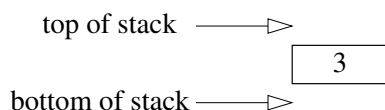
The second item is also a number, so after it is read, the stack is as follows:

```
top of stack  ——————▷
                  ┌─────┐
                  │  2  │
                  ├─────┤
                  │  3  │
                  └─────┘
bottom of stack ——————▷
```
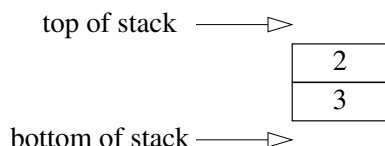
The third item in this fragment is the word 'add'. This is not a number – it is an *operator*. The operators used in PMW drawings are based on those found in the PostScript language. When an operator is encountered, it causes PMW to perform an operation on the numbers that are already on the stack. In 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:

```
top of stack  ——————▷
                  ┌─────┐
                  │  5  │
                  └─────┘
bottom of stack ——————▷
```
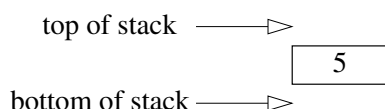
If an operator is encountered that requires more numbers on the stack than there are present, *stack underflow* is said to occur. PMW generates an error message and abandons the drawing. The stack mechanism is very simple, and operates quickly. However, it does make it possible to write very obscure 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* (☞ 9.13) must be used instead.

9.2 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 (☞ 9.14), separated by spaces. These are pushed onto the drawing stack immediately before the function is obeyed, and therefore act as arguments for the function. Here are some examples:

```
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 (☞ 8.2.4).

9.3 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
```

9.4 Mathematical function operators

The following operators make some mathematical functions available in drawing descriptions:

- **cos**: Replace the top number on the stack, which is interpreted as an angle in degrees, with the value of its cosine.

- **sin**: Replace the top number on the stack, which is interpreted as an angle in degrees, with the value of its sine.

- **sqrt**: Replace tne top number on the stack with its square root. The number must not be negative.

9.5 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.

9.6 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.

9.7 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.

9.8 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:

top of stack ——▷

67
45
91
33

bottom of stack ——▷
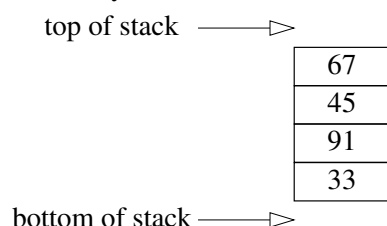
9.9 Coordinate systems

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 (☞ 10.1.51) 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]** (☞ 12.2.42) and **[slurgap]** (☞ 12.2.84) directives.

- A drawing can be associated with the text the precedes 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 (☞ 12.2.90).

9.10 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.

9.11 Graphic operators

PMW follows the PostScript/PDF 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 **stroke** operator. 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.

At the start of an external call to a draw function, the current colour is set to black, the current line width is set to 0.5, and the current setting for dashed lines is no dashes. However, these values are not changed when a drawing function is called from within another drawing function (☞ 9.16).

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:

- **currentcolor**: Push onto the stack the three values of the current colour setting – see **setcolor** below (**currentcolour** is a synonym).

- **currentdash**: Push onto the stack the two values of the current dash setting – see **setdash** below.

- **currentgray**: Push onto the stack the current value of the grey setting – see **setgray** below (**currentgrey** is a synonym). If the current colour is not a grey setting (the red, green, and blue values are not all the same), a grey value is computed using the formula from the NTSC video standard for converting colour television to black and white, as used by PostScript:

  gray = 0.3 x *red* + 0.59 x *green* + 0.11 x *blue*

- **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 other 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** or **setcolor** 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.

- **setcolor** or **setcolour**: This operator specifies a colour for subsequent items drawn by **stroke** or **fill** and also for text that is output by **show**. Three numbers on the stack specify the red, green, and blue components of the colour. Their values must be between 0 (none) and 1 (maximum). This example sets a shade of purple:

      0.5 0.2 1 setcolor

  For shades of grey the three values are identical; see the **setgray** operator below.

- **setdash**: This operator enables **stroke** to draw simple dashed lines. There must be two numbers on the stack: a dash length and a gap length. For example,

      10 3 setdash

  causes any subsequently drawn lines to consist of dashes of length 10, interspersed with gaps of length 3.

- **setgray** or **setgrey**: This operator is used to specify a grey shade for drawn items resulting from the use of **stroke** or **fill**. It has exactly the same effect as **setcolor** with three identical arguments. From release 5.22 a colour or grey setting also applies to text that is output by **show**. 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 outputs a text string (☞ 9.14).

- **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.

9.12 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. Note that if the bar is uncounted, its number includes a decimal fraction (☞ 8.3). If this variable is used in a drawing that is obeyed 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]** or **[slurgap]** directive; the value is positive for a line or slur that is above the stave, and negative for one that is below. Otherwise the variable contains zero.

- **gapx** and **gapy**: When a drawing is being obeyed as part of a **[linegap]** or **[slurgap]** directive, these variables contain the coordinates of the start of the next part of the line or slur. Otherwise they contain 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**: These variables are old synonyms for **gapx** and **gapy**.

- **linelength**: This variable contains the current line length, as set by the **linelength** header 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).

- **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 (☞ 9.24).

- **pagelength**: This variable contains the current page length, as set by the **pagelength** header 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** header 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. When encountered, it pushes *two* values onto the stack. The normal starting position is 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 or page borders. The file *cropmarks* in the *contrib* directory in the PMW distribution contains an example.

9.13 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 cases, a dummy drawing function that is defined but never obeyed can be used for the sole purpose of defining user variable names.

9.14 Text strings in drawings

Text strings can be output from within drawing functions. 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 output 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 (☞ 11.9)
/s<*n*>	output the string at settable size <*n*> (☞ 10.1.128)
/S<*n*>	output the string at fixed size <*n*> (☞ 10.1.128)
/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, outputting 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 outputting a string and continued afterwards. As an example of the use of the text facility, consider music in the sixteenth and seventeenth century style where, instead of using ties that cross bar lines, augmentation dots without notes are placed 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 shown because of the use of **[notes off]** but its pitch is available to the drawing function, which uses it to put 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.

9.15 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 output. There must be two arguments on the stack: the number to be converted, and a string that provides a place to store the converted number. The string may be empty, in which case the roman font is used; if not, the font is taken from the first character in the string. 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 output the current bar number 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 output. 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 (☞ 9.19).

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

9.16 Drawing subroutines

One drawing can be called as a subroutine from within another by means of the **draw** operator. The caller and the called drawing both see the same drawing stack and graphics state (current position, current path, line width, colour, etc). Changes made by the subroutine can therefore affect the caller. The behaviour is as if the **draw** operation is replaced by the text of the subroutine. 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.

9.17 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.

9.18 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 output 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 bar number 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.

9.19 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 output 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 showing these 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.

9.20 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 (☞ 10.1.51).

9.21 Drawing at stave starts

Drawing functions can be obeyed at the start of a stave, as well as, or instead of text. For details see the description of the **[stave]** directive (☞ 12.2.90).

9.22 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.

9.23 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.

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.

9.24 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 (☞ 9.13); 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.

10. Header directives

A header section in a PMW file contains *header directives*. Each starts with a keyword, and sometimes the keyword is followed by numerical or other data. The case of letters in keywords is not significant. It is usual to start a new line for each header directive, but this is not a requirement. A header 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 header information is mandatory because there are default values for all the parameters; a header section may be completely empty.

Most of the header directives may appear at the start of a new movement as well as at the start of the input, but a few may only appear at the very start of the file, that is, only in the first movement (☞ 8.1.7). In general, header 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.

10.1 Alphabetical list of header directives

The header directives are now described in alphabetical order. Several of them take a list of numbers as data. In all cases, such numbers can be separated by commas and/or spaces, but the whole list must all be on one line. The values set by most directives persist from movement to movement. When this is not the case, it is noted in the description.

10.1.1 Accadjusts

Accidentals are normally placed 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.

10.1.2 Accspacing

The **accspacing** directive must be followed by at least five numbers. They set the widths that are used for accidentals in time signatures and when outputting notes, in the order double sharp, flat, double flat, natural, and sharp. The default values are:

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

The values for sharp and flat also apply to half flats and half sharps in style 1 (☞ 11.6.2). The value for the narrower style 0 half sharps is by default set to the sharp value minus 0.2, but this can be overridden by supplying a sixth value.

The default widths are not the same as the widths that are set in the PMW music font, where the values are better suited for including the characters in text. It should not be necessary to change these widths unless a non-standard music font is being used.

10.1.3 B2PFfont

This directive is permitted only in the first movement. B2PF is a library for transforming strings of basic Unicode characters into their 'presentation forms', where the displayed letter shapes may vary, depending on their surroundings. PMW can optionally be compiled to make use of the B2PF library. This makes available the **b2pffont** directive, which has the following form:

```
b2pffont <fontword> <options> "<B2PF rules file>"
```

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 configured. Most commonly it will be one of the extra fonts. Any number of B2PF options may then follow, followed by one or more B2PF rules file names in quotes. If just one empty string is given, B2PF processing is initialized without any defined rules. This could be used to reverse the code points in a string.

The rules files are first sought in directories listed in the **-F** command line option, followed by B2PF's default directory. Here are some examples:

```
B2PFfont extra 1 "Arabic" "Arabic-LigExtra"
B2PFfont extra 2 input_backchars output_backchars "Arabic"
```

The options specify whether the input and output strings are in reading order or reverse reading order. The available options are `input_backchars`, `input_backcodes`, `output_backchars` and `output_backcodes`. See the B2PF documentation for more details.

10.1.4 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 fragments of pieces, or continuing a bar number sequence through several movements.

10.1.5 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.

10.1.6 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 for a piece with no bar lines, where 'invisible bar lines' can be used to tell PMW where systems can be ended, but no space must be inserted.

10.1.7 Barlinestyle

This directive specifies the way in which bar lines are to appear 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 output. These styles work only when the stave spacing is 32 points or greater (which is normally the case).

- Style 4 specifies a half-height bar line 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:
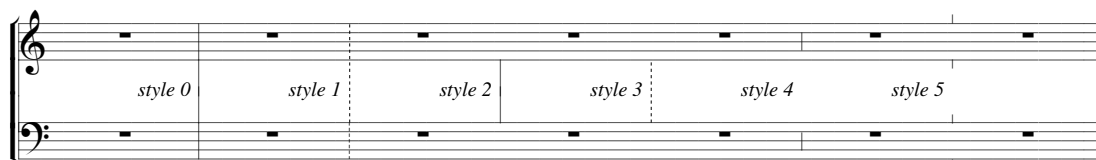
Note that the **breakbarlines** directive can be used to specify breaks in all bar lines at particular staves when style 0 or 1 is used, and an individual bar line can be broken by using **[breakbarline]**.

10.1.8 Barnumberlevel

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

```
barnumberlevel +4
```

This example puts all bar numbers four points higher up the page than they would otherwise have been. The default level is calculated using the size of the top stave of the system (excluding stave 0), but the adjustment value set by this directive is not scaled.

10.1.9 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 **[nocount]** (see section 6.2.2 for an example). The **barnumbers** directive completely ignores any bars that contain **[nocount]**. Automatic bar numbering can be overridden for individual bars, and uncounted bars may be numbered, by means of the stave directive **[barnumber]** (☞ 12.2.8). Several different automatic 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’, ‘roundboxed’, or ‘ringed’. These specify that bar numbers are to appear inside rectangular boxes (with either mitred or round corners), or roughly circular rings. If none of these words are given, the numbers appear without any special identification. The second argument must be present, and is either the word ‘line’ or a number. If ‘line’ is given, a bar number is output 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 requests a bar number, enclosed in a box, every 10 bars. The third argument is optional; it specifies the size of the font. 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). The default is roman.

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

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

10.1.10 Beamendrests

This directive, which has no arguments, causes PMW to include rests that are adjacent to beamed groups within the beams (☞ 11.7.3).

10.1.11 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.

10.1.12 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.

10.1.13 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 10.1.56, 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 vertical 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-movement 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.

10.1.14 Brace and Bracket

These two directives specify which staves 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 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 output for just one stave.

10.1.15 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**.

10.1.16 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.

10.1.17 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.

10.1.18 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.

10.1.19 Breverests

By default, PMW uses a semibreve rest sign for a complete bar's rest, whatever the time signature. This header 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.

10.1.20 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.

10.1.21 Check

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

10.1.22 Checkdoublebars

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

10.1.23 Clefsize

By default, new clefs that appear in the middle of lines of music are shown 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 0.7 times the normal size.

10.1.24 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. The layout of clefs is 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.

10.1.25 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 all the key signatures on a page do not line up vertically, a feature that is sometimes required. 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:

```
clefwidths 13 16 15 15 0
```

The values given may include fractions. 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.

10.1.26 Codemultirests

By default, when there are consecutive rest bars in all the staves that are selected, they are combined into a single bar and output as a 'long rest' sign with the number of bars above. This is the modern convention; in older music, when the number of bars was between two and eight, coded signs were used instead, as shown in this example:

If **codemultirests** is specified, PMW follows this older convention. Otherwise, multiple rests are all shown in the same way as the nine bars rest above. This directive can be cancelled in a subsequent movement by specifying **nocodemultirests**.

10.1.27 Copyzero

This directive makes it possible to have copies of stave zero 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 8.18.

10.1.28 Cuegracesize

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

10.1.29 Cuesize

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

10.1.30 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.

10.1.31 Doublenotes

This directive, which applies to the current movement only, causes the length of each note to be doubled throughout the movement. It also affects the current time signature and any subsequent time signature settings as follows:

- C and A are turned into 2*C and 2*A, that is, they are shown 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.

An entire movement can be formatted in different ways by adding or removing **doublenotes**, without any other changes. If you do not want time signatures to be affected, use the **[doublenotes]** stave directive instead (☞ 12.2.25). Note length adjustment is cumulative; for example, if **doublenotes** appears twice, notelengths are quadrupled. See also **halvenotes** (☞ 10.1.50).

10.1.32 Draw

The **draw** directive is used for defining simple drawn shapes. A full description of this facility is given in chapter 9.

10.1.33 Drawbarlines

This directive, which takes no arguments, is permitted only in the first movement. It has exactly the same effect as the **-drawbarlines** command line option, which is to cause bar lines to be output as drawing commands instead of using characters in the music font.

10.1.34 Drawstavelines

This directive is permitted only in the first movement. It has exactly the same effect as the **-drawstavelines** command line option, which is to cause staves to be output as drawing commands instead of using characters in the music font. Like the command line option, **drawstavelines** can be followed by an integer argument, which specifies the thickness of the stave lines, in units of one tenth of a point. If this is not given, the default thickness is 0.3 points.

10.1.35 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.

10.1.36 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.

10.1.37 Eps

This directive is an obsolete synonym for

```
output eps
```

It dates from before PDF output was available. Please see the **output** directive (☞ 10.1.97).

10.1.38 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 using 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.

10.1.39 Fbsize

By default, text that is specified as being part of a figured bass is output at the same default 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 (☞ 11.9).

10.1.40 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 for 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 placed 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.

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 output 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**.

10.1.41 Footnotesep

This directive specifies the amount of extra vertical white space to leave between multiple footnotes on the same page. The default is 4 points. See the **[footnote]** directive (☞ 12.2.34) for a full description of footnotes, which should not be confused with footings.

10.1.42 Footnotesize

This directive sets the default type size used for footnotes. If it is not present, the default size is 9 points. Individual footnotes may specify different sizes.

10.1.43 Gracesize

The default size of the music font used for 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.

10.1.44 Gracespacing

By default, a grace note is placed 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.

10.1.45 Gracestyle

When two or more staves are selected, 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 output 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]**), for some forms of plainsong music.

10.1.46 Hairpinlinewidth

This directive specifies the width of line used to draw crescendo and diminuendo 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 on high resolution printers.

10.1.47 Hairpinwidth

This directive specifies the default 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. A different width may be set for any individual hairpin (☞ 11.5.6).

10.1.48 Halfflatstyle

This directive selects which character to use 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 11.6.2.

10.1.49 Halfsharpstyle

This directive selects which character to use 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 11.6.2.

10.1.50 Halvenotes

This directive, which applies to the current movement only, causes the length of each note to be halved throughout the movement. It also affects the current time signature and any subsequent time signature settings 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.

An entire movement can be formatted in different ways by adding or removing **halvenotes** without any other changes. If you do not want time signatures to be affected, use the **[halvenotes]** stave directive instead (☞ 12.2.37). Note length adjustment is cumulative; for example, if **halvenotes** appears twice, notelengths are divided by four. See also **doublenotes** (☞ 10.1.31).

10.1.51 Heading

The **heading** directive defines a line of text as a heading for 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 output 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 output 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 output 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 (☞ 8.17); 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 directive is read. **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 headings of different sizes on different parts of the same line. It may also be negative; this can be used with an empty text string to make PMW start 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 outputs 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 9 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
  linelength 0 rlineto
  1 setlinewidth stroke
enddraw
```

```
heading "│Some Text" 0
heading draw rule 20
```

Unlike footings, where the first one is always at a fixed vertical position on the page, the first heading's vertical position may be affected by the vertical justification setting (☞ 10.1.56) and/or the top margin setting (☞ 10.1.13). See the **pageheading** directive for a means of setting up headings for pages other than the first.

10.1.52 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 11.12.5 for other facilities that can be used to control exactly what appears between underlaid syllables.

10.1.53 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 output, 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 output, 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.

10.1.54 IncPMWfont aka IncludePMWfont

This directive, which takes no arguments, is permitted only in the first movement. It has exactly the same effect as the **-incPMWfont** command line option, which is to include the PostScript Type 1 version of the PMW-Music font and/or the PMW-Alpha font in every PostScript output file that needs them. This directive is ignored when PDF output is selected.

10.1.55 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 output 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.

10.1.56 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** header 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 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 output 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 output 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 output 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 output 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 header 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 (☞ 3.5), 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 header 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.

10.1.57 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 header is a convenient shorthand. The single argument to the directive is a key signature in the format described in section 8.9.

```
key A$
key BM
```

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

10.1.58 Keysinglebar and keydoublebar

PMW puts 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.

10.1.59 Keytranspose

When transposing a piece, for each transposition amount, PMW needs to know two things: the new key and the number of letter changes for each note. For the standard keys this information is built in. For example, if you transpose the key of B up by a semitone, the result is C, and every note has its letter name increased by one, whereas if the transposition is down a semitone, the new key is B♭ with no change of letter.

For custom keys defined by the **makekey** directive (☞ 10.1.69) the transposition information must be explicitly supplied in one or more **keytranspose** directives. This directive is followed by the name of a key and then any number of triples that specify a transposition amount, a new key, and the number of letter changes. An equals sign and a slash are used as separators. For example:

```
keytranspose X1 2=X2/1 -1=X3/0
```

This example specifies that if the custom key X1 is transposed up by 2 semitones, the new key is X2 and each note's letter name is to be increased by 1 (with G wrapping round to A, of course); if the transposition is down by one semitone, the new key is X3, with no change of letter. Any number of triples may be present, or the information can be given in separate directives. If a transposition is specified more than once, the last value is used. A transposition of zero may be given. Transpositions whose absolute value is greater than 11 are adjusted to be in the range -11 to +11.

The **keytranspose** directive is not confined to custom keys. It can be used with standard keys in order to achieve special effects such as re-setting a part with a key signature using explicit accidentals and no key signature, as in this example:

```
key E
keytranspose E 0=N/0
transpose 0
```

A hard error occurs if a note cannot be transposed in the manner specified.

10.1.60 Keywarn

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

10.1.61 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.

10.1.62 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.

10.1.63 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** can be 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 you use **layout**, changes made by **notespacing** will not affect the layout.

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 (unlike lists of staves in directives such as **breakbarlines**). 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, each containing two systems; the second specifies one page containing four systems.

10.1.64 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.

10.1.65 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.

10.1.66 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 output is to occur, in absolute terms.

10.1.67 Longrestfont

The font used for the number that appears 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 is a 10-point roman font.

10.1.68 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 individual instrumental parts.

```
magnification 0.5
```

This example reduces the output to half size. Reduction is helpful for 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** or **layout** for this purpose.

10.1.69 Makekey

This directive is used to define a custom key signature. Its first argument is the name of one of the custom keys X1 to X10. This is followed by a definition of where accidentals exist for this key. Each accidental is specified by an accidental code followed immediately by a number in the range 0 to 9. The number specifies the position of the accidental for the treble clef. Zero is the bottom line of the stave, one is the first space, two is the next line, and so on. Optional spaces may appear between each definition. For example:

```
makekey X3 #2 $6
```

This example defines a (strange) key in which Gs are sharpened and Ds are flattened. The accidentals are output from left to right in the order of definition. Although the definition relates to the treble clef, custom keys can be used with any clef. For other clefs, the accidental positions are first raised or lowered as appropriate, then if any accidental ends up too high or too low, it is lowered or raised by an octave. If the result is not what is wanted, the **printkey** directive can be used to vary what is output for specific clefs.

Any accidental can be specified, including half sharps and flats (#- and $-) and even double accidentals or naturals. However, half accidentals are not supported in MIDI output. Custom keys can be redefined in multiple-movement pieces. They can be transposed, but only if you supply additional information in the form of one or more **keytranspose** directives (☞ 10.1.59). If you use a custom key without defining it, it behaves as C major.

The **makekey** directive was added to PMW long after **printkey** (☞ 10.1.105), which addresses some of the same issues. The differences are as follows:

- **printkey** affects only what is output for a specific key signature and clef. The key itself is unaffected and can be transposed.

- To use **printkey** you need to understand a bit about PMW's music font and how to 'program' character positioning.

- **makekey** constructs an actual key which can be used with any clef, and which affects MIDI output (though half accidentals are not supported by MIDI).

- Using **makekey** is simpler than **printkey** but you have to supply additional information if you want to transpose a custom key.

10.1.70 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:

```
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.

10.1.71 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**.

10.1.72 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 (see also **midivolume**). 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.

10.1.73 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 display. A percussion stave could therefore be set up thus:

```
midichannel 10 4 "cowbell"
[stave 4/1 "Cowbell" hclef 1]
b  r  b  r | ...
```

After the stave number, /1 defines a stave containing only one line. The notes are specified as Bs so that they appear 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 (☞ 12.2.67).

10.1.74 Midifornotesoff

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

10.1.75 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). The **midistart** setting is carried forward from the movement in which it is specified to all subsequent movements that do not have their own setting.

10.1.76 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 (☞ 8.3). 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).

10.1.77 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 shown on the stave. **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.

10.1.78 Miditremolo

When generating MIDI output, PMW recognises two kinds of tremolo: *Repeat* tremolos cause a single note to be repeated many times, and are notated in PMW using a note option, for example G\///\. *Trill* tremolos cause a repeated alternation between two notes, and are notated using the **[tremolo]** directive, for example G[tremolo]B.

Whereas repeat tremolos can sound quite reasonable on a MIDI keyboard instrument, they may sound weird on strings. The **miditremolo** directive can be used to set default tremolo handling on all staves. There is also a **[miditremolo]** stave directive for individual stave control. Each must be followed by one of the following words:

both	both kinds of tremolo are played
none	no tremolos are played
repeat	only repeat tremolos are played
trill	only trill tremolos are played

If your MIDI player supports the *tremolo strings* instrument, you could use a macro to change to it for just one note:

```
*define ST() [miditremolo none midivoice "tremolo strings"] &&&
 &&1 [miditremolo trill midivoice "strings 1"]
```

Note the use of three ampersands to continue the logical line onto the second input line. The macro could then be used like this: `&ST(A\///\)`. However, remember that changing a voice affects all staves that are using the same MIDI channel.

10.1.79 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, and **midichannel**, which can set a per-channel volume. The final volume used for any MIDI note is 127 multiplied by the channel volume and the stave volume and divided by 225.

10.1.80 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.

10.1.81 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.

10.1.82 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.

10.1.83 Nobeamendrests

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

10.1.84 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 (☞ 12.2.55).

10.1.85 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.

10.1.86 Nocodemultirests

This directive cancels the effect of **codemultirests** in a previous movement.

10.1.87 Nokerning

This directive is permitted only in the first movement. It disables the use of kerning for text strings (☞ 8.17.14).

10.1.88 Nokeywarn

By default, when there is a key signature change at the start of a new system, PMW also puts the new key signature at the end of the previous system, as is conventional in most music. The header 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.

10.1.89 Nosluroverwarnings

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

10.1.90 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.

10.1.91 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, semi-quavers, demi-semiquavers and hemi-demi-semiquavers respectively. The values in the default table are those of the following example:

```
notespacing 30 30 22 16 12 10 10 10
```

Internally, note spacings are held to an accuracy of 0.001 points. An alternative form of this directive 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 (☞ 12.2.60).

Another way of controlling the assignment of bars to systems is to use the **layout** header directive (☞ 10.1.63). If you are using **layout**, changes made by **notespacing** will not affect the layout.

10.1.92 Notime

This directive, which has no arguments, specifies that time signatures are not to be shown 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**.

10.1.93 Notimebase

This directive requests that only the 'numerator' (that is, the upper number) in time signatures be shown, in the middle of the stave. For example, in 3/4 time, only the 3 would appear. 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.

10.1.94 Notimewarn

By default, when there is a time signature change at the start of a new system, PMW also puts the new time signature at the end of the previous line, as is conventional in most music. The header 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.

10.1.95 Nounderlayextenders

This directive suppresses 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.

10.1.96 Nowidechars

This directive, which takes no arguments, is permitted only in the first movement. It has exactly the same effect as the **-nowidechars** command line option, which is to disable the use of very wide characters as part of stave lines. This option is ignored if **drawstavelines** is set.

10.1.97 Output

This directive is permitted only in the first movement. It must be followed by one of the words 'PDF', 'PostScript', 'PS', or 'EPS'. It sets the format of the output that PMW generates, and has the same effect as the **-pdf**, **-ps**, or **-eps** command line options. An error occurs if conflicting settings are given.

10.1.98 Overlaydepth

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

10.1.99 Overlaysize

By default, text that is specified as being vocal overlay is output 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 specified at different sizes by using the /s or /S text qualifier. The overlay size and the underlay size are separate parameters.

10.1.100 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.

10.1.101 Pagefooting

The **pagefooting** directive defines text for 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.

10.1.102 Pageheading

The **pageheading** directive defines text for 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 8.1.7 and also the **[newmovement]** directive for discussions of headings when there is more than one movement in a file.

10.1.103 Pagelength

This directive is permitted only in the first movement. See section 10.1.66 (*Linelength and pagelength*) above.

10.1.104 PMWversion

This directive checks the version of PMW is in use. It must be followed by a version number or a version number preceded by > (greater than), >= (greater than or equal), < (less than), <= (less than or equal), = or == (equal, same as no sign):

```
pmwversion    4.50
pmwversion >= 5.00
pmwversion <  5.00
```

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

10.1.105 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. Another way of doing this (a more recent addition to PMW) is to use the **makekey** directive (☞ 10.1.69) to define a custom key. The syntax of **printkey** is as follows:

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

There may be many occurrences of **printkey** in a single input file. The directive applies to the movement in which it occurs, and any subsequent movements, unless overridden by a subsequent **printkey** directive with the same key and clef. The most recent occurrence is always the one that is

used. The use of **printkey** affects only what is output for the given key/clef combination. It has no other effect on PMW's behaviour.

The first string specifies what is to appear instead of the normal key signature. The second string is optional, and specifies what is to appear for a 'cancellation' key signature, that is, when there is a change to a key with an empty signature. If not supplied, this string is set empty.

The string is normally a sequence of characters in the music font, which is set as the default font at the start of the string. 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\" "\40\"
printkey E$ treble "x~\39\x~\191\ww\191\"
printkey E$ bass   "~\39\x~\191\ww\191\"
[stave 1 treble 1]
GG | [key C] GG | [key E$] GG [bass 0] | [key E$] GG ||
[endstave]
```
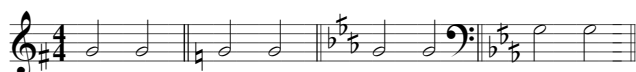
In the above example, the G major key signature is displayed as a single sharp (character 37 in the music font), in the initial vertical position, with a natural (character 40) in the same position for cancellation. For the E-flat treble clef signature, characters 120 (x) and 126 (~) are used to move up four and two points, respectively, so that the flat (character 39) is 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 16.

10.1.106 Printtime

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

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

There may be many occurrences of **printtime** in a single input file. The directive applies to the movement in which it occurs, and any subsequent movements, unless overridden by a subsequent **printtime** directive for the same time signature. The most recent occurrence is always the one that is used. Whenever the given time signature is to be output, the **printtime** setting is used instead. If the second string is empty, the first is output vertically centred on the stave; otherwise the two strings are output on the third and first stave lines, respectively, with their horizontal centres aligning. Some examples of possible uses are:

```
printtime 8/8  "3+3+2" "8"
printtime 12/8 "3 2"   "2 2"
printtime 3/4  "3"/s2  ""
printtime 3/2 "\**147\" ""
```

Setting both strings empty suppresses all output for the time signature. The default font at the start of each string is the font specified by the **timefont** directive for the current movement (as long as it precedes **printtime**) or for an earlier movement. 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 for **printtime** is that specified by **timefont**, with 11.8 points as the ultimate default. However, as shown in the third example above, you can follow each text string with /s or /S and a number, to specify a

particular size for a given string (☞ 11.9.3). For s, the number refers to the list of text sizes specified by the **textsizes** directive, which in this example has set size 2 to 20 points. For /S, the number refers to one of the fixed sizes. The fourth example uses character 147 from the music font (requested by the asterisks) to output an old-fashioned signature in the form of a circle.

10.1.107 Rehearsalmarks

This directive controls the way rehearsal marks (☞ 11.11) are shown. It has this syntax:

```
rehearsalmarks <align> <style> <size> <fontname>
```

All the arguments are optional, except that if *<fontname>* is present, one of either *<style>* or *<size>* (or both) must precede it. Parameters that are unspecified are left unchanged. The settings are copied to any following movement, but can be changed therein.

The *<align>* argument affects the positioning of rehearsal marks that occur at the start of the first bar of a system (that is, at the start of a line). By default, such marks are aligned with the first note in the bar. If 'linestartleft' is present, these marks are instead placed at the start of the line; 'nolinestartleft' can be used to restore the default in a following movement. If *<style>* is present, it must be one of the words 'boxed' or 'roundboxed' (enclose in a rectangular box, in the second case with rounded corners), 'ringed' (enclose in a ring), or 'plain' (do not enclose). The size is the font size, and the final argument specifies the font to be used (☞ 8.13).

```
rehearsalmarks boxed 13
rehearsalmarks ringed italic
rehearsalmarks 11 bolditalic
rehearsalmarks linestartleft plain 14 bold
```

By default, rehearsal marks are enclosed in a box with mitred corners, using a 12-point roman font.

10.1.108 Repeatbarfont

The font used for 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. If the **[1st]** or **[2nd]** directive specifies a text string to replace the number (☞ 12.2.1), this font is the default, but can be changed within the string.

10.1.109 Repeatstyle

This directive specifies how repeat marks appear. The default is the conventional combination of two dots with a thin and a thick vertical line. In its basic form 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 appear 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.

style 0   style 1   style 2   style 3   style 4

When outputting single parts it is common to see repeat marks with small 'wings' above them, to make them more prominent. You can request these by adding 10 to the argument of **repeatstyle**. Thus, a value of 10 generates repeats like this:

You can use conditional pre-processing directives (☞ 8.2.7) to arrange for wings to appear only in parts, and not in a score. If there is more than one stave in a system, wings are added to the top and bottom staves only.

10.1.110 Righttoleft

The **righttoleft** directive causes PMW to typeset 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, clefs, and other markings are not altered, except for the horizontal wedge accent \>\, which is flipped to point the other way.

Right-to-left output is implemented by transforming the 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 are still output 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 output 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 put; 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.

If PMW has been compiled with B2PF support, you can use the **b2pffont** directive to cause strings in certain fonts to be processed by the B2PF library, which can convert base characters to presentation forms, and can also reverse the order of characters in a string.

When right-to-left output 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 the pages on the sheets is reversed. This makes it possible to generate correct right-to-left pamphlet-style pages for folding and binding.

10.1.111 Selectstave(s)

This directive is used to specify a selection of staves to be output. 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.

10.1.112 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 vertical and horizontal paper 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**.

10.1.113 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).

10.1.114 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.

10.1.115 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. **Note:** This applies only to non-music fonts. If `\sc\` is used with a music font, it has the same effect as using `\mu\`, that is, it switches to a music font at 0.9 nominal size.

10.1.116 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 also applied at the actual start of the system; by default nothing appears there. See section 8.12 for an example of the use of **startbracketbar**.

10.1.117 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.

10.1.118 Startnotime

This directive, which has no arguments, applies only to the movement in which it appears. It causes no time signature to appear at the start of the movement, but does not suppress subsequent time signature changes. This is useful for typesetting parts of pieces. The **notime** directive suppresses all time signatures in a piece.

10.1.119 Stavesize(s)

This directive specifies different sizes for certain staves. 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 0.8 times the normal size, and the second specifies that staves 4–6 are 1.2 times the normal size. A change in the relative size of a stave affects everything on that stave, both notes and text items. However, the text that appears to the left of the stave (the instrument or voice 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 output on this stave, only text items are affected. Bar lines are 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.

10.1.120 Stavespacing

This directive controls the amount of vertical white space between staves. Whatever is set for one movement carries over as the default for the next movement. The distance between staves is measured from the bottom of one stave to the bottom of the next. The initial default is 44 points. If the **stavespacing** directive is followed by a single 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).

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 an 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 **[ssabove]**, **[sshere]**, and **[ssnext]** (☞ 12.2.89).

A value of zero may be given for the spacing. This causes two successive staves to appear on top of each other, and can be useful for setting two lines of music on the same stave. It can also be useful for 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]** or **[backup]** stave directives may be more convenient than the use of a complete overprinted stave.

10.1.121 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, up to a maximum shortening of 8 points. 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. Note that these values do not apply to beamed notes; however, the stem lengths of individual notes in a beam can be adjusted (☞ 11.6.17, 11.6.23).

10.1.122 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 11.8 (*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.

10.1.123 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. Like **stavespacing**, if **stemswaplevel** is followed by a single number, this sets the level for all staves to that value. After such a single number (which is optional), further pairs of numbers separated by slashes can be given to set different levels for individual staves. The swap level value 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. For all other staves the value is unchanged.

10.1.124 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 (☞ 12.2.98).

10.1.125 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. Whatever is set for one movement carries over as the default for the next movement. The initial 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 (☞ 10.1.56), 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]**, **[sgnext]**, and **[sgabove]** in section 12.2.74.

10.1.126 Systemseparator

This directive configures separating marks between systems, also known as system dividers, as shown in the following example:

The directive has up to five arguments. Only the first is mandatory, and it specifies the length of the separator lines. Setting it to zero (the default) turns off separators. A value of 16 is used in the example above. The complete set of arguments is:

```
systemseparator <length> <width> <angle> <x-adjust> <y-adjust>
```

The default width is 2 and the default angle is 20 (degrees). The lines are by default positioned with their lefthand end aligned with the start of the system, at a fixed position above it. The last two arguments, which may be positive or negative, adjust this position.

```
systemseparator 20 2.1 25 -16 -2
```

This example specifes slightly longer, thicker, and more angled lines, moved leftwards and downwards. Separator lines are not output before the first system on a page, the first system of a new movement if it has any heading lines, or any system that contains only one stave.

10.1.127 Textfont

This directive is permitted only in the first movement. By default, all text strings use 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 final 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 `\xx1\`. See section 8.17.9 for details of font-changing escape sequences. The capitalization of font names is important.

The **textfont** directive has a third, optional argument that can be given immediately before the font name. It is the word 'include', for example:

```
textfont extra 1 include "FancyFont"
```

If 'include' is present, PMW searches for a font file to include in PostScript output. The file name must be the font name with the extension `.pfa`, and the file must be in one of the directories that is searched for font files (see the **-F** command line option). Note that the **-incPMWfont** command line option and the **incPMWfont** directive apply only to PMW's music fonts. When output is in PDF format, 'include' has no effect because non-standard fonts are always included in the output.

10.1.128 Textsizes

Text that is specified with music on a stave can be output in twenty different settable 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 twenty 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 output using the first size specified, but underlay, overlay, and figured bass text uses the size specified by the **underlaysize**, **overlaysize**, or **fbsize** header directives, respectively. To output text at any of the other settable sizes, the /s qualifier must be used. There are also 10 fixed sizes that can be selected by the /S qualifier (☞ 11.9.3).

10.1.129 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.

10.1.130 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.

10.1.131 Timebase

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

10.1.132 Timefont

The **timefont** directive is used to specify the font for time signatures. Its syntax is:

```
timefont <size> <name>
```

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 time signatures. The parameters set by **timefont** do not affect the time signatures C and A – they affect only numeric time signatures, or those output 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.

10.1.133 Timewarn

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

10.1.134 Topmargin

See section 10.1.13 (*Bottommargin and topmargin*) above.

10.1.135 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 semitones of transposition up or down, respectively. If a transposition is also specified from the command line, the two values are added together. Section 8.10 gives more details about transposition.

10.1.136 Transposedacc

By default, unless the stave has the pseudo-key N set, PMW always outputs an accidental on a transposed note if an accidental is present on the original, thereby preserving cautionary accidentals. If **transposedacc** is followed by the word 'noforce', it changes this behaviour such that accidentals are shown only when strictly necessary. The standard behaviour can be reinstated for subsequent movements by specifying 'force'. There is also a **[transposedacc]** stave directive that can alter the behaviour within a single stave. It is possible to force either behaviour for individual notes (☞ 11.6.7). For the special treatment of the pseudo-key N, see section 8.10.1.

10.1.137 Transposedkey

When there is a choice of standard 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 8.10. 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 process 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 format the transposed music either with a modern key signature, or using the same convention.

10.1.138 Trillstring

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

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

This example replaces *tr* by the letters *tr* in italic. The string may be preceded by a number, specifying the size of font to be used. The default size is 10 points.

10.1.139 Tripletfont

This directive is a synonym for **tupletfont**.

10.1.140 Tripletlinewidth

This directive is a synonym for **tupletlinewidth**.

10.1.141 Tupletfont

This directive specifies the size and style of the text font used for the '3' over triplets, and also similar numbers over other irregular note groups (tuplets). The syntax is:

```
tupletfont <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' (☞ 8.13). It cannot be omitted. When this directive is not used, a 10-point roman font is used for tuplet numbers.

10.1.142 Tupletlinewidth

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

10.1.143 Underlaydepth

If two or more character strings, all designated as underlay, are attached to the same note, they are automatically placed 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 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.

10.1.144 Underlayextenders

This directive restores extender lines at the ends of underlay words whose last syllable extends over more than one note if they were suppressed by **nounderlayextenders** in an earlier movement.

10.1.145 Underlaysize

By default, text that is specified as being vocal underlay is output 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 output at different sizes by using the /s or /S text qualifier. The size parameters for underlaid and overlaid text are separate and independent.

10.1.146 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 (☞ 6.3.3), 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.

10.1.147 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.

10.1.148 Vertaccsize

The size of accidentals that are placed above or below notes (☞ 11.6.6) is controlled by this header 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 slightly smaller than the default.

11. Stave data

This is the first of two chapters that 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 11.9.

- 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 12.2.

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 except to avoid ambiguity . PMW makes no attempt to check on the musical sense of what it is asked to typeset, 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, though it is possible to handle some special cases such as a 2/4 stave and a 6/8 stave in the same system (☞ 12.2.106). The length of the notes in a bar must agree with the time signature, unless **nocheck** or **[nocheck]** has been used (☞ 6.2).

11.1 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 without any intervening space characters gives a double bar line. Unless the final bar ends with a double bar line or the **unfinished** directive (☞ 10.1.147) is 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 such a bar line may be needed in the middle of a piece. This is notated by three vertical bars in succession. To encode a totally empty bar it is necessary to include at least one space between two vertical bar characters. The amount of horizontal space that is inserted after a bar line is controlled by the **barlinespace** directive.

There are six different styles for single barlines (☞ 10.1.7). 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 that the **breakbarlines** directive can be used to specify breaks in bar lines at particular staves. This overall setting can be overridden for individual bars by using the **[breakbarline]** or **[unbreakbarline]** stave directives.

Normally, the end of a bar marks the end of a set of beamed notes. If a bar line encoding is followed by an equals sign, any existing beam is carried over the bar line and into the next bar if it starts with a suitable note (☞ 11.7.2). This applies all forms of bar line, including invisible bar lines.

11.1.1 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 output. 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 shown, the **[nocount]** stave directive should be used to stop one of them from being counted.

11.1.2 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 appear 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.

11.2 Repeated bars or part bars

A bar containing the conventional 'repeat previous bar' sign ✗ can be coded using a special form of whole bar rest (☞11.6.8). If you want to duplicate a sequence of bars, or a fragment of a bar, you can arrange this via the string repetition feature (☞ 8.2.5), which replicates an arbitrary sequence of input characters at preprocessing time.

For repeating a single, complete bar there is a simpler shortcut mechanism. The appearance of a number enclosed in square brackets causes those items to the right of it in the current 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 repetition, the key signature is included in the first bar only. If it had followed `[10]` it would have been included in all ten bars. Within any string that is part of a bar that is repeated in this way, the escape sequence `\r\` is replaced by the repetition number. Thus, to number a sequence of rest bars, centring the number over the rest sign, coding like this can be used:

```
[10] "\r\"/a/c R! |
```

You can also use `\r2\`, which does the same thing, but starting at the second bar; nothing is inserted in the first repeated bar. This makes it possible to put something different on the first bar. Note the use of `/cb` in the next example to centre each text string in its bar. See section 11.9 for details of string options.

```
"1–6"/a/cb [6] "\r2\"/a/cb f–x3 |
```
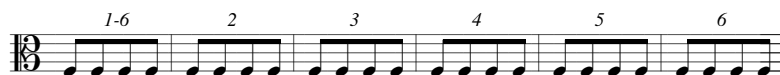

There is danger of confusion between repeated bars and rehearsal marks. Accidental omission of the quotes from a numerical rehearsal mark such as `["42"]` can lead to some very strange effects. There is more discussion of repeated rest bars in the section entitled *Repeated rest bars* (☞ 11.6.15).

Warning: This bar-repeating mechanism should not be used with multi-syllable underlay texts, because the syllables are apportioned to notes as they are read from the input, and bars that are repeated in this way are not re-read. The string repetition facility, however, can be used in this situation.

11.3 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 styles of repeat mark are provided (☞ 10.1.109). First and second time bars are catered for (☞ 12.2.1). 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' (☞ 11.1.1). 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 shown depends on the order in which they appear in the input.

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

This example causes the time signature to be 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 first on all staves.

11.4 Caesuras

A caesura (pause) in the music is encoded in the input in very much the way it appears on the stave.

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

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

11.5 Hairpins

The characters > and < are used within a stave to encode hairpins (crescendo and diminuendo marks, sometimes called 'wedges'). 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 diminuendo or the start of the continuation of a crescendo is drawn with a small gap to indicate the continuation.

11.5.1 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 enclosed note, but there are options to 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.

Another way of specifying horizontal position is to use /lc or /rc followed by a number that may have a fractional part. This specifies a left or right movement by a number of crotchets, thereby positioning the hairpin with respect to the musical offset in the bar. This is similar to the 'offset' positioning that is used in MusicXML. An offset value is used only if neither /h nor /bar is present.

11.5.2 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, /bar, /lc, or /rc, the effect is cumulative.

11.5.3 Vertical hairpin positioning

Hairpins are positioned under the stave by default, but the **[hairpins]** directive (☞ 12.2.35) can be used to place them 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 positions the hairpins with their 'sharp end' 10 points above the top of the stave and twelve points below the bottom of the stave, respectively.

11.5.4 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 outputs 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.

11.5.5 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.

11.5.6 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** header directive (for the whole piece) or by the **[hairpinwidth]** directive (for the current stave). There is also a **hairpinlinewidth** directive, which is used to change the thickness of the lines used for drawing hairpins. The default thickness is 0.2 points.

11.6 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.

11.6.1 Note pitch

The pitch of a note is indicated by one of the usual note-letters, A to G. As is conventional in music, by default the letters represent the notes C to B in the octave below middle C (octave 0). 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 (☞ 11.6.9).

The default octave for notes can be changed by means of the **[octave]** stave directive (☞ 12.2.61) or when a clef is specified (☞ 12.1). Each octave setting replaces the previous one; they are not cumulative. The setting can a positive or a negative number:

```
[octave -1]      C is the note two octaves below middle C
[octave 0]       C is the note one octave below middle C
[octave 1]       C is middle C
```

PMW supports 8 octaves, from -3 to +4. A note's pitch is raised one octave above the default 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, when the octave is set at 0 (the default):

```
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
```

11.6.2 Half accidentals

Two different symbols for half sharps and half flats are provided in the PMW-Music font; the **halfsharpstyle** and **halfflatstyle** directives specify which ones to use. A half sharp is notated as `#-` and a half flat as `$-`.

Style 0   Style 1

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. Custom key signatures containing half accidentals are supported via the **makekey** directive (☞ 10.1.69).

11.6.3 Bracketted and parenthesized accidentals

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

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

11.6.4 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 visible 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 displayed. 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.

11.6.5 Moved accidentals

Occasionally it is necessary to move an accidental sign to the left of where it would normally be. If the character < follows the accidental, it is placed 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
```

11.6.6 Accidentals above and below notes

Some editors like to place 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 placed over or under the note (the letters `a` and `b` cannot be used because they are note names).

```
#oa $ub
```

This example puts 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 header 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 (☞ 11.6.3) are required above or below notes, the bracket must follow `o` or `u` and any up/down movement specification.

11.6.7 Transposed accidentals

Normally, unless the stave is using the pseudo-key N, PMW outputs an accidental sign for a transposed note if there is an accidental in the input, thus preserving cautionary accidentals. The special handling for the pseudo-key N is described in section 8.10.1. For other keys, 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 position the accidental above or below the note.

PMW can be forced to choose an 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 input accidental with one of the following character sequences:

`^#`	use a sharp ('black' notes and C and F natural)
`^$`	use a flat ('black' notes and B and E natural)
`^##`	use a double sharp ('white' notes except C and F)
`^$$`	use a double flat ('white' notes except B and E)
`^%`	use 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 shown as A-natural, it will now appear as G-double-sharp.

11.6.8 Rests

There are four 'note letters' that are used instead of pitch letters to specify rests of various kinds. For ease of remembering, they are adjacent in the alphabet. 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 specifies a rest in the same way as R, but it causes nothing at all to be output. It is an 'invisible rest' that is useful for special effects when overprinting staves or using coupled staves.

The letter S behaves like 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 appearance of 'long rest' bars where possible (☞ 11.6.15). When a rest bar is specified using S instead of R, it is always output as an individual bar and never amalgamated. You can think of S as standing for 'single' or 'separate'.

The letter T behaves like R except when it is used for a crotchet rest or a whole bar's rest. In these cases it is a pseudo rest that is used as a way of coding two special features. When T specifies a crotchet, instead of a rest sign, a thick slash ╱ is output. This mark is used to indicate a beat without specifying a pitch (often because the part specifies chords by means of chord symbols). When T is followed by an exclamation mark to indicate a whole bar rest, the bar is never amalgamated with adjacent bars, and instead of a rest sign, the conventional 'repeat previous bar' sign ╱. is output. However, no sound is generated for MIDI output.

11.6.9 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. The primary length is changed by the addition of +, −, and = characters, which follow any quotes or grave accents that adjust the note's octave, for example: `C'+`. These characters can be freely mixed on any note, but + normally follows an upper case letter whereas − and = normally follow a lower case letter. The only restriction is the maximum and minimum supported note length (breve and hemi-demi-semiquaver).

A plus character doubles the note's length. If used after an upper case letter it gives a semibreve (one plus) or a breve (two plusses). A hyphen halves the note length and an equals sign divides it by four. If used after a lower case letter, the number of horizontal lines in these characters is equal to the number of 'flags' that are attached to the stem of the note. 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.

Some music contains long runs of quavers or semiquavers, which requires typing minus or equals signs after every note letter by default. This can be avoided by making use of the **[halvenotes]** stave directive (☞ 12.2.37). Notes that follow this directive have their length halved, so upper case letters

are now crotchets and lower case letters are quavers. The effect is cumulative, so using **[halvenotes]** twice turns lower case letters into semiquavers. The effect can be reversed with **[doublenotes]** (☞ 12.2.25). For example, these three bars contain exactly the same notes:

```
a-b-c-d-; a-b-c-d- |
[halvenotes] abcd; abcd |
[doublenotes] a-b-c-d-; a-b-c-d- |
```

There are also **halvenotes** and **doublenotes** header directives (☞ 10.1.50, 10.1.31), which cause every stave to begin with halved or doubled notes, but these directives also affect time signatures and are more useful for adjusting the overall format of a piece than for minimizing typing.

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 shown 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 outputs 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 without any automatic halving or doubling:

```
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
```

11.6.10 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 or ornaments on the chord (staccato, etc.), these must be specified on the first note. Other note options that apply just to one note (for example, notehead size or shape) can be on any note. A stem length adjustment is ignored unless it is on the note nearest to the stem. Chords consisting of quavers or shorter notes are beamed in the usual way (☞ 11.7); a semicolon after the closing parenthesis breaks all the beaming, whereas a comma breaks secondary beams only. If the chord is tied (☞ 11.6.27), the underline character that indicates a tie must appear after the closing parenthesis and before any beam break character. 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 (☞ 11.6.5). In this case, no automatic positioning is done; it is assumed that the user has positioned all the accidentals in the chord by hand.

11.6.11 Horizontal movement of augmentation dots

It is occasionally necessary to move augmentation dots to the right, usually for 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.

11.6.12 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 (☞ 11.6.17). This option affects only notes that lie on stave lines. Normally dots for such notes are placed in the stave space above, but if the colon option is present, they are placed 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 placed 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 is shown 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.

11.6.13 Notehead shapes and sizes

The shape of noteheads is controlled by the **[noteheads]** directive (☞ 12.2.58). 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 output 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 (☞ 11.6.17). 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.

Another way of indicating optional notes is to enclose the notehead in round brackets (parentheses). This can be requested for individual notes by means of the \)\ note option (☞ 11.6.17). However, this does not work for noteheads in a chord that are on the 'wrong' side of the stem.

11.6.14 Whole bar rests

There is one other special character that may follow the rest letters R, Q, S, or T. 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 outputs a conventional whole bar rest sign, whereas the latter causes nothing at all to be output 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. T! is a special notation for the conventional 'repeat last bar' sign (☞ 11.6.8.

Whole bar rests specified using an exclamation mark are normally output 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** header directive or by using the masquerade facility (☞ 11.6.24).

Rests that happen to fill the bar, but which are not specified with exclamation marks, are output as rests of the appropriate length. For example, in 3/4 time the rest R. is output as a dotted minim rest. If bar lengths are being checked, such a rest is positioned centred in the bar, but if they are not, it is put 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 (☞ 11.6.17).

11.6.15 Combining a sequence of rest bars

When each bar in a sequence of bars contains only a single whole-bar rest specified using R or Q (but not S or T) they are amalgamated into a single bar. If the first bar's rest used Q, nothing is output in the bar. Otherwise, a conventional 'long rest' sign is output, with the number of bars above. Of course, this happens only if all the staves in the system have rest bars, typically when one or more parts are being extracted from a score. By default, any number of repeated rest bars are shown in the same way, using the ▬ sign, which is the modern convention. Older music sometimes used special code signs when the number of bars was between two and eight; see the **codemultirests** header directive if you want to use this style (☞ 10.1.26).

A rest bar is considered eligible for amalgamation with its neighbours if it contains nothing but an unadorned rest item. A rest bar with a fermata on the rest (for example) is always treated 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 or a clef. 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 output as a single multi-bar rest when this stave is the only one selected. As it stands, the G.P. bar is output on its own, followed by an 8-bar multiple rest.

When a multi-rest bar is stretched sufficiently, the long rest sign is also stretched. This provides a way of generating a single multi-rest bar across the whole page for a *tacet* movement in an instrumental part. For example:

```
[stave 1 treble 1] [space 200][40]R! [endstave]
```

If left and right justification is in effect (☞ 10.1.56), all the **[space]** directive has to do is to widen the bar sufficiently for the justification to kick in. It must be given before the repeat count, to ensure that the remaining bars contain nothing other than the rest.

If a widened repeated rest is not at the start of a line, and there is something at the start of the first bar, the situation is a bit more complicated. Consider:

```
[stave 1 treble 1] EF | GG | [key E$ space 200][30]R! [endstave]
```

Whether the **[space]** directive appears before or after the **[key]** directive makes no difference, because what **[space]** affects is the position of the following note or rest. A mid-line key signature is always positioned relative to the following note/rest position. In this example, the effect of **[space]** is to move both the key signature and long rest sign well to the right. The (messy) solution is to use **[move]** directives to adjust their positions:

```
[stave 1 treble 1] EF | GG |
[space 200 move -200 key E$ move -100][30]R!
[endstave]
```

Notice that the second **[move]** specifies half the distance of the first. The **[space]** directive can be used with a small negative argument to make a repeated rest bar a bit narrower than the default. Reducing it by up to 20 points is usually reasonable.

11.6.16 Notes that fill a bar

Sometimes there is an unorthodox requirement to specify that a note fills a bar, independent of its conventional length. Here is an example:

This effect can be achieved by following a note letter with an exclamation mark, exactly as for a whole bar rest. The second stave in the example above is notated like this:

```
[stave 2 bass 0] G!_ | G! | [endstave]
```

By default, a semibreve notehead is output, but this can be changed by using the masquerade facility (☞ 11.6.24).

11.6.17 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
\!\	put accent on stem side, trill or fermata below (☞ 11.6.19)
\.\	staccato dot
\..\	staccatissimo mark
\:\	invert augmentation dot position (notes on lines, ☞ 11.6.12)
\::\	move augmentation dot up (notes in spaces, ☞ 11.6.12)
\-\	solid line emphasis mark
\>\	horizontal wedge emphasis mark
\'\	'start of bar' accent
\)\	notehead in round brackets
\a<n>\	accent number <n> (☞ 11.6.18)
\ar\	arpeggio mark
\ard\	arpeggio mark with downward arrow
\aru\	arpeggio mark with upward arrow
\c\	put on coupled stave (☞ 6.9.4)
\C\	flip centring 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 put on coupled stave (☞ 6.9.4)
`\m\`	masquerade note (☞ 11.6.24)
`\nc\`	circular notehead
`\nd\`	'direct' notehead
`\nh\`	harmonic (diamond) notehead
`\nn\`	normal notehead
`\no\`	notehead only, no stem
`\nx\`	cross notehead
`\nz\`	no notehead
`\o\`	small circle over note (harmonic)
`\rt\`	inverted turn (reversed)
`\rt|\`	reversed turn with bar
`\sd\`	force note stem down
`\sl<n>\`	lengthen stem by <n> points (☞ 11.6.23)
`\sl-<n>\`	shorten stem by <n> points (☞ 11.6.23)
`\sm\`	use small (cue sized) notehead (☞ 11.6.13)
`\sp\`	spread chord
`\su\`	force note stem up
`\sw\`	swap note stem direction in beam (☞ 11.7.5)
`\t\`	turn
`\t|\`	inverted turn (with bar)
`\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 (☞ 11.6.22)

Here are some examples (there is further discussion below):

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.

There are two conventions for notating inverse turns: either a normal turn sign with a vertical bar through it, or a reversed turn sign. PMW caters for both, and for completeness also has an reversed turn sign with a bar.

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 (☞ 11.6.12).

When there is a whole bar rest in some staves, and just a single note in the others, 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. On any other note or rest `\C\` is ignored. In fact, `\C\` flips the centring state of the note or rest, so `R!\C\` can be used for a whole bar rest that is *not* centred.

The options that start with the letter n adjust the notehead setting for an individual note. The default is set by the **[noteheads]** stave directive, whose description contains the full details (☞ 12.2.58). In a chord, these options apply only to the note on which they appear, so it is possible to have a mixture of noteheads in a chord.

11.6.18 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  ʌ
 6  string down bow  ⊓
 7  string up bow  ∨
 8  ring (harmonic)  ∘
 9  'start of bar' accent  ∣
10  staccatissimo mark  ▾
```

11.6.19 Position of accents and ornaments

By default, accents and the harmonic ring are positioned opposite a note's stem. That is, they are below the note if the stem is up, and above the note if the stem is down. (A notional stem direction is computed for breves and semibreves.) Fermatas, trill signs, and other ornaments are by default placed above the note, independent of the stem direction.

If an accent or harmonic ring mark is followed by an exclamation mark, PMW positions it on the same side of the notehead as the stem, which is occasionally necessary when more than one part is being output on the same stave. If ! is used with a fermata or trill or other ornament, the sign is placed below instead of above the note. String bowing marks are not affected by the use of the ! option. They are put above the stave unless the **[bowing]** directive has specified otherwise.

Three accents (staccato, staccatissimo, emphasis bar) and the harmonic circle are positioned within the stave for notes of an appropriate pitch. Also, they are placed inside ties and short slurs (☞ 11.6.27) when they are both opposite the stem. The other accents and ornaments are always placed outside the stave and beyond ties and short slurs by default. However, the position of all accents and ornaments can be adjusted, as described in the next section.

11.6.20 Moving accents and ornaments

It is possible to move all accents and ornaments up and down, or, except for tremolos, 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.

Tremolo markings themselves can be moved up and down, but not left or right. The notation looks confusing, but is consistent: for example, `G\//u4\` specifies a single tremolo that is moved up by 4 points.

11.6.21 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\  |
```


11.6.22 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 output 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 any of them can be put back again afterwards.

11.6.23 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 of 8 points, beyond which shortening is ignored. The **shortenstems** header 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.

The sl notation for shortening or lengthening the stem of a note works for beamed as well as unbeamed notes, and is an alternative to the **[beammove]** directive for adjusting the vertical position of beams. (Note that the **stemlengths** setting does not apply to beamed notes.) Lengthening the stem of any note in a beamed group pushes the beam further away from the noteheads. However, to move a beam nearer to the noteheads you have to shorten the stems of all the notes in the beam, which is messier than using **[beammove]**. Consider this example:

```
[stems up]
g`-c-e-g- |
g`-c-e-\sl5\g- |
g`-\sl-4\c-e-g- |
g`-\sl-4\c-\sl-4\e-\sl-4\g-\sl-4\ |
[beammove -4] g`-c-e-g- |
```


In the first bar there are no stem length adjustments. In the second bar, the third note has its stem lengthened by 5 points, causing the other notes' stems also to be lengthened to make them join the beam. In the third bar, one note's stem is shortened, but this has no effect because the other notes still have full-length stems. In the fourth bar, all the notes' stems are shortened. In the fifth bar, the same effect is achieved using **[beammove]**.

11.6.24 Masquerading notes and rests

For special effects (for example, tremolos between notes – see **[tremolo]**) it is sometimes desirable to use 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. This kind of *masquerading* 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. For a chord, the masquerade indication must appear on the first note.

The only effect of masquerading is to substitute a different note for display; the position of the note is not affected. Masquerading does not disable beaming. When a masquerade is requested, an augmentation dot can be requested with it, and if it is not, no dot is output, even if the original note is augmented. The ability to add augmentation dots makes it easier to set 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++\     outputs a breve instead of a semibreve
g-\m\       outputs a crotchet instead of a quaver
g.\M\       outputs 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 shown 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.

11.6.25 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 outputting these marks on their own at positions in a bar that are not associated with visible notes.

11.6.26 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 number of points.

```
R\l4\
```

This example outputs a minim rest on the fourth instead of the third line. If rests are generally to be appear 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]** or **[move]**, the effect is cumulative.

11.6.27 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 put 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.

11.6.28 Editorial and intermittent ties

Ties can be marked editorial, or drawn 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

11.6.29 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.

11.6.30 Glissando marks

Glissando marks are available for single notes but not for chords. 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 6.5 and 11.9.4.

11.6.31 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.
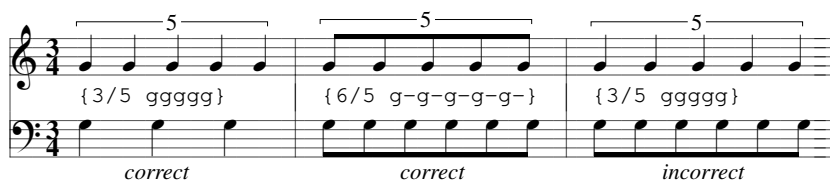
In PMW input, brace characters (curly brackets) are used to enclose a group of notes that is not a standard division of a longer note or group. In simple cases, the opening brace is followed by a single number to indicate the number of notes in the irregular group. If this number is omitted, the group 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 groups on a second stave to show how the irregular groups are interpreted.

11.6.32 Nested irregular note groups

From release 5.33, PMW supports the nesting of irregular note groups, for example:

```
g b {/a e'c' {c'-c'-c'-} }
```


11.6.33 Options for irregular note groups

By default, PMW puts the number of an irregular note group (for example, the '3' for a triplet) on the same side of the noteheads as the stems. The **tupletfont** directive is used to specify the size and type of font used. If the notes are beamed, just the number is output; if not, a horizontal 'bracket' is drawn 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 **[tuplets]** 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/unsuppress mark
/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

The /x option suppresses the mark unless **[tuplets]** has been used to suppress all irregular note group marks, in which case /x causes the mark to appear. When a dimension is given after /a or /b, the value given is the position above or below the stave of the baseline of the numerical text for a horizontal bracket. Subsequent adjustment of either end of the bracket is then possible, as described above. If no dimension is given after /a or /b, the vertical position is computed from the positions of

the notes that form the group. The left and right movements are available only if no horizontal bracket is being output; 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 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 tuplet 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.

11.6.34 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, tie and beam breaking indicators 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    {g=g=g=}_,g=  {g=g=g=;}g=
```

The only restriction is that (as elsewhere) a tie indicator must precede a beam break.

11.6.35 Treating certain regular groups as triplets

In some music, note pairs consisting of a dotted quaver followed by a semiquaver are intended to be performed as triplets, that is, with the semiquaver taken as a third of a crotchet, instead of a quarter. This notation can be seen in some Telemann manuscripts, for example. In light popular music, such groups may also be 'swung' into triplets. This performance convention does not cause any problems when considering a single part, but for a score that has actual triplets in another part, semiquavers after dotted quavers are not by default vertically aligned with the final triplet notes in other parts. PMW has a facility for adjusting the output so that the noteheads do line up. In the following example, the first bar shows what happens by default; in the second bar the bass part has been 'tripletized'.

The form of the notes remains the same, but their performance length has been adjusted, which affects the horizontal position of the semiquavers. The adjusted note lengths are also used when MIDI output is generated.

The **[tripletize]** directive controls this behaviour. It can be followed by one of the words 'on' or 'off'. If neither is present, 'on' is assumed. At the start of a stave, tripletizing is turned off. When turned on, in addition to dotted quaver pairs, groups consisting of a dotted crotchet followed by a quaver or two semiquavers (notes or rests) are also adjusted. For such groups, a quaver is treated as having the

length of two thirds of a crotchet, or the semiquavers are each treated as having the length of one third of a crotchet, and the dotted crotchet's length is appropriately shortened.

11.6.36 Short cuts for inputting notes

Some short cuts are available to reduce the amount of typing needed when inputting notes. They do not affect the appearance of the output in any way.

- The previous note or chord can be repeated exactly, including its expression marks or ornaments, 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 (☞ 11.7) or a tie may follow, and a subsequent x can be used for further repetitions. For example, a bar of eight identical quaver chords, broken in the middle, can be notated 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 its own 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-
```

A note defined with the letter p can be followed by another p, possibly with a different note length, and x may follow p and *vice versa*. For both x and p, accidentals are not repeated 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 shown, but if p or x follows, the accidentals *are* repeated, according to the usual notation convention.

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

In this example, the minim at the start of the second bar does not have an accidental, but there is one before the following crotchet.

11.7 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** header directive can be used to set the thickness of line used for drawing beams.

11.7.1 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)

These characters are ignored if they follow longer notes. PMW automatically beams across rests that are shorter than a crotchet, unless a break in the beam is specified. A beam breaking character must appear after the end of a note or after the closing parenthesis of a chord, though 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 optionally 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.
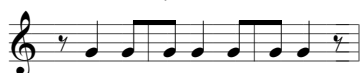
```
g=-a=-b=-,2 c'=-d'=-e'=-
```

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.

11.7.2 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.

This notation causes a beamed group to extend into the subsequent bar; in the example above, two pairs of beamed quavers each straddle a bar line. Without the = characters after the bar characters, each quaver would be separate. 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.

11.7.3 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 header. 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.

11.7.4 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. 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 outputs 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.

11.7.5 Beams with notes on both sides

The stem direction that is determined for a beamed group by the rules described in section 11.8 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 put 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 arrange 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.

11.8 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** header directive (☞ 10.1.122). The 'stem swap level' is normally the middle line of the stave, but can be changed by the **stemswaplevel** directive (☞ 10.1.123).

11.8.1 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 layout of chords containing adjacent notes.

11.8.2 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 (☞ 10.1.122).

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.

11.8.3 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 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 on the other side of the beam (☞ 11.7.5).

11.9 Text strings in stave data

Section 11.12 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 output 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 (☞ 11.11).

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 output in the roman typeface. The directives **[underlayfont]**, **[overlayfont]**, and **[fbfont]** can be used to change the default font for these kinds of text on individual staves.

Text strings, enclosed in double quote characters, are coded in among the notes of a stave. There may be no more than 100 text strings before any one note; this large limit is intended to cover all possible applications. The escape character conventions using the backslash character that apply to all PMW strings are relevant (☞ 8.17). 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 qualifiers, separated from the quote and from each other by slash characters. The following are available:

/a	unspecific text above the stave
/a<*n*>	unspecific text at fixed distance above the stave
/ao	unspecific text above the stave, at the overlay level
/b	unspecific text below the stave
/b<*n*>	unspecific text at fixed distance below the stave
/bu	unspecific text below the stave at the underlay level
/m	unspecific text below the stave, midway to the next stave
/ul	this text is underlay
/ol	this text is overlay
/fb	this text is figured bass
/fbu	this text is figured bass at underlay level
/h	position halfway between notes
/lc<*n*>	position left by <*n*> crotchets
/rc<*n*>	position right by <*n*> crotchets
/bar	position at start of bar
/ts	position at time signature
/c	centre the text
/cb	centre the text in the bar
/e	align end of text
/nc	do not centre
/ne	do not align the end
/box	enclose in a box (mitred corners)
/rbox	enclose in a box (round corners)
/ring	enclose in a ring
/rot<*n*>	rotate by <*n*> degrees
/s<*n*>	use settable size <*n*>, where <*n*> is between 1 and 20
/S<*n*>	use fixed size <*n*>, where <*n*> is between 1 and 10
/u<*n*>	move up <*n*> points

(The group of qualifiers from /bar through /rot is annotated: "ignored for underlay/overlay".)

`/d<n>`	move down *<n>* points
`/l<n>`	move left *<n>* points
`/r<n>`	move right *<n>* points

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`, `/fb`, or `/fbu` 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 11.12.

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 for dynamic marks in keyboard parts. If two overprinting 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`, `/ul`, or `/fbu`, 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
```

By default, if two or more strings precede the same note or the end of a bar, 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. There is, however, a facility for horizontal concatenation (☞ 11.9.5). The level of any string can always be adjusted by the use of `/u` or `/d`.

11.9.1 Horizontal alignment

The horizontal alignment of underlay and overlay strings is described in section 11.12. Except for follow-on strings (☞ 11.9.5), unqualified strings are positioned by default with their first character aligned with the left-hand edge of the next note or rest, or with the bar line if there are no following notes or rests in the bar.

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.

If the `/cb` qualifier is present, the text is centred in the bar. Such a string may appear anywhere in a bar. This qualifier unsets any previous horizontal positioning, but relative movement left or right is honoured.

The `/l` and `/r` qualifiers move a string left or right with respect to the default position, and there are several qualifiers that can be used to vary the position of the alignment point itself. If more than one of these is present, the earliest described here takes precedence (for example, `/bar` overrides `/ts` and `/h` overrides `/rc`).

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.

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.

The alignment point can also be positioned with respect to the musical offset in the bar. This is similar to the 'offset' positioning that is used in MusicXML. Whereas `/l` and `/r` specify a distance in points, `/lc` and `/rc` specify a distance in crotchets. For example:

```
"some text"/rc1 a b
```

The text in this example is aligned with the second crotchet instead of the first. The value given may be fractional, for example:

```
"some text"/rc0.5 a b
```

In this case the text is aligned with where a quaver that follows the first note would be. This is not necessarily halfway between the notes when there are other staves that contain different note lengths. If `/rc` is used on text that precedes the final note of a bar, or if `/lc` is used on text that precedes the first note of a bar, the distance is calculated from the default note spacing.

11.9.2 Enclosed text (boxed or ringed)

The longer the string is, the more elliptical a ring will be. For a single character, the ring is approximately circular. If the `/box`, `rbox`, or `/ring` options are used with underlay or overlay, they apply to each individual syllable.

11.9.3 Text sizes

The `/s` option refers to the sizes of text defined by the **textsizes** header directive (☞ 10.1.128); `/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**. The default size for text that is not underlay, overlay, or figured bass is the size set by the **[textsize]** directive or, if that is not present, the first settable size. Underlay, overlay, and figured bass 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.

As well as the 20 settable text sizes, there are 10 fixed sizes that cannot be changed. The `/S` option is used to select one of these sizes. The following are defined:

```
/S1   12 points
/S2   11 points
/S3   10 points
/S4    9 points
/S5    8 points
/S6    7 points
```

The remaining four are set to 10 points, but may be changed in future so should not be used. These fixed sizes are provided for the use of the standard macro packages, but are not restricted to them.

11.9.4 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. If the string is centred or right-aligned (the /c, /cb, or /e options), this position is computed from the length of a horizontal string, before rotation. That is, rotation happens last.

11.9.5 Follow-on text

By default, a string that immediately follows another string is placed above or below the previous string and may have its own horizontal positioning. There is, however, a facility for horizontally concatenating two or more strings. This is useful when the strings use fonts of different sizes, or are rotated by different amounts. Follow-on strings are characterised by the /F qualifier.

The first string and any follow-on strings must not use the /box or /ring qualifiers. The first string may use any of the available positioning qualifiers, but a follow-on string must not have any explicit positioning qualifiers such as /a, or /ts. The starting point for a follow-on string is the endpoint of the preceding string. The only qualifiers that are permitted for follow-on strings are those for relative movement (/u, /d, /l, /r), /s or /S to select a font size, and /rot to specify a text rotation. A follow-on string that follows a rotated string starts from the rotated end point. Here are some silly examples (text size 2 is defined as 14 points):

```
"small"/a " BIG "/s2/F "small"/F a
"square" "square"/F/rot270
"square"/F/rot180 "square"/F/rot90 a
```
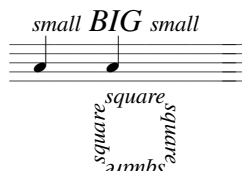

11.10 Fingering indications

The small caps feature of text strings is useful for selecting a smaller than normal font for fingering indications. Alternatively, a specific font size can be defined and used with the /s text qualifier, or one of the fixed sizes can be selected by /S. The music font contains the special characters ♀ and ♂ for indicating the use of the thumb in cello parts. The use of macros is suggested when setting music with lots of fingering, and a suggested set of macros is provided in the **Fingering** standard macro file (☞ 7.2). 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- |
```

11.11 Rehearsal marks

Rehearsal marks are specified as text items enclosed in square brackets. The text may be longer than one character. It is output above the stave, and by default is in bold type and enclosed in a rectangular box with mitred corners. The **rehearsalmarks** directive can be used to change the size and the font, to specify round corners for the box, or to specify a ring instead of a box, or to specify no enclosure at all. If necessary, a rehearsal mark can be moved up, down, left or right, in the same manner as other text. However, any other options that are given with the string are ignored (with a warning).

```
["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 immediately to the right of the preceding bar line when the bar is not the first in a system (that is, not the first in a line). At the start of the first bar in a system, a rehearsal mark is by default aligned with the first note in the bar, but the **rehearsalmarks** directive can be used to move it to the left edge of the system instead. 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 normally appear 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 (☞ 8.18), so that they are always shown above the top stave, whichever staves are selected.

11.12 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, and in addition, the characters # (sharp), – (hyphen), = (equals), and ^ (circumflex) have special meanings. If you actually need one of these characters as a literal, you need to escape it with a backslash.

11.12.1 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. However, after an occurrence of the **[reset]** or **[backup]** directives (☞ 12.2.68, 12.2.5), distribution of underlay syllables is suspended until the overprinting notes or rests have reached the reset/backup point. In the following example, the words 'brown fox' are associated with the final two crotchets in the bar.

```
"The quick brown fox"/ul gg [reset] eeee |
```

Rests are excluded from the underlay distribution process (with one exception, which is described in section 11.12.5). A simple underlay example is shown in chapter 5.

Hyphens are used to indicate that a multi-syllable word is to associated with several notes. All but the final syllable must end with a hyphen. This applies however the input is supplied – either as a separate syllable for each note or as a longer string that PMW splits up. PMW uses one or more hyphens in the output, depending on the distance between the syllables. The header 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 used, centred in the space. Otherwise, a number of hyphens appear, the distance between them being the threshold value divided by three. PMW can alternatively use en-dash characters (or any other characters) as 'hyphens' between syllables of underlaid text. See the **hyphenstring** header directive for details. Whatever is output, 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 is output as a space, but does not count as a space when PMW is splitting up the text. The width of a space is half the size of the font.

```
"Where# hast #tha" c' g-.a= |
```

If the default, 10-point font is in use, this example outputs 'Where' 2.5 points to the left of where it would otherwise appear, and 'tha' 2.5 points to the right.

Sometimes several words are required to appear 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 output.

```
"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, only '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- |
```

There must not be any space characters between the end of a syllable and any following equals. If the final syllable of a word is followed by one or more equals characters, PMW automatically draws an extender line that finishes 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 (☞ 10.1.38). By default, PMW centres all underlay and overlay syllables at the position of their respective notes. The **underlaystyle** directive (☞ 10.1.146) 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 can be specified in multi-syllable fashion before the relevant notes by giving each verse as a separate string.

```
"With-in the wood-lands, flow'r-y gladed"
"When leaves that late-ly were a-spring-ing"
"Let other folk make mon-ey fast-er"
d-; g-a- | b d' e'-d'- | c b
```

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 placed 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 appear. If an underlay string ends with a hyphen, the equals characters can be omitted; PMW automatically outputs 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 1: The shortcuts for omitting equals characters should not be relied on when there is more than one underlay verse because it may not be clear which string belongs to which verse.

Warning 2: There is an 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 square bracket 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. However, if you use the string repetition feature (☞ 8.2.5) to repeat the bar's input characters, this problem does not arise.

```
"one two three four"/ul &*2(ab|)    this works as expected
"one two three four"/ul [2] ab |    this does not
```

11.12.2 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. Individual underlay or overlay strings can specify different sizes by means of the /s or /S qualifier.

11.12.3 Underlay and overlay levels

Text that is marked as part of the underlay or overlay is always positioned 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.

11.12.4 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 included when examining the available space. The minimum space allowed between syllables is one space character in the appropriate font.

There is a header 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** header directive (☞ 10.1.63) 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.

11.12.5 Other uses of underlay and overlay

The underlay and overlay facilities can be used for things other than the words of a vocal part. It is common, for example, for the word *crescendo* to be stretched out, 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 shown (because # is output 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\dim.-"/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 notated 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 for 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 have the 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 an explicit /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 or /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 for 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 handled using an overlay string. However, it is normal to have 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 output at the end of the sequence, once. (If, by some chance, a real vertical bar is required to be repeated, it must be escaped with a backslash.) There are some angle-shaped characters in the music font that can be used for 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 output.

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 output at the start of each continuation line, before the start of the repeating strings. The only option permitted after this string is /s or /S, to set its size (which defaults to the size of the original underlay or overlay string). Using this feature to output a small '8' 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
```

This assumes that size 2 is suitably defined using the **textsizes** directive.

12. Stave directives

This chapter describes the directives that can appear interspersed with the notes and rests of a stave. They must be enclosed in square brackets, though more than one may appear within a single set of brackets. There are three other items that can appear between square brackets: repeated expression marks (☞ 11.6.22), repeated bar counts (☞ 11.2), and rehearsal marks (☞ 11.11).

12.1 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 (☞ 10.1.24). 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
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. **[hclef]** and **[noclef]** behave like a treble clef for note pitch and positioning. 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 output 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** header directive is used to specify the size of such clefs.

12.2 Alphabetical list of stave directives

The stave directives are now described in alphabetical order.

12.2.1 [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 6.6. 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

shown only above the top stave of a score. If parts are to be extracted from a score, first and second time marks should be specified on stave 0 (☞ 8.18), so that they will be output above the top stave, whichever staves are selected.

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 output 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 output 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.

12.2.2 [All]

See the immediately preceding section.

12.2.3 [Alto]

This specifies a C clef with its centre on the third stave line (☞ 12.1).

12.2.4 [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 output.

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

This example has the effect of changing the stave into the bass clef so that 'gbc' are output in this clef, and a bass clef is shown at the next start of line, but no clef is output 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.

12.2.5 [Backup]

This directive moves the musical position in the bar back to that of the previous note, thus allowing the previous note to be overprinted. To overprint a whole bar, see **[reset]** (☞ 12.2.68), and to overprint a whole stave, use another stave with zero stavespacing (☞ 10.1.120). **[Backup]** is an alternative to **[reset]** followed by invisible rests, but is not exactly equivalent:

- After **[reset]** the state of the implied accidentals (relevant for MIDI output) is re-initialized from the key signature, whereas there is no change for **[backup]**.

- **[Backup]** is permitted inside irregular note groups such as triplets, provided that it follows at least one note, whereas **[reset]** is not allowed.

Here's an example of the use of **[backup]** to create a non-standard shorthand for a minim that is to be played as four quavers. A macro is used to demonstrate how this can be packaged for multiple uses.

```
*define X() &&1-\nz\[backup]&&1+
[stave 1 treble 1]
&X(g) #&X(g)\>\ &X(c')
```


The argument of the macro is a note letter with an optional octave indication. This is first output as a quaver without a notehead (using \nz\), followed by an overprinted minim, which is coded as a doubled crotchet so that a single lower case letter can be used for both notes. Note that the shorter of the two notes is given first so that the whole item has the length of a minim. Note also that an accidental may precede the whole item, and note options may follow.

12.2.6 [Baritone]

This specifies an F clef based on the third stave line (☞ 12.1).

12.2.7 [Barlinestyle]

This directive must be followed by a number, and it sets the bar line style for subsequent bar lines in the stave (☞ 10.1.7).

12.2.8 [Barnumber]

The header directive **barnumbers** (☞ 10.1.9) is used to request automatic bar numbering. The stave directive **[barnumber]** is used to control bar numbering for individual bars. If **[barnumber]** appears without any arguments, a number is output for the current bar, independently of the overall setting. This is the only way to generate a number with a fractional part for an uncounted bar, or to generate a number for bar 1, which is not normally numbered. The size of font used and whether or not the number is enclosed in a box or ring is controlled by the header 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 outputs 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 output for the current bar, even if the header directive implies there should be one.

12.2.9 [Bass]

This specifies a bass clef, that is, an F clef on the fourth stave line (☞ 12.1).

12.2.10 [Beamacc]

This directive causes the next beam to be drawn as an accelerando beam (☞ 11.7.4).

12.2.11 [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.

12.2.12 [Beamrit]

This directive causes the next beam to be drawn as an ritardando beam (☞ 11.7.4).

12.2.13 [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.

12.2.14 [Bottommargin]

This directive provides a way of changing the value given by the **bottommargin** header 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.

12.2.15 [Bowing]

String bowing marks are normally output above the stave. The **[bowing]** directive is provided for changing this. It must be followed by one of the words 'above' or 'below'.

12.2.16 [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]**.

12.2.17 [Cbaritone]

This specifies a C-clef on the 5-th stave line (☞ 12.1).

12.2.18 [Comma]

The **[comma]** directive inserts a comma ❜ pause mark above the current stave. See also **[tick]**.

12.2.19 [Contrabass]

This specifies a bass clef with a little '8' below it (☞ 12.1).

12.2.20 [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 (☞ 8.18) is overprinting 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** header 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).

12.2.21 [Couple]

A single music part, notated as one PMW stave, can be spread across two staves. This is a common requirement in keyboard music. If **[couple up]** is given on a bass clef stave, it specifies that notes higher than middle C should be positioned 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 coupling is given in section 6.9.4.

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 positioned 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.

12.2.22 [Cue]

The directive **[cue]** causes the subsequent notes of the current bar, on the current stave, to be output 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 for single parts; 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         @ output 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 handling complicated ornaments or optional notes. Another way of handling optional notes is to use the \sm\ note option ☞ 11.6.13.) When cue notes are dotted, the dots are spaced horizontally in proportion to the size of the cue notes. However, when there are 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.

12.2.23 [Deepbass]

This specifies an F clef based on the fifth stave line (☞ 12.1).

12.2.24 [Dots]

Augmentation dots are normally put 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 (☞ 11.6.17).

12.2.25 [Doublenotes]

This directive causes the length of subsequent notes in the current stave to be doubled. Unlike the header directive with the same name (☞ 10.1.31), it does not affect time signatures. It is, however, cumulative with the header directive and with **[halvenotes]** (☞ 12.2.37).

12.2.26 [Draw]

The **[draw]** directive is described in chapter 9.

12.2.27 [Endcue]

See **[Cue]** (☞ 12.2.22) above.

12.2.28 [Endline]

See **[line]** (☞ 12.2.41) below.

12.2.29 [Endslur]

See **[slur]** (☞ 12.2.76) below.

12.2.30 [Endstave]

The data for each stave of music must end with the directive **[endstave]**. This can be followed only by the start of a new stave or the start of a new movement or by the end of the file.

12.2.31 [Ensure]

The **[space]** directive always inserts extra space before a note. Sometimes all that is needed is an assurance that a certain amount of space is available, for example, when using the drawing facilities for 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 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 are 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.

12.2.32 [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.

12.2.33 [Fbtextsize]

This directive must be followed by a number in the range 1 to 20. It selects the default size to be used for figured bass text on the current stave. The actual font sizes that correspond to the twenty available sizes are set by the **textsizes** header directive. If this directive is not used, the size used is the one set by the **fbsize** header 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.

12.2.34 [Footnote]

This directive usually defines a text string that is output below the bottom system of the page in which the current bar appears. However, the syntax of **[footnote]** is the same as the syntax of the **heading** and **footing** directives (☞ 10.1.51), and therefore **[footnote]** may alternatively define a drawing. Footnotes are different from footings, in that the space in which they are placed is taken from the normal page length; consequently the bottom system of music is positioned higher up the page, in order to leave room for footnotes. If a textual footnote is longer than the line length, it is automatically split into several lines in the same was as headings and footings.

```
[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** header directive. An individual footnote may override the default by including a number before the text string. If there are several footnotes on one page, extra vertical white space is left between them. The default amount of extra space is 4 points, but this can be changed by the **footnotesep** header directive. If there are several footnotes in one system, they are ordered by stave, those for the lowest numbered stave appearing first.

12.2.35 [Hairpins]

Hairpins (☞ 11.5) are normally placed 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 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 8 points above the stave, whereas the latter adds 8 points to whatever position PMW computes from the notes under the hairpin.

12.2.36 [Hairpinwidth]

This directive, which must be followed by a dimension, sets the default width of the open ends of any subsequent hairpins on the current stave. The overall default is 7 points. The width of the open end of an individual hairpin can be set by following the initial < or > character with /w and a dimension.

12.2.37 [Halvenotes]

This directive causes the length of subsequent notes in the current stave to be halved. Unlike the header directive with the same name (☞ 10.1.50), it does not affect time signatures. It is, however, cumulative with the header directive and with **[doublenotes]** (☞ 12.2.25). For example, if there are many bars consisting mostly of quavers in a stave, using **[halvenotes]** saves having to type hyphens after each lower case note letter. **[Doublenotes]** can be used to restore the previous state.

12.2.38 [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 (☞ 12.1).

12.2.39 [Justify]

The justification parameters can be changed by the appearance of this stave directive. Unlike the header 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.

12.2.40 [Key]

This directive specifies a change of key signature. If this falls at the start of a system, a cautionary key signature is output at the end of the previous system unless the word 'nowarn' is included in the directive.

```
[key E$ nowarn]
```

There is also a header directive, **nokeywarn**, for suppressing all cautionary key signatures. Key signature changes are shown in 'modern style'. That is, unless the new key signature is empty (C major, A minor, the pseudo-key N, or an empty string from **printkey**), all that is output 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 a key with an empty signature. For example, in the default configuration (no **printkey** settings):

```
[key C key A]
```

This example uses a number of naturals to cancel the previous signature before outputting three sharps for A major. When a bar starts with a new key signature and a repeat mark, the order in which these appear depends on their order in the input.

```
[key G] (:
```

This example causes the key signature to be 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 put first on all staves.

12.2.41 [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]** (☞ 12.2.76) and internally shares the same code, 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 with optional text or drawing 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 (☞ 12.2.83). The /= option that is used in **[endslur]** to identify tagged slurs is also available in **[endline]** for tagged lines. Slurs and lines share the same name space for tags.

12.2.42 [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 referenced, 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 outputs 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 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`, `/S`, `/box`, `/rbox`, 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 for some text in the middle of a horizontal line:

```
[line/a/h linegap/h/"unis. \it\ad lib"] efge | bag [el] r |
```
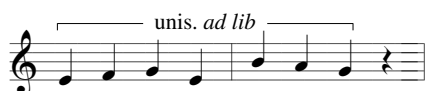

Because the `/h` option is used, the **[linegap]** directive can be given right at the start of the line. Do not confuse `/h` as used with **[line]**, where it means *horizontal*, with `/h` as used with **[linegap]**, where it means *halfway*. This latter usage was chosen to be similar to the `/h` option 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 ℘ℯ𝒹 text at the start, which is generated by 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 it is.

12.2.43 [Mezzo]

This specifies a C clef with its centre on the second stave line (☞ 12.1).

12.2.44 [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 optionally be given after the voice name, separated by a slash.

```
[midichannel 1 "trumpet"/12]
```

See the **midichannel** header directive for how to set up channels at the start of a piece.

12.2.45 [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** header 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.

12.2.46 [Miditranspose]

The **[miditranspose]** directive can be used to change the playing transposition of a stave, which is initially set from the **miditranspose** header 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.

12.2.47 [Miditremolo]

This directive changes the MIDI tremolo setting for the current stave. Its argument is one of the words 'repeat', 'trill', 'both', or 'none'. For more details see the **miditremolo** header directive (☞ 10.1.78), which sets the default for all staves.

12.2.48 [Midivoice]

The **[midivoice]** directive can be used to change the MIDI voice without changing the channel. Like **[midichannel]** a relative volume may optionally be given after the voice name, separated by a slash.

```
[midivoice "french horn"]
[midivoice "piano"/12]
```

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.

12.2.49 [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.

12.2.50 [Move]

If the **[move]** directive is followed by a single number, it causes the next non-textual thing 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 positions a mid-stave clef two points to the left and four points higher than normal. Vertical movement does not apply to notes, and is ignored if specified. It does apply to rests. See **[rlevel]** and section 11.6.26 for other ways of moving rests vertically. If more than one is used, the effect is cumulative.

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. Text items and slurs have their own movement options.

Those items to which the **[move]** directive applies are: clefs, key signatures, time signatures, dotted bar lines, repeat marks, caesuras, commas, ticks, rests, and notes (horizontal movement only). If **[move]** 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 output above the long rest sign. See also **[ensure]**, **[rmove]**, **[rsmove]**, **[rpace]**, **[smove]** and **[space]**.

12.2.51 [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 and/or drawings that appear 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 shows 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 (☞ 12.2.90). Each occurrence of **[name]** adds to the list of strings and drawings.

The second form of **[name]** is used to change what appears 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 systems. Each start-of-stave 'item' consists of a string or a call to a drawing function, or both, and the items 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 any output; alternatively, empty strings can be used.

12.2.52 [Newline]

The directive **[newline]** can be used to force a new line (system) of music to be started for a particular bar. It should normally be at the start of a bar, but it need only appear in one stave that is selected. If a stave is not selected, an appearance of **[newline]** within it is ignored. See also the **layout** header directive.

12.2.53 [Newmovement]

This directive must appear either immediately after an **[endstave]** directive, or in the header section of a movement. In the latter case it terminates a movement that has no staves; this could be used for a title page. There are a number of optional arguments, described below, of which `newpage`, `thispage`, and `thisline` are mutually exclusive.

[Newmovement] is followed by a new set of header directives and stave data. When it 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 output on all subsequent pages, including pages that are the start of new movements. Section 8.1.7 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 output, 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 put 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 have 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, putting the **lastfooting** text on the current page, 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's first stave is 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 have two or more of them side by side.

- One style of incipit 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.

12.2.54 [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, however, an appearance of **[newpage]** within it is ignored. See also the **layout** header directive.

12.2.55 [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 header 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.

12.2.56 [Noclef]

This specifies an invisible clef (☞ 12.1). 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.

12.2.57 [Nocount]

If the first bar of a piece is incomplete, it is conventional not to include it in the bar numbering. Occasionally this applies in other situations, for example, when using an 'invisible bar line' to make PMW split a bar over two systems. The directive **[nocount]** causes the current bar not to be counted; such a bar is never numbered by the **barnumbers** header directive, but an explicit **[barnumber]** within a stave is honoured (unless it is zero, the number will have a decimal fraction). **[Nocount]** need appear in only one stave. If it appears anywhere in a bar whose contents are repeated by means of a number in square brackets, all the repeated bars are uncounted. Section 8.3 explains how PMW identifies uncounted bars if it needs to refer to them, for example, in an error message.

12.2.58 [Noteheads]

Four alternative notehead shapes are supported: diamond-shaped for string harmonics, cross-shaped, circular, 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 additional form of notehead. You can use any of the notehead shapes without stems.

[Noteheads] controls the default settings of these features for subsequent notes. It must be followed by one of the words 'normal', 'harmonic', 'cross', 'circular', 'none', 'only', or 'direct'.

For individual notes, option settings such as \nh\ and \nx\ (☞ 11.6.17) can be used to override the default, which makes it possible to have different noteheads within a chord.

[Noteheads only] and **[noteheads direct]** cause subsequent notes not to have stems. For the former, the notehead type is not altered, so **[noteheads cross noteheads only]** gives stemless crosses, and **[noteheads none noteheads only]** makes subsequent notes invisible. Any setting other than 'only' or 'direct' enables note stems.

Because this is quite a long directive to type, and one which might appear frequently in some music, abbreviations are provided as follows:

```
[c]     is equivalent to     [noteheads circular]
[h]     is equivalent to     [noteheads harmonic]
[o]     is equivalent to     [noteheads normal]
[x]     is equivalent to     [noteheads cross]
[z]     is equivalent to     [noteheads none]
```

When **[noteheads direct]** is selected (to show notes as ⌒), ledger lines are output as normal, but there are no stems or beams. When no noteheads are being output (**[noteheads none]**), breves and semibreves are completely invisible. Stems and beams are still drawn for other notes, but no ledger lines. 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 used, and minims and semibreves look the same with circular noteheads.

12.2.59 [Notes]

The directive **[notes off]** suppresses the output 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 ornaments or fermatas are specified, these are shown. Making the note or chord at the end of a tie invisible is a convenient way to output 'hanging' tie marks (☞ 11.6.29). 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. This can be changed by including the header directive **midifornotesoff**.

12.2.60 [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** header directive.) Trailing zero values can be omitted.

```
[notespacing 0 0 3 -2]
```

This example specifies that minims are to appear 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 are below it), 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, it can be given on stave 0.

12.2.61 [Octave]

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 (which is the default). The **[octave]** directive, which

must be followed by a number, requests transposition by the number of octaves given. The octave can also be specified at the same time as the clef (☞ 12.1).

Each octave setting replaces the previous one; they are not cumulative. The argument for **[octave]** may be positive or negative:

```
[octave -1]     C is the note two octaves below middle C
[octave 0]      C is the note one octave below middle C
[octave 1]      C is middle C
```

Octave transposition is in addition to any general transposition that is in force.

12.2.62 [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 (☞ 12.2.118).

12.2.63 [Oltextsize]

This directive must be followed by a number in the range 1 to 20. It selects the default size to be used for overlay text on the current stave. The actual font sizes that correspond to the twenty numbers are set by the **textsizes** header directive. If this directive is not used, the size set by the **overlaysize** header 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.

12.2.64 [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** or **setcolor** 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** or **setcolor** with **[overdraw]** makes it possible to 'white out' parts of staves.

12.2.65 [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.

12.2.66 [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]
```
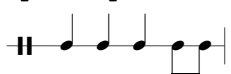
12.2.67 [Printpitch]

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 visible pitch instead of the playing pitch is available. The **[printpitch]** directive takes a note letter and optional octave indication as its argument. It causes all subsequent notes on the stave to be output 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 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 8/1 hclef 0]
[printpitch b' stems up] d`d`d` $a-\sd\$a-   |
```


After the stave number, /1 defines a one-line stave. 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 stave, and arrange for them to overprint each other. Then the appropriate MIDI sound can be permanently set for each stave.

12.2.68 [Reset]

Sometimes it is convenient to notate a bar as two different sequences of notes, to be overprinted on the stave. **[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. If only a single note is overprinted, the **[backup]** directive (☞ 12.2.5) may be more convenient.

[Reset] should be used with caution, because it can cause unexpected effects if items such as slurs are in use. The distribution of pending underlay or overlay syllables is suspended after **[reset]** until the position in the bar has 'caught up' to the reset point.

```
[stems up] gabc' [reset] [stems down] efga  |
```

This example outputs 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. 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 non-note 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.

12.2.69 [Resume]

This directive forces a resumption of a suspended stave – see **[suspend]** for details.

12.2.70 [Rlevel]

Rests are normally positioned 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. A note option (☞ 11.6.17) or **[move]** can be used to do this for individual rests. 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 12 points lower than normal, so that a whole bar rest, which normally appears below the fourth line, is now below the bottom line of the stave. Semibreve and minim rests that are moved off the stave are shown with a single ledger line to indicate which they are. Each occurrence of **[rlevel]** sets a new level relative to the default position. They are not cumulative.

12.2.71 [Rmove]

This directive operates exactly as **[move]**, except that the horizontal dimension is scaled to the relative stave size.

12.2.72 [Rsmove]

This directive operates exactly as **[smove]**, except the horizontal dimension is scaled to the relative stave size.

12.2.73 [Rspace]

This directive operates exactly as **[space]**, except that the horizontal dimension is scaled to the relative stave size.

12.2.74 [Sghere], [sgabove], and [sgnext]

These directives affect the system gap value, that is, the amount of vertical space that is left between systems. Note that when just one stave is being output (typically for a single part), it is the system gap that determines the spacing. When vertical justification is enabled (☞ 10.1.56), the system gap is a minimum amount of space.

[Sghere] changes the spacing below the current system (that is, the one in which the current bar is to appear), **[sgabove]** changes the spacing above the current system, and **[sgnext]** makes the change for all systems that follow the current one, but not for the current system itself. These directives can be placed on any of the staves that comprise the system. For all of them, a single number is required as an argument. It can be preceded by a plus or a minus sign to indicate a relative change from the existing value.

If there is more than one occurence of these directives in a system, any that are in stave 0 are processed first, in bar number order, followed by those in the following staves. The last absolute setting (if any) is used, with any relative changes acting cumulatively. Thus, for example, an absolute setting in stave 2, bar 1 will override a relative setting in stave 1, bar 4.

The system gap is used only between systems, not at the top or bottom of a page. If **[sghere]** is present in the last system on a page, or **[sgabove]** in the first system on a page, they have no effect.

12.2.75 [Skip]

When setting vocal or keyboard music it is common to use two overprinting staves for notes with stems in different directions. Sometimes 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 correct 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 before proceeding with the current bar. If there is more than one **[skip]** their effect it cumulative. A warning is given if **[skip]** is not at the start of a bar because wherever it appears, its effect is the same (that is, empty bars are inserted before the current bar).

```
[stave 2 bass 0] gg | [skip 50] aa | [endstave]
```

This example defines a stave in which only bars 1 and 52 are defined. The intermediate bars are treated as being completely empty. If **[skip]** is required at the very start of a stave, do not include a clef setting, unless you want a clef as part of the resumed bar after the skip. You can use the **[assume]** directive to set a clef without outputting anything. See also the the **omitempty** option for **[stave]**.

12.2.76 [Slur]

Slurs between adjacent single notes can be input by inserting an underline character after the first note (☞ 11.6.27). 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 (☞ 12.2.41).

12.2.77 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

Another way of specifying horizontal position for slur ends is to use /llc, /lrc, /rlc or /rrc followed by a number that may have a fractional part. These options specify a base horizontal position for a slur end as a musical offset, in units of crotchets. At the start of a slur this offset is relative to the next note. By default, at the end of a slur it is relative to the previous note (that is, the last enclosed note). However, if the /cx option is present, an ending offset is taken relative to the following note. These features were added to PMW in order to simplify processing MusicXML input. 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 cumulative. 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 notating 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 has 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 6.5.

12.2.78 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.

12.2.79 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.

12.2.80 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.

12.2.81 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** header 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 or not the slur is split.

- 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.

12.2.82 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 – something 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.

12.2.83 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 ASCII alphanumeric character, which acts as an identifier. It is recommended that capital letters normally be used, as they are visually distinctive. Tagging is also available for lines drawn with **[line]**. These are just treated as differently drawn slurs, and they share the same tagging name space. 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] |
```

12.2.84 [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.

12.2.85 [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. **[Smove]** can take two arguments, like **[move]**, with the second one specifying vertical movement for the move. The horizontal value is not scaled by the stave size – use **[rsmove]** if you want this to be scaled.

12.2.86 [Soprabass]

This specifies a bass clef with a little '8' above it (☞ 12.1).

12.2.87 [Soprano]

This specifies a C clef with its centre on the bottom stave line (☞ 12.1).

12.2.88 [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]**.

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 positioned relative 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
```

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.

The handling of multiple occurences of **[space]** at the same point in the bar depends on their positioning. Multiple occurrences on the same stave are cumulative, so the following are equivalent:

```
[space 2] [tick] [space 2] A
[space 4] [tick] A
[tick] [space 6 space -2] A
```

After any accumulation, PMW checks the extra space value for this position, as set by any lower-numbered stave(s), with default zero. If the new value is non-negative, it is retained if it is greater than the current setting; otherwise it is retained if it is less. This normally gives the right effect if extra space is accidentally specified in two different staves. See also **[ensure]**, **[move]**, **[smove]**, **[rspace]**, **[rmove]**, and **[rsmove]**.

12.2.89 [Ssabove], [sshere], and [ssnext]

These directives affect vertical stave spacing in the current movement only. See section 10.1.120 for how to set initial values for the movement that also apply by default to subsequent movements.

[Ssabove] differs from the other two in that it sets a minimum spacing *above* staves; the other two adjust the actual settings for spacing below staves. For example, if the spacing for stave 2 has been set to 50, an occurrence of **[ssabove 40]** in stave 3 would have no effect, but **[ssabove 60]** would increase the distance between staves 2 and 3.

The difference between the other two directives is that **[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 any of these directives is followed by a single number, the setting 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. 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. If there is more than one occurence of any of these directives in a system, those in stave 1 are processed first, in bar number order, followed by those in the following staves. For any given stave, the last absolute setting (if any) is used, with any relative changes acting cumulatively. Thus, for example, an absolute setting in stave 2, bar 1 will override a relative setting in stave 1, bar 4.

When **[ssnext]** is used with a plus or minus sign, without a preceding absolute setting, the value is relative to the original spacing for the stave, ignoring any changes that might have been made with **[sshere]**.

12.2.90 [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. This may be followed by a slash and a number in the range 0–6, specifying the number of stave lines (default 5). 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** header directive. The **contrib** directory in the PMW distribution contains an example of guitar tablature.

The number of stave lines has no implications for key signatures or clefs. For staves with fewer than five lines, ledger lines are not shown for notes that are off the stave. On one-line staves, whole bar rests are output under the single line, and on three-line staves they are under the top line. In all other respects the behaviour of PMW is unchanged by the number of stave lines.

Following the stave number and optional number of lines, the word **omitempty** may appear. This causes nothing whatsoever to be output for completely empty bars. See section 12.2.94 below for details. Further arguments may be given to specify text or drawings to be output at the lefthand side of the stave. Most commonly, **[stave]** is used just with text, and this form is described first.

12.2.91 Text at stave starts

The text-only form of **[stave]** has the following format, where the stave number *n* can be followed the number of stave lines and/or `omitempty`, as just described:

```
[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 subsequent systems (see the next section for multiple strings for the same system). 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 has '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 at the start. 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 (☞ 12.2.51), 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 outputs two lines at the start of the stave 5 in the first system. The amount of space used for start-of-stave text depends on the longest text line for any stave. By default, shorter lines are left-aligned in this space, but the options /c and /e can be used to centre or right-justify the lines. 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. If there is only one line, it is right justified.

You can position text vertically halfway between two successive staves by appending /m (for 'middle') to the text on the upper of the two staves.

```
[stave 1 "Piano"/m]
```

If two overprinting 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, unless such a stave is suspended, in which case /m is ignored. It is also ignored if the following stave is suspended. As an example of where this is relevant, consider this input:

```
[stave 1 "Horns"/m "Hn"/m]
...
[endstave]
[stave 2]
...
[endstave]
```

When both horn parts are being output, the text is shown halfway between them, but if the second stave is suspended, the text aligns with the first stave.

You can specify a size for text at the start of a stave by following the string with /s or /S and a number. For /s, the number selects a text size from the list given to the **textsizes** directive, whereas /S selects one of the fixed sizes (☞ 11.9.3).

```
[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, a 10-point font is used.

You can also specify that the text be rotated through 90° so that it runs vertically up the page and its midpoint is approximately at the middle of the stave. This is specified with the /v option; /c and /e are ignored.

```
[stave 3 "Chorus"/v]
```

Only a single line of text is supported when output is vertical, and hence the vertical bar character has no special meaning in this case. When /v is combined with /m, the text is both rotated and moved down so that its midpoint is halfway between the staves.

Finally, you can use /u, /d, /l, or /r to move text up, down, left, or right from its default position. When there is more than one line, these qualifiers apply to all of them. For vertical text, 'up' still means up the page.

If more than one string is given for any stave, the various qualifiers can be used on any of them, and apply only to those strings for which they appear.

12.2.92 Multiple strings at stave starts

Sometimes it is useful to be able to specify two or more strings, possibly at different sizes or with different options, to be output together at the start of the same stave. This is notated by ending a string with a slash followed immediately by the next string, with no intervening space. Here is an example where this feature is useful:

The start of the input for stave 1 in this example is:

```
[stave 1 "1"/e/"Horns in F"/m/s2 treble 1]
```

Both strings are output at the start of the stave, with /m being used to move the second one down to the halfway point before the second stave, and /s being used to select a different size. If /m had not been used, the two strings would have overprinted each other. This facility is not confined to the first string in a **[stave]** directive; it can be used with any string.

12.2.93 Drawings at stave starts

A drawing function (see chapter 9) can be specified at the start of a stave, 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 generate 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 is 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.

12.2.94 Omitting empty bars

When a stave is about to be suspended (☞ 8.19, 12.2.98), it is sometimes desirable not to output 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 **omitempty** option is specified for a **[stave]** directive, *nothing at all* is output 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]
```

A 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 output 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 is needed at its start. By default, a conventional solid bar line is output, 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.

```
[stave 1 omitempty treble 1] ggg |? [skip 3] |? aaa |
```


This example specifies no bar line 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 positioning 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.

12.2.95 [Stavelines]

This is an obsolete directive that was used to set the number of stave lines. Because this is a characteristic of the entire stave, this parameter is now set as part of the **[stave]** directive. The old directive still works, with a warning, but may be removed in a future release.

12.2.96 [Stemlength]

The **[stemlength]** directive is used to set a default value for the stem length adjustment on a stave. It applies to all stemmed notes, both single and beamed.

```
[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** header directive for details.

12.2.97 [Stems]

This directive controls the vertical direction of stems and can also adjust their horizontal positioning. Normally PMW chooses for itself in which direction to draw note stems. Details of the rules it uses are given in section 11.8. Some variation is possible by means of the **stemswap** header directive (☞ 10.1.122). 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. When used for this purpose 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.

The **[stems]** directive may also be followed by 'central'. This causes note stems to be positioned horizontally aligned with the centres of noteheads instead of at the edges. This feature is provided for some experimental music notation; it can be turned off by the word 'beside'. Currently this works only with full-sized notes, not cue or grace notes.

12.2.98 [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 included. 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 on this stave in that system. The suspension ends automatically when a system is encountered in which there are notes or text 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 output before the stave is suspended. If no rest bar is wanted 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 output, **[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** header directive is provided for this purpose.

12.2.99 [Tenor]

This specifies a C clef with its centre on the fourth stave line (☞ 12.1).

12.2.100 [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, which in two cases may optionally be followed by a dimension.

```
[text above]          output above the stave
[text above <n>]      ditto, at a fixed level
[text below]          output below the stave
[text below <n>]      ditto, at a fixed level
[text underlay]       default is underlay
[text overlay]        default is overlay
[text fb]             default is figured bass
[text fbu]            default is figured bass at underlay level
```

The words `above` or `below` may be followed by an unsigned number, which specifies a fixed level above or below the stave, respectively. To override a default with a fixed level (for example `[text above 15]`) on an individual string, the text options `/a` or `/b` without a following number can be used (as well as `/ul`, `/fb`, `/fbu`, or `/m`). Similarly, the appearance of **[text above]** or **[text below]** without a number resets to the initial state, where the default level depends on the next note.

Ordinary text that is positioned above or below the stave is by default output at settable size 1 (specified by the **textsizes** header directive), unless the **[textsize]** directive has specified otherwise. Underlay, overlay, and figured bass text is output by default at the sizes specified by the **underlaysize**, **overlaysize**, and **fbsize** header directives. The default text type and size can always be overridden by explicit qualifiers following the string. For example, if **[text underlay]** has been specified, an italic dynamic mark above the stave is coded like this:

```
"\it\ff"/a
```

The default text type can be changed many times within one stave.

12.2.101 [Textfont]

The default typeface for text other than underlay, overlay, or figured bass 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 this kind of text is italic. The defaults for underlay, overlay, and figured bass text are set by **[underlayfont]**, **[overlayfont]**, and **[fbfont]**, respectively. In any given text string it is always possible to change font by using the appropriate escape sequence.

12.2.102 [Textsize]

This directive must be followed by a number in the range 1 to 20. It selects the default size to be used for text that is neither underlay nor overlay nor figured bass, which have their own size setting directives. The actual font sizes that correspond to the twenty numbers are set by the **textsizes** header 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, or by `/S`, which selects from a set of fixed sizes.

12.2.103 [Tick]

The **[tick]** directive causes PMW to insert a tick √ pause mark above the current stave. See also **[comma]**.

12.2.104 [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', 'up', 'below', 'down', or 'auto' – the last causing a reversion to the default state. The effect of this directove on chords is to force *all* the tie marks for a chord to the given side of the noteheads by default. However, when overriding the default for an individual chord, you can specify that some are above and some below the noteheads (☞ 11.6.27).

12.2.105 [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 output at the end of the previous line unless the word 'nowarn' is included in the directive.

```
[time 6/8 nowarn]
```

There is also a header 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 (☞ 12.2.106).

When a bar starts with a new time signature and a repeat mark, the order in which these appear depends on their order in the input.

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

The first example causes the time signature to be 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 put first on all staves.

Sometimes two time signatures are needed 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** header directive.

12.2.106 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 that is output, and this corresponds to the notes in the bar; the second signature is the one from the other stave. The notes are stretched or compressed (in position on the stave and in time when generating a MIDI file) to make the bar lengths match.

12.2.107 [Topmargin]

This directive provides a way of changing the value given by the **topmargin** header 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.

12.2.108 [Transpose]

The **[transpose]** directive specifies that subsequent notes on the current stave are to be transposed by a number of semitones. A positive number transposes upwards; a negative number transposes downwards. If transposition is also specified at an outer level (either in the header, or by using the **-t** command line option), the transposition specified here adds to, rather than replaces it, as does any subsequent appearance of **[transpose]**. 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 that is already set for the current stave when it encounters the **[transpose]** directive, but it does transpose any subsequently encountered key signatures, except for the non-transposing pseudo-key N. To ensure a transposed key signature for a stave which has its own transposition specified, you should include the key signature after **[transpose]**, even if it is the same key signature that is specified in a header 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 8.10, and details of the transposition of chord names are given in section 8.17.7.

12.2.109 [Transposedacc]

This directive must be followed by one of the words 'force' or 'noforce'. It changes the option for forcing an accidental on a transposed note when there was an accidental on the original, even if the accidental is not strictly needed (☞ 8.10).

12.2.110 [Treble]

This specifies a treble clef (☞ 12.1).

12.2.111 [Trebledescant]

This specifies a treble clef with a little '8' above it (☞ 12.1).

12.2.112 [Trebletenor]

This specifies a treble clef with a little '8' below it (☞ 12.1).

12.2.113 [TrebletenorB]

This specifies a clef that is exactly like the trebletenor clef, except that the little '8' is enclosed in parentheses.

12.2.114 [Tremolo]

Tremolo marks that appear as beams between notes that are not normally beamed, or as disconnected beams between notes, are notated by **[tremolo]** between the two notes (or chords). 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 outputs two crotchets with two disconnected beams between them.

```
G [tremolo/x3/j3] B
```

This example outputs 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 use a note of the 'wrong' length. For example, a tremolo that lasts for the length of a crotchet is shown 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 11.6.17. Here are some examples:

```
d-\m\ [tremolo] g-\m\;
d-\m\ [tremolo/j1/x3] g-\m\;
d\M\ [tremolo] g\M\
```


The **[tremolo]** directive must appear between two notes in a bar. An error occurs 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. From release 5.32 **[tremolo]** has an effect on MIDI output.

12.2.115 [Tripletize]

This directive enables and disables 'tripletizing', which causes certain groups of regularly notated notes to be positioned (and played if MIDI is generated) as triplets. Details are given in section 11.6.35.

12.2.116 [Triplets]

This directive is a synonym for **[tuplets]**.

12.2.117 [Tuplets]

This directive is used to control the marks on triplets and other irregular note groups. It must be followed by one or more of the words 'on', 'off', 'above', 'below', 'bracket', 'nobracket, or 'auto'. 'On' and 'off' apply to the number, with or without a bracket, that appears above or below a tuplet group. The default is 'on'. When 'off' is specified, nothing is output. Note that the qualifier /x can be used to suppress the number for an individual tuplet, or to enable it, if it has been previously disabled. The other words set default options for irregular note groups, and they are independent of each other.

```
[tuplets above]
```

This example causes all the irregular note marks to be placed 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.

```
[tuplets above +4]
```

This example causes subsequent marks to be 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 are placed 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 **[tuplets]** directive. Note that the use of `/a` or `/b` forces a bracket to be drawn, unless followed by `/n`.

12.2.118 [Ulevel] and [ulhere]

For each stave, PMW computes a default level for 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. **[Ulhere]** specifies a temporary change for the current line, and **[ulevel]** sets an absolute level to be used until further notice. **[Ulhere]** takes a positive or negative number of points as an argument. This is added 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. **[Ulevel]** takes effect for the text 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.

12.2.119 [Ultextsize]

This directive must be followed by a number in the range 1 to 20. It selects the default size to be used for underlay text on the current stave. The actual font sizes that correspond to the twenty numbers are set by the **textsizes** header directive. If this directive is not used, the size set by the **underlaysize** header 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.

12.2.120 [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 output 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]**.

12.2.121 [Underlayfont]

The default typeface for underlay text 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.

12.2.122 [Xline]

See **[line]** (☞ 12.2.41) and also section 12.2.82.

12.2.123 [Xslur]

See section 12.2.82.

13. PostScript vs PDF

The ability to create PDFs directly was introduced at PMW release 5.30. The PDF format is now standardised and is widely supported by viewing and printing software. Earlier versions of PMW can generate only PostScript, which is a page description language, originally created by Adobe for controlling laser printers. Some printers can interpret PostScript directly; for others intermediate translation software such as *GhostScript* is needed. Appropriate translation is often automatically installed in printing systems. *GhostScript* can be used to interpret PostScript for display on the screen and to convert it into a PDF.

A PDF created directly by PMW will be larger than one created by running PostScript output through *GhostScript* because *GhostScript* can deconstruct fonts and include only those characters that are actually used in the PDF. PMW always includes entire fonts when it creates a PDF, except for the fourteen 'standard' fonts that can be assumed available in all PDF processors.

PMW supports only OpenType (`.otf`) fonts when generating PDF output. The PDF format does support other types of font, but as there are several utilities for converting PostScript Type 1 and TrueType fonts to OpenType, this is not seen as a major issue.

14. Changes for release 5.00

There was a major revision of the code for release 5.00. Many previously existing input files should continue to work as before, but there have been some changes. A number of deprecated obsolete features, some of which have not been documented for a long time, have been removed. There are also some extensions and simplifications.

• There is no longer the need to use **barcount** for movements longer than 500 bars, and some other limits have been removed or relaxed.

• Extra fonts are now accessed in strings by, for example, \xx2\ rather than \x2\, which is ambiguous; it could be a hexadecimal character number. The old notation is, for the moment, still interpreted as a font change, but gives a warning.

• The synonym **-includefont** for **-includePMWfont** is removed.

• The synonym **origin** for **originx** in draw functions is removed.

• Earlier versions of PMW used a more restricted set of options starting with /s to control split slurs. These are no longer supported.

• All the deprecated directives whose names began with 'old' have been removed.

• An obsolete feature of irregular note groups, allowing a hyphen after the count of regular notes (e.g. {3-/11) is removed.

• Support for including literal PostScript has not been retained. It could be put back should anyone actually need it.

• A list of numbers that follows some directives must now be all on one line. Such lists can no longer be automatically continued onto the next line by terminating the first line with a comma. Instead, use the &&& facility to concatenate input lines.

• If a character that is treated specially in a string is required as a literal, it must be escaped with a backslash. It no longer works to include it as a character number, because escape sequences are now processed when a string is read, before any interpretation.

• Similarly, a redundant font change no longer works to disable a kern. You have to use some actual characters that have no effect on the output, for example *u*d\ (which does an up and down move in the music font).

• White space is permitted before a beam break character.

• The **accspacing** directive can now be given with an optional sixth value, giving the width for the narrower style 0 half sharp.

• The **timefont** directive must precede **printtime** if it is to apply to the strings in **printtime**.

• A completely empty stave now starts with a clef unless there was a previous **[noclef]** setting.

• A masquerade setting for a chord must appear on the chord's first note.

• The test output files in the *testdist/outfiles* directory are no longer complete PostScript files that include the music font. However, they can be viewed by a command such as:

```
cat PSheader testdist/outfiles/Test01.ps | gv -
```

though this method does not show a list of page numbers.

• The separate font name bigmusic is abolished.

• **[omitempty]** is no longer a separate directive, but is instead an option of the **[stave]** directive, because it applies to the entire stave.

• **[Stavelines]** also applies to the whole stave; it still works, but is now deprecated, with a warning message. Instead of **[stavelines 3]** (for example), you should now use **[stave n/3]**.

• Slur and line ids are now required to be ASCII alphanumeric characters.

- The obsolete directives whose names began with 'play' have been removed (replace 'play' with 'midi').

- The obsolete **[percussion]** directive is removed.

- The [move] directive can now move rests vertically as well as horizontally.

- A Unicode translation (`.utr`) file can now specify the character to use for unsupported code points in non-standardly-encoded fonts.

15. Characters in text fonts

Text fonts such as *Times-Roman* that use the Adobe standard encoding contain over 300 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 8.17.4. This chapter lists all the characters in the standard text fonts, with their Unicode values (in hexadecimal, as is conventional), their PMW escape sequences when defined, and their character names. However, some of these characters may be missing in older fonts.

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.

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 appear as opening and closing quotes, respectively. If you want an actual grave accent or an ASCII single quote character, you can use the escape sequences \` and \'.

Unicode	Escape	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	»
00BC		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

Characters in text fonts (15)

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
0108	\C^	Ccircumflex	Ĉ
0109	\c^	ccircumflex	ĉ
010A		Cdotaccent	Ċ
010B		cdotaccent	ċ
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
0114	\Eu	Ebreve	Ĕ
0115	\eu	ebreve	ĕ
0116		Edotaccent	Ė
0117		edotaccent	ė
0118		Eognonek	Ę
0119		eogonek	ę
011A	\Ev	Ecaron	Ě
011B	\ev	ecaron	ě
011C	\G^	Gcircumflex	Ĝ
011D	\g^	gcircumflex	ĝ
011E	\Gu	Gbreve	Ğ
011F	\gu	gbreve	ğ
0120		Gdotaccent	Ġ
0121		gdotaccent	ġ
0122		Gcommaaccent	Ģ
0123		gcommaaccent	ģ
0124	\H^	Hcircumflex	Ĥ
0125	\h^	hcircumflex	ĥ
0126		Hbar	Ħ
0127		hbar	ħ
0128	\I~	Itilde	Ĩ

0129	\i~	itilde	ĩ
012A	\I-	Imacron	Ī
012B	\i-	imacron	ī
012C	\Iu	Ibreve	Ĭ
012D	\iu	ibreve	ĭ
012E		Iogonek	Į
012F		iogonek	į
0130		Idotaccent	İ
0131		dotlessi	ı
0132		IJ	IJ
0133		ij	ij
0134	\J^	Jcircumflex	Ĵ
0135	\j^	jcircumflex	ĵ
0136		Kcommaaccent	Ķ
0137		kcommaaccent	ķ
0138		kgreenlandic	κ
0139	\L'	Lacute	Ĺ
013A	\l'	lacute	ĺ
013B		Lcommaaccent	Ļ
013C		lcommaaccent	ļ
013D	\Lv	Lcaron	Ľ
013E	\lv	lcaron	ľ
013F		Ldot	Ŀ
0140		ldot	ŀ
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
0149		napostrophe	ʼn
014A		Eng	Ŋ
014B		eng	ŋ
014C	\O-	Omacron	Ō
014D	\o-	omacron	ō
014E	\Ou	Obreve	Ŏ
014F	\ou	obreve	ŏ
0150	\O"	Ohungrumlaut	Ő
0151	\o"	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

Characters in text fonts (15)

015B	\s'	sacute	ś
015C	\S^	Scircumflex	Ŝ
015D	\s^	scircumflex	ŝ
015E	\S,	Scedilla	Ş
015F	\s,	scedilla	ş
0160	\Sv	Scaron	Š
0161	\sv	scaron	š
0162	\T,	Tcedilla	Ţ
0163	\t,	tcedilla	ţ
0164	\Tv	Tcaron	Ť
0165	\tv	tcaron	ť
0166		Tbar	Ŧ
0167		tbar	ŧ
0168	\U~	Utilde	Ũ
0169	\u~	utilde	ũ
016A	\U-	Umacron	Ū
016B	\u-	umacron	ū
016C	\Uu	Ubreve	Ŭ
016D	\uu	ubreve	ŭ
016E	\Uo	Uring	Ů
016F	\uo	uring	ů
0170	\U"	Uhungrumlaut	Ű
0171	\u"	uhungrumlaut	ű
0172		Uogonek	Ų
0173		uogonek	ų
0174	\W^	Wcircumflex	Ŵ
0175	\w^	wcircumflex	ŵ
0176	\Y^	Ycircumflex	Ŷ
0177	\y^	ycircumflex	ŷ
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
017F		longs	ſ
0192		florin	ƒ
0218		Scommaaccent	Ș
0219		scommaaccent	ș
021A		Tcommaaccent	Ț
021B		tcommaaccent	ț
0302		circumflex	^
0303		tilde	~
0306		breve	˘
0307		dotaccent	·
030A		ring	°
030B		hungrumlaut	˝
030C		caron	ˇ

Characters in text fonts (15)

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	√
221E		infinity	∞
2260		notequal	≠
2264		lessequal	≤
2265		greaterequal	≥
25CA		lozenge	◊
FB01	\fi	fi	fi
FB02	\fl	fl	fl

16. The PMW music font

This chapter contains a list of all the characters in the PMW-Music font. Characters from this font can be referenced by number in character strings (☞ 8.17.12). Those with character codes less than 127 can also be referenced by switching to the music font and entering the corresponding ASCII character or giving the Unicode code point if there is one. The following three examples produce the same effect:

```
"\rm\this clef \*33\ is treble"
"\rm\this clef \mu\!\rm\ is treble"
"\rm\this clef \mu\\x1D11E\\rm\ is treble"
```

The second method is more convenient when a whole sequence of music font characters is required. Character 33 in the music font (which corresponds to an exclamation mark in ASCII) is the treble clef.

Most of the characters in the music font are positioned '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 may or may not be used by PMW when setting music, are mostly set to values that are reasonable when these characters are 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 and Unicode code points. The character width is also given as a fraction of the font size. For example, when a 10-point treble clef is output, its width is 15 points.

ASCII	Code	Unicode	Width	Char	Comment
	32	00020	0.75		space
!	33	1D11E	1.5		treble clef
"	34	1D122	1.5		bass clef
#	35	1D121	1.5		alto clef
$	36	1D1AF	1.0		piano end pedal sign
%	37	0266F	0.6		sharp
&	38	1D12A	0.6		double sharp
'	39	0266D	0.5		flat
(	40	0266E	0.45		natural
)	41	1D110	0.0		fermata (over)
*	42	1D13A	0.66		breve rest
+	43	1D13B	0.66		semibreve rest
,	44	1D13C	0.66		minim rest
−	45	1D13D	0.66		crotchet rest
.	46	1D13E	0.59		quaver rest
/	47	1D111	0.0		fermata (under)
0	48	1D129	3.5		many bars rest
1	49	1D15C	1.34		breve
2	50	1D15D	0.84		semibreve
3	51	1D15E	0.84		up minim
4	52		0.84		down minim
5	53	1D15F	0.84		up crotchet
6	54		0.84		down crotchet
7	55	1D160	1.2		up quaver

8	56		0.84	♪	down quaver
9	57	1D161	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	1D100	0.6	|	bar line
A	65	1D101	0.76	‖	double bar line
B	66		0.76	▌	thick bar line
C	67	1D11A	1.0	≣	normal stave
D	68	1D116	1.0	—	percussion stave
E	69	1D16E	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	1D165	0.0	|	upward note stem
K	75		0.0	|	downward note stem
L	76	1D158	0.84	●	solid notehead
M	77	1D157	0.84	o	minim notehead
N	78	1D112	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	1D197	0.0	∞	turn
T	84	1D170	0.0	‾	horizontal bar accent
U	85	1D181	0.0	>	accent
V	86	1D113	1.0	//	caesura
W	87	1D19C	0.0	∧	accent
X	88		0.0	∨	accent
Y	89		0.0	'	accent
Z	90	1D17E	0.0	‚	accent
[	91	1D104	0.6	┊	dashed bar line
\	92		1.0	/	single-line caesura
]	93		0.0	8	for use with clefs
^	94	1D134	1.0	𝄴	'common' time
_	95	1D135	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	1D10C	1.5	𝄊	dal segno
d	100	1D10B	1.5	𝄋	dal segno
e	101	1D1AA	0.0	⊓	down bow

f	102		0.0	⊔	inverted down bow
g	103	1D1AB	0.0	V	up bow
h	104		0.0	∧	inverted up bow
i	105	1D199	0.0	∞	inverted turn
j	106		0.55	7	for figured bass
k	107		0.76	4	for figured bass
l	108	1D1BA	0.84	◆	solid diamond notehead
m	109	1D1B9	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	6	for figured bass
t	116		0.55	·	dot for guitar grid
u	117		0.55	○	circle for guitar grid
v	118		0.0		outputs nothing; moves down by 0.1
w	119		0.0		outputs nothing; moves down by 0.4
x	120		0.0		outputs nothing; moves up by 0.4
y	121		-0.1		outputs nothing; moves left by 0.1
z	122		0.1		outputs nothing; moves right by 0.1
{	123		-0.33		outputs nothing; moves left by 0.33
|	124		0.0		outputs nothing; moves down by 0.2
}	125		0.55		outputs nothing; moves right by 0.55
~	126		0.0		outputs nothing; moves up 0.2
	127		-		unassigned
	128		0.6	√	tick
	129		0.0	╱	accaciatura bar
	130		0.0	╲	accaciatura bar
	131	1D11C	0.0	▦	grid for guitar chords
	132	1D1A0	0.6	I	short bar line
	133		0.0	╠	breath
	134		0.0	○	ring above
	135		0.0	+	cross
	136	1D196	0.8	tr	trill
	137		0.6	ı	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	1D10D	0.5	╱	for bar repetition
	144		0.0	··	for bar repetition
	145		0.0	⸴	for arpeggios – moves upwards by 0.4
	146	1D167	0.0	⁓	tremolo bar – moves upwards by 0.4
	147	1D1C8	1.0	○	old time signature

148	1D1C9	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	Ɔ	inverted 'common' time
154		1.0	Ɖ	inverted 'cut' time
155		1.58	‖∞‖	unison breve
156		0.0	'	'start of bar' accent
157		0.35	⟨	for bracketing *8*
158		0.35	⟩	for bracketing *8*
159		0.33	˥	
160		0.33	˞	for *8va* lines etc.
161		0.33	˩	
162		0.33	˪	
163	1D1AE	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	1D117	1.0	⎓	2-line stave
170	1D118	1.0	≡	3-line stave
171	1D119	1.0	≣	4-line stave
172	1D11B	1.0	≣	6-line stave
173	1D125	1.5	▮▮	percussion clef
174		1.5	℗:	old-style F clef
175		1.5	𝕂	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		outputs nothing; moves left 0.42, up 0.4
186		-0.76		outputs nothing; moves left 0.76, down 0.4
187		0.0		outputs nothing; moves up 1.2
188		0.0		outputs 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	ꝺ	half flat, Turkish style
193		0.6	'	pause comma, inverted for R-to-L music

194		0.0	'	staccatissimo
195		0.0	ı	staccatissimo, inverted
196	1D198	0.0	∽	reversed turn
197		0.0	∾	inverted reversed turn
198		0.0	⌒	top half circle
199		0.0	⌣	bottom half circle
200		0.84	•	circular crotchet notehead
201		0.84	○	circular minim notehead
202		0.0	‹	reversed accent
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 32, 118–126, and 185–188 do not cause anything to be output; instead they just cause the current position to be moved by a distance that depends on the point size of the font.

Note: All these characters exist as actual characters in the original PostScript version of the font, which is in a file called *PMW-Music.pfa*. From release 5.20 an OpenType version called *PMW-Music.otf* is also provided because the PostScript font formats are becoming obsolete.
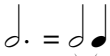
The OpenType format does not support leftwards, upwards, or downwards movements in fonts whose primary movement is to the right. For this reason, the PMW program has been updated so that it no longer relies on the font mechanism for such movements. Instead, when it encounters any of these characters, it emulates what would happen with the original PostScript font. This means that PMW output works with either form of the font, but if the OpenType font is used with any other program, the characters with unsupported movements are not available.

The values given in the table 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 position down by 4 points. If a space character (number 32) is output from the music font, it moves the position by 0.75 units to the right.

There is a discussion on the use of the special characters for guitar chord grids in section 8.17.13. The larger round and square brackets (characters 139–142) are designed so that they can be output 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 original time signatures (see the example in section 10.1.132). 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 showing 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 placed before the notes.

```
"\**m.\ = \mf\\149\\51\\53\"
```

This example is output as:

17. 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 music symbols that are useful in conjunction with normal alphabetic text. The font was originally designed as an Acorn RISC OS font; a PostScript Type 3 version was generated automatically from the original and was distributed with PMW up to and including version 5.30. Later releases of PMW include a Type 1 version of this font, created manually from the Type 3 version, together with an automatically generated OpenType version. This means it can be used with both PostScript and PDF output. 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 for 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.

17.1 Use of PMW-Alpha from within PMW

Here is an example of some header 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    "\xx1\f"
*Define fr37 \xx1\\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 use this font. The first ***define** directive defines a macro for the *forte* mark f. The second macro is for the fraction ⅔. 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\"
```

17.2 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 �γ 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.''

17.3 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	⅞	small semiquaver rest
#	35	0.46	♯	
$	36	0.38	♭	
%	37	0.36	♮	
&	38	0.7	♭♭	
'	39	0.38	⅞	big semiquaver rest
(	40	0.924	←	
)	41	0.924	→	
*	42	0.77	✳	
+	43	0.64	+	for time signatures
,	44	0.26		'double dot' space
.	46	0.13	.	dot for upstemmed notes

/	47	0.38	⸮	big crotchet rest
0	48	0.64	**0**	
1	49	0.64	**1**	
2	50	0.64	**2**	
3	51	0.64	**3**	
4	52	0.64	**4**	
5	53	0.64	**5**	
6	54	0.64	**6**	
7	55	0.64	**7**	
8	56	0.64	**8**	
9	57	0.64	**9**	
:	58	0.26	⸓	small quaver rest
;	59	0.38	⸓	big quaver rest
<	60	0.13		'dot' space
=	61	0.564	=	
>	62	0.13	·	dot for downstemmed notes
?	63	0.26	⸮	small crotchet rest
B	66	0.64	𝄢	
C	67	0.64	𝄴	
E	69	0.38	⌐	
F	70	0.80	⌢	
G	71	0.64	𝄞	
H	72	0.00	▬	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	▣	'downstem' breve
R	82	0.38	⌐	
S	83	0.72	𝄪	
T	84	0.38	𝄋	
V	86	0.64	𝄡	
W	87	0.38	∘	'downstem' 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	*f*	

g	103	0.64	$\oint$	
h	104	0.00	—	
j	106	0.00	=	beams for upstemmed notes
k	107	0.00	≡	
m	109	0.88	*m*	
o	111	0.76	*tr*	
p	112	0.69	*p*	
q	113	0.64	○	'upstem' breve
r	114	0.38	↓	
s	115	0.32	*s*	
t	116	0.38	♪	
u	117	0.817	↗	
w	119	0.38	○	'upstem' semibreve
y	121	0.38	♪	
z	122	0.42	*z*	
{	123	0.26	▪	small semibreve rest
|	124	0.02	|	bar line
}	125	0.26	▪	small minim rest
~	126	0.70	∾	
	160	0.00	≣	stave segment
	178	0.75	⅔	
	179	0.75	⅗	
	180	0.75	⅕	
	181	0.75	⅖	
	182	0.75	⅘	
	183	0.75	⅐	
	184	0.75	2/7	
	185	0.75	⅓	
	186	0.75	⅑	
	187	0.75	8/9	
	188	0.75	¼	
	189	0.75	½	
	190	0.75	¾	
	200	0.30	0	
	201	0.30	1	
	202	0.30	2	
	203	0.30	3	
	204	0.30	4	for fraction numerators
	205	0.30	5	
	206	0.30	6	
	207	0.30	7	
	208	0.30	8	
	209	0.30	9	
	210	0.30	0	
	211	0.30	1	
	212	0.30	2	

213	0.30	3	for fraction denominators
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

18. Syntax summary

18.1 Input line concatenation

A line that ends with `&&&` is concatenated with its successor (with `&&&` and the newline removed) before any other processing.

18.2 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 <macro name> <rest of line>
*else
*fi
*if <condition>
*if not <condition>
*include "file name"
```

A macro name must be followed by parentheses, with optional default argument values, if it is to be used with arguments.

18.3 Macro calls

```
&<macro name><optional semicolon>
&<macro name>(<arguments>)
```

Macro calls may occur in macro arguments to any depth. Use `&)` or `&(` to include unmatched parentheses in a macro argument, and `&,` for a comma at top level (not needed if in matched parentheses or quotes).

18.4 Input string repetition

The argument of a string repetition is processed as a macro argument.

```
&*<number>(<argument>)
```

18.5 Header 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>
 *B2pffont <fontword> <options> "<rules names>"
  Bar <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>
 Codemultirests
 Copyzero <n>/<m> ...
 Cuegracesize <n>
 Cuesize <n>
 Dotspacefactor <n>
†Doublenotes
 Draw <drawing definition> enddraw
*Drawbarlines
*Drawstavelines <n> (optional)
 Endlinesluradjust <n>
 Endlineslurstyle <n>
 Endlinetieadjust <n>
 Endlinetiestyle <n>
*Eps
 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>
*IncPMWfont aka IncludePMWfont
 Join <n>-<m> ...
 Joindotted <n>-<m> ...
 Justify <edges>
†Key <key signature>
 Keydoublebar
 Keysinglebar
 Keytranspose <key signature> <details>
 Keywarn
*Landscape
 Lastfooting <fontsize> "<string>" <space>
 Lastfooting draw <name> <space>
†Layout <n1> <n2> ...
 Ledgerstyle <n>
 Leftmargin <n>
 Linelength <n>
 Longrestfont <fontsize> <font>
```

```
*Magnification <n>
 Makekey X<n> <key definition>
 Maxbeamslope <n>
*Maxvertjustify <n>
 Midichannel <n> "<name or number>" <staves>
 Midifornotesoff
 Midistart <n> <n> <n> ...
 Miditempo <n> <n>/<m> ...
 Miditranspose <n>/<m> ...
 Miditremolo <type>
 Midivolume <n> <n>/<m> ...
 Midkeyspacing <n>
 Midtimespacing <n>
*Musicfont "<font name>"
 Nobeamendrests
 Nocheck
 Nocheckdoublebars
 Nocodemultirests
*Nokerning
 Nokeywarn
 Nosluroverwarnings
 Nospreadunderlay
 Notespacing *<factor>
 Notespacing <n1> ... <n8>
†Notime
 Notimebase
 Notimewarn
 Nounderlayextenders
*Nowidechars
*Output <type>
 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>
 Pmwversion <n>
 Printkey <key> <clef> "<string 1>" "<string 2>"
 Printtime <time> "<string 1>" "<string 2>"
 Rehearsalmarks <align> <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>
  Systemseparator <length> <width> <angle> <x-adjust> <y-adjust>
*Textfont <fontword> "<font name>"
*Textfont <fontword> include "<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>
  Tupletfont <fontsize> <name>
  Tupletlinewidth <n>
  Underlaydepth <n>
  Underlayextenders
  Underlaysize <fontsize>
  Underlaystyle <n>
†Unfinished
  Vertaccsize <n>
```

18.6 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 and many are not relevant for rests. An upper case letter sets the initial note
length of a minim, and a lower case letter is a crotchet, with +, −, =, and ! characters available to
adjust the length. Except for !, which may only follow an upper case letter, these can be freely
intermixed.

accidental	# ## $ $$ %
half sharp	#-
half flat	$-
invisible	?
over note	o
under note	u
transposed	^# ^## ^$ ^$$ ^% ^- ^+
bracketed	)]
moved	< or <*number*
note or rest letter	a-g A-G q-t Q-T
octave change	' to raise, ` to lower
note length flags	− = =− == divide length by 2, 4, 8, or 16
	+ ++ multiply length by 2 or 4
	! note length is bar length
move dot	> or *number*>
dot(s)	. or .. or .+
expression/options	*expression and options indications*\\
tie or short slur	_

above/below	`/a /b`
editorial	`/e`
dashed/dotted	`/i /ip`
glissando	`_/g`
full beam break	`;`
partial beam break	`,`

There are two short cuts for note entry:

exact copy	x or x*<digit>*
pitch only copy	p or P

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
`)`	notehead in round brackets
`a`*<n>*	accent *<n>*
`ar`	arpeggio
`aru`	arpeggio with up arrow
`ard`	arpeggio with down arrow
`c`	output on coupled stave
`C`	flip centring if only note/rest 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 output on coupled stave
`l`*<n>*	rest level
`m`*<flags>*	masquerade
`nc`	circular notehead
`nd`	'direct' notehead
`nh`	harmonic (diamond) notehead
`nn`	normal notehead
`no`	notehead only, no stem
`nx`	cross notehead
`nz`	no notehead
`o`	indicate harmonic with small circle
`rt`	inverted turn (reversed)
`rt|`	reversed turn with bar
`sd`	stem down
`sl`*<n>*	extend stem length
`sl-`*<n>*	shorten stem length
`sm`	use small notehead
`sp`	spread chord
`su`	stem up

Syntax summary (18)

sw	swap stem direction in beam
t	turn
t|	inverted turn (with bar)
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 except tremolos 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. Tremolos themselves can be moved up and down, but not left or right. The notation looks confusing, but is consistent: for example, G\//u4\ specifies a single tremolo that is moved up by 4 points.

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

18.7 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 irregular note group
}	end irregular note group
//	caesura

>	diminuendo hairpin
<	crescendo hairpin

18.8 Hairpin adjustment options

The /a, /b, /m and /w options may appear only at the start of a hairpin. The other options may appear at either end. If vertical movement is specified at the start of a hairpin, it applied to the whole hairpin. If specified at the end, it affects only the end.

/a<n>	position above the stave (dimension optional)
/b<n>	position below the stave (dimension optional)
/bar	position at previous bar line
/d<n>	move down <n> points
/h	position halfway between notes
/l<n>	move left <n> points
/lc<n>	position left <n> crotchets
/m	position in midway between two staves
/r<n>	move right <n> points
/rc<n>	position right <n> crotchets
/slu<n>	split left up
/sld<n>	split left down
/sru<n>	split right up
/srd<n>	split right down
/u<n>	move up <n> points
/w<n>	set width of open end

18.9 Stave text item options

These options may occur after any text item other than stave names, but for a rehearsal mark inside square brackets only those specifying movement are relevant, and /bar, /c, /cb, /e, /nc, /ne, and /ts are ignored on underlay and overlay strings. Other restrictions apply to follow-on strings (☞ 11.9.5).

/a	position above the stave
/ao	position above, at the overlay level
/a<n>	position <n> points above the stave
/b	position below the stave
/bar	position at previous bar line
/box	enclose in a rectangular box (mitred)
/bu	position below, at the underlay level
/b<n>	position <n> points below the stave
/c	centre the string horizontally at its position
/cb	centre the string horizontally in the bar
/d<n>	move down <n> points
/e	align end of string, not start
/F	this is a follow-on string
/fb	this is figured bass
/fbu	this is figured bass, at underlay level
/h	position halfway between notes
/l<n>	move left <n> points
/lc<n>	position left <n> crotchets
/m	position below, halfway to the next stave
/nc	cancel a previous /c
/ne	cancel a previous /e
/ol	string is overlay
/r<n>	move right <n> points
/rbox	enclose in a rectangular box (rounded)
/rc<n>	position right <n> crotchets
/ring	enclose in a ring shape

/rot*<n>*	rotate *<n>* degrees
/s*<n>*	use settable font size *<n>* (1–20)
/S*<n>*	use fixed font size *<n>* (1–10)
/ts	position at time signature
/ul	string is underlay
/u*<n>*	move up *<n>* points

Settable font sizes are defined in the header by the **textsizes** directive. For underlay and overlay strings, additional strings may follow as options, to control the hyphens between syllables (☞ 11.12.5).

18.10 Stave name string options

These are the only options that are recognized for strings given in the **[name]** or **[stave]** directives.

/c	centre align
/d*<n>*	move down *<n>* points
/e	right justify
/l*<n>*	move left *<n>* points
/m	position below, halfway to the next stave
/r*<n>*	move right *<n>* points
/s*<n>*	use settable font size *<n>* (1–20)
/S*<n>*	use fixed font size *<n>* (1–10)
/u*<n>*	move up *<n>* points
/v	rotate to run vertically
/"...	add another string at the same point

18.11 Character string escapes

The escape sequences that specify accented and other special characters in text fonts are shown in the character list in chapter 15. What the remaining escape sequences insert is summarized here.

\c]	© from the Symbol font
\p\	page number
\pe\	page number, if even
\po\	page number, if odd
\se\	skip to next \se\ if page number even
\so\	skip to next \so\ if page number odd
\r\	repeated bar number
\r2\	repeated bar number, except the first bar
\t\	transposition (number of semitones)
\t*x*	transpose chord name *x* (one of A–G, optionally followed by # or $)
\@	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 for 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

Syntax summary (18)

`\*<\`	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.

`\<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
`\xx1\`	change to the first extra font
...	
`\xx12\`	change to the twelfth extra font

Extra fonts are defined in the header by the **textfont** directive.

18.12 Underlay strings

These characters are treated specially in underlay strings:

–	end of syllable in mid-word
=	continue syllable over additional note
#	convert to space; doesn't terminate a word
^	centre only characters to the left of this or between two ^ characters

18.13 Bracketed stave directives

These directives occur in square brackets interspersed in among the notes and rests:

`[\!\]`	repeat accent movement
`[\.\]`	repeat staccato
`[\..\]`	repeat staccatissimo
`[\-\]`	repeat accent
`[\>\]`	repeat horizontal wedge accent
`[\v\]`	repeat small vertical wedge accent
`[\V\]`	repeat large vertical wedge accent
`[\'\]`	repeat 'start of bar' accent
`[\d\]`	repeat string down bow
`[\u\]`	repeat string up bow
`[\o\]`	repeat harmonic ring
`[\a<n>\]`	repeat accent *<n>*
`[\/\]`	repeat single tremolo
`[\//\]`	repeat double tremolo
`[\///\]`	repeat 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
[backup]	backup to previous note in bar
[baritone <*octave*>]	select baritone clef
[barlinestyle <*n*>]	select bar line style for stave
[barnumber</*options*>]	explicit bar number
[barnumber off]	suppress bar number
[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
[c]	synonym for [noteheads circular]
[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
[doublenotes]	double note lengths
[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; see **heading** for extended arguments
[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
[halvenotes]	halve note lengths
[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
[miditremolo <type>]        change MIDI tremolo handling
[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 notes
[notes off]                 turn off notes
[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
[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>
[printpitch <note>]         force display pitch
[reset]                     reset position to bar start
[resume]                    resume stave
[rlevel <n>]                set rest level
[rmove ...]                 as [move] but scale horizontally
[rsmove ...]                as [smove] but scale horizontally
[rspace <n>]                as [space] but scale horizontally
[sgabove <n>]               set system gap for previous system
[sghere <n>]                set system gap for this system
[sgnext <n>]                set system gap for next system
[skip <n>]                  skip <n> bars
[sl]                        synonym for [stemlength]
[slur/<options>]            start long slur
[slurgap/<options>]         leave gap in slur
[smove ...]                 combined [move] and [space]
[soprabass <octave>]        select soprabass clef
[soprano <octave>]          select soprano clef
[space <n>]                 insert space before next note
[ssabove <n>]               set ensured stave spacing for this system
[sshere <n>]                set stave spacing for this system
[ssnext <n>]                set stave spacing for next system
[stave <n> ...]             start new stave
[stemlength <n>]            set default stemlengh adjustment
[stems <direction>]         force/unforce stem direction
[stems <alignment>]         set/unset centralized stems
[suspend]                   suspend stave at next system
[tenor <octave>]            select tenor clef
```

Syntax summary (18)

```
[text <name>]                set default text type
[text above <n>]             set default text above at fixed level
[text below <n>]             set default text below at fixed level
[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]        show cautionary accidentals
[transposedacc noforce]      don't show cautionary accidentals
[treble <octave>]            select treble clef
[trebledescant <octave>]     select trebledescant clef
[trebletenor <octave>]       select trebletenor clef
[trebletenorB <octave>]      select trebletenorB clef
[tremolo]                    tremolo between notes
[tripletize off]             disable tripletizing
[tripletize on]              treat certain groups as triplets ('on' is optional)
[triplets off]               don't show triplet indications
[triplets on]                show triplet indications
[triplets <options>]         control triplet format
[tuplets off]                don't show tuplet indications
[tuplets on]                 show tuplet indications
[tuplets <options>]          control tuplet format
[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]
```

18.14 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
/llc<n>       move the left end left by <n> crotchets
/lrc<n>       move the left end right by <n> crotchets
/rlc<n>       move the right end left by <n> crotchets
```

`/rrc<n>`	move the right end right by *<n>* crotchets
`/cx`	`/rlc` and `/rrc` are relative to next note
`/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'

18.15 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
Note spacing	30 30 22 16 12 10 10 10
Note stem direction	automatically chosen
Note style	with stems
Output format	chosen when PMW is built
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	shown
Time signature warnings	enabled
Top margin	10
Trill string	*tr*
Tuplet font	roman
Tuplet size	10 points
Transposition	none
Underlay depth	11 points
Underlay size	10 points

Syntax summary (18)

A. Reading MusicXML files

The ability to read MusicXML input in addition to PMW's own input format was added in release 5.10. However, this is highly experimental and is currently very incomplete (and probably quite buggy). Please do not expect great things of it.

There is a particular problem with MusicXML parts that use two or more staves. PMW can mostly handle a treble/bass clef pair if the notes are straightforward, but MusicXML examples exist which do weird things such as ending a slur on one stave before starting it on another after 'backing up'. This particular case causes PMW to give an error, but in other situations the result is just crazy output.

The MusicXML-reading code is an optional part of PMW. To use it you must add `--enable-musicxml` when running `./configure` before building PMW. When this is not done, MusicXML files are still recognized as such, but cannot be processed and so cause a hard error. There are two situations in which MusicXML files are recognized:

- If the first line of PMW's input file begins with `<?xml version=` the file is assumed to be MusicXML, and processed accordingly.

- If, within a PMW input file, a file specified by the ***include** preprocessing directive (☞ 8.2.6) begins with `<?xml version=` it is recognized as MusicXML. As long as it is not within a PMW stave, it is processed, and control then returns to process more PMW input. If it is within a PMW stave, an error occurs.

The second case makes it possible to change some of PMW's behaviour by including header directives before including a MusicXML file. For example, to enlarge the music:

```
magnification 1.1
*include "file.musicxml"
```

Not all directives are effective, and some that are may be overridden by the contents of the MusicXML file. In particular:

- Key and time signature settings are usually overridden.

- Added headings are positioned above MusicXML headings.

- Added footings are positioned below MusicXML footings.

- No transposition is applied to the MusicXML notes.

Some exotic effects are possible by mixing PMW staves with MusicXML files. The full extent has not been explored.

B. Writing MusicXML files

MusicXML is a music encoding that has become popular for exchanging music information between different applications. It uses the standard XML syntax to encode the music, and contains performance as well as notational information. There is a lot of mismatch between the ways in which PMW and MusicXML encode music. For example, dynamic markings such as *mf* and *pp* are handled as text strings in PMW whereas MusicXML has dedicated elements for dynamics.

The ability to write MusicXML output in addition to PostScript, PDF, and MIDI was added to PMW in release 5.33. However, this is experimental and currently incomplete (and probably quite buggy). Please do not expect great things of it.

MusicXML output is requested by means of the **-musicxml** or **-xml** command line option, which must supply a file name. This output is in addition to the normal PostScript or PDF output. A MusicXML file can contain only a single movement, but a PMW input file may contain many. This mismatch has been addressed in two ways:

- The **-musicxmlmovement** or **-xm** command line option can be used to select a specific movement.

- If no movement is selected from a multi-movement PMW file, and the file name given to the **-musicxml** option contains the character sequence %d with optional digits between the two characters, then each movement is written as MusicXML to a separate file, with the movement's number replacing the %d sequence, which is used as a format instruction.

When using the second strategy, a number included in the %d sequence defines the minimum number of digits that are to be inserted. It should normally be preceded by a zero, which requests leading zeros, as otherwise you might end up with file names that contain spaces. This example will create files called *movement-01.xml*, *movement-02.xml*, etc.

```
pmw -xml movement-%02d.xml mysymphony.pmw
```

If the file name does not contain %d when no movement is selected, the first movement is output.

It will never be possible to convert everything that might be in a PMW file into MusicXML. An obvious example is the use of the drawing facility. After importing a piece that PMW has exported, it will almost always be necessary to make some modifications from within the importing program.

Some 'difficult' PMW features that are currently missing may be partly or wholly added later as time permits. MusicXML coding for absolute and relative positioning of items is often ignored by programs that process it. They tend to lay out the musical content using their own algorithms. For that reason, many of PMW's positioning instructions (e.g. the **[space]** directive]) are currently ignored when generating MusicXML output. There are also features that are straightforward to translate, but which have just been overlooked. I will add these if asked or if I happen to notice.

Some features that are translatable may or may not be wanted in MusicXML output. For example, explicit formatting instructions such as **[newpage]** may not be wanted when another typesetting program is used. For this reason, there are some options for controlling the inclusion of some items. These options are given as a list of words, separated by plus signs, following the **-x** command line option, without a space. For example:

```
pmw -xml export.xml -x+newline+newpage something.pmw
```

The options are as follows:

newline
   The **[newline]** directive causes PMW to start a new system in its PostScript or PDF output. It is ignored when generating MusicXML unless this option is set.

newpage
   The **[newpage]** directive causes PMW to start a new page in its PostScript or PDF output. It is ignored when generating MusicXML unless this option is set.

numberlyrics

When there is more than one underlay verse in a vocal part, PMW will output a number for each `<lyric>` item when generating MusicXML. By default, it does not do this if there is only one verse. There is at least one version of one processor that completely ignores unnumbered lyrics, so this option is provided to ensure that even a single verse is numbered '1'. However, this may result in the addition of the text '1.' at the start.

suspend

The **suspend** and **[suspend]** directives can cause staves to be omitted from systems in PostScript or PDF output when all a stave's bars are empty or are rest bars. These directives are ignored for MusicXML output unless this option is set. When it is, the stave bars that were omitted in the PostScript or PDF output are flagged as non-printing, but because another processor may put different numbers of bars into its systems, the effect may not be ideal.

B.1 Specific issues

Here are a number of specific issues relating to MusicXML output.

• If a transposition is set, it is the transposed music that is output. The `\t` string escape sequence works.

• Settings for stave spacings do not seem to be much honoured. In particular, a stave spacing of zero probably will not carry over successfully.

• Stave 0 is completely ignored.

• MusicXML (release 4.0) does not have a way of specifying a footnote that uses more than one font. Changes of font within a footnote are therefore ignored.

• If a PMW string contains just the character 99 or just the character 100 in the music font, it is translated into a MusicXML `<segno>` element.

• If the setting `[bowing below]` is used, upbow and downbow 'accents' are translated into into MusicXML's `<toe>` and `<heel>` elements.

• Strings that are marked as figured bass are translated into MusicXML's figured bass elements, but there may be issues with some of the more complicated ones.

Index

Windows, running PMW under 3

X
[xline] 150
[xslur] 163