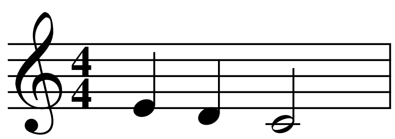
Mup
Music Publisher
User's Guide



Mup Version 7.2

Mup Music Publisher User's Guide — Mup Version 7.2

CONTENTS

Mup — Music Publisher
User's Guide

1. INTRODUCTION

The music publisher program called "Mup" takes a text file describing music as input, and generates PostScript* output for printing that music. After being available as shareware for 17 years, as of version 6.1, it is now available as free, open-source software. The input file can be created using your favorite text editor, or with the help of the companion Mupmate program, or generated from any other source, such as another program. The input must be written in a special language designed especially for describing music. The majority of the Mup User's Guide is the explanation of this language and how to use it.

Mup has the power to print almost any kind of music, everything from a single melody line to full orchestral or choral scores complete with tempo and dynamic marks. In addition to standard 5-line staffs, it can handle 1-line staffs (typically used for percussion), and tablature notation (typically used for guitar). Because Mup can do so much, it takes a while to master its entire language. However, it has built-in default values for many things, so that you can start using it for simple songs after learning just the basics, then learn the more complicated features as you need them. Mup also has an option to produce output in the standard Musical Instrument Digital Interface (MIDI) format.

The User's Guide begins by introducing the basics and describing the general framework of the language. Then it gives detailed information about all the features of Mup. Appendix A gives a sample input file. There is a Quick Reference available that may be useful for jogging your memory after you've had a little experience using Mup.

This User's Guide is for Mup version 7.2.

---

* PostScript is a trademark of Adobe Systems Incorporated

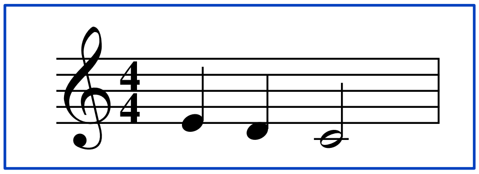
2. MUP BASICS

This section introduces the Mup language, giving some simple examples to give you the flavor of a Mup input file. Subsequent sections will go into greater detail.

2.1 Notes and chords

Music is described one measure at a time. Each note is specified by its pitch value, "a" to "g." As an example, the first measure of "Three Blind Mice" can be described like this:

```
1:    4e; 4d; 2c;
```



The "1:" at the beginning of the line tells Mup that we are describing the notes on staff number 1. In this very simple example, we only have one staff, but later we'll do songs with more than one. Each staff of each measure is normally put on a separate line.

The first three notes of "Three Blind Mice" are E, D, and C. For Mup input, these pitches are given in lowercase to avoid having to use the shift key. No octave information was specified in this simple example, so Mup would use its default, which in this case would be the octave beginning with middle C.

The first two notes are quarter notes, and the last note is a half note. Time values of notes are given as shown in the example. A quarter note is marked by a 4, a half note by 2, a sixteenth note by a 16, etc.

A semicolon is used to separate chords. In this simple example, each chord has only a single note in it, but it is possible to have lots of notes in one chord.

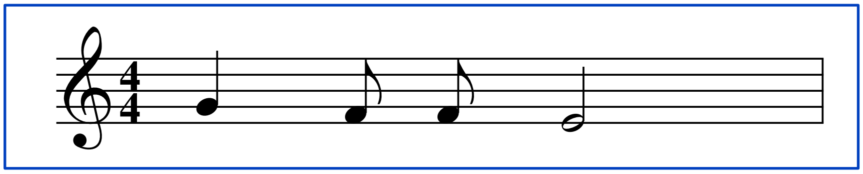
At the end of each measure, we have to tell Mup what kind of bar line to draw. The standard bar line is just called "bar." So a complete description of the first measure would be:

```
1:    4e; 4d; 2c;
bar
```

To save typing, Mup allows a lot of shortcuts. One such shortcut is that it assumes that unless you tell it otherwise, each note in the measure is like the note before. You can leave out the second 4, because if you don't specify a time value, Mup will assume the note is the same length as the previous note.

The same sort of idea works with pitches. The third measure of "Three Blind Mice" could be stated like this:

```
1: 4g;8f;;2e;
bar
```



The third note has no information given at all in this example—there is only a semicolon. In this case, Mup will get both pitch and time value from the previous note, so the actual third note in this measure would be an eighth note with pitch F.

You may have noticed that this example doesn't have spaces between chords. There are only a few places where the Mup language requires you to put spaces. However, you can always put some in other places to make things easier to read.

2.2 Parameters

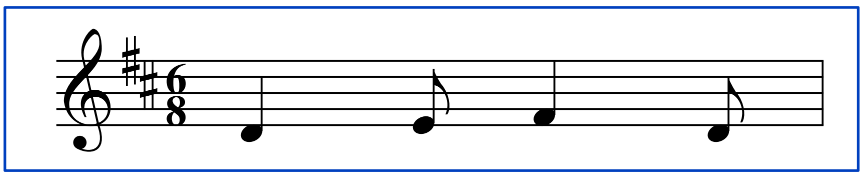
Printed music contains a lot more than just notes and bar lines. Among other things, each staff normally begins with a clef, key signature, and time signature. Mup provides default values for these, which you can then override if you want something different. In the examples so far, we didn't override anything, so Mup would assume its default values, which are treble clef, a key signature with no sharps or flats, and a time signature of 4/4.

There is a long list of "parameters" that can be set. Things like clef and key signature are among them. Parameters can be changed with a line of the form:

parameter_name=value

For example, suppose we have a song written in 6/8 time in the key of D major. We can convey this information to Mup like this:

```
score
  time=6/8
  key=2#

music
  1: 4d;8e;4f;8d;
  bar
```



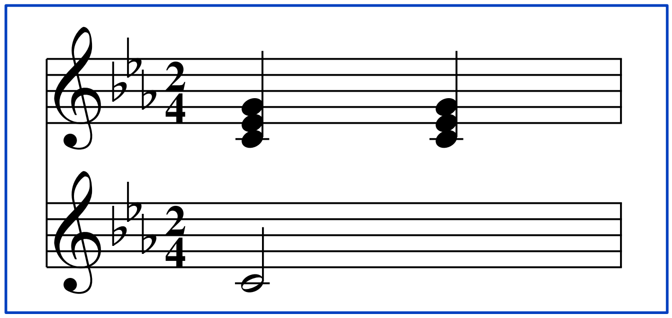
Note that in this example, the key was specified as two sharps. You can also specify the key by name:

```
  key = d major
```

These parameters give a very different sort of information than the notes of a measure, so they go in a separate section of the input file. Each section of the file describes information for a specific "context." Information about musical notes is given in "music" context, while things that apply in general to the whole score are given in "score" context. Once you start a measure in music context, you have to complete that measure before switching to another context, but otherwise you can pretty much change from one context to another as necessary. Each new context section is headed by its name (e.g., "music" or "score"). At the beginning of input, music context is assumed.

Here is a more complicated example:

```
score
      staffs=2
      key=3&
      time=2/4

music
1: ceg;;
2: 2c;
bar
```



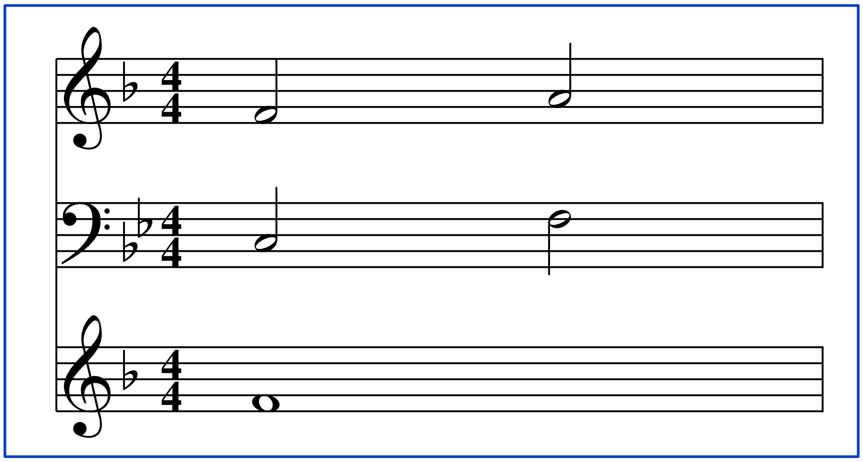
This example starts by setting some parameters. First it states that this piece of music should be printed with two staffs, instead of the default of only one. Then it gives a key signature. Since there is no "flat" symbol on a standard computer keyboard, Mup uses the "&" symbol for flat. The time signature is then set to 2/4.

Next we find the keyword "music," which indicates the end of parameters and the beginning of the music. Data is given for both staff 1 and staff 2. Staff 1 has two chords in the measure. The first is a C minor triad (it's minor since the key is three flats). No time value is specified for this chord. Since it is the very first chord of the piece, Mup cannot use the previous chord's time value, because there is no previous chord. In this case, Mup falls back to using the denominator (bottom number) of the time signature, so the chord is a quarter note. Incidentally, if Mup has to back up to previous notes to deduce pitch and/or time values, it only goes back as far as the beginning of the current measure. That means the default time value for the first chord of every measure in this piece would be quarter note. The second chord on staff 1 is the same as the first, since only a semicolon is specified.

Staff 2 has only a single chord, consisting of a half note with pitch C. Mup checks to make sure the time values on each staff add up to the time signature—no more or less. It is an error to specify too much time. For too little time, Mup will print a warning message (unless warnings are turned off via the warn parameter). If you have something like a "pickup" measure, which doesn't add up to the time signature, you can specify "space" rather than a chord, to account for the rest of the time.

Some parameters can be set on a per-staff basis as well as for the entire score. Mup also allows for up to three independent voices on each staff, and each voice can have parameters set that apply to only that voice. To get the value of a parameter, Mup always starts at the most specific place it could be defined and works toward the most general. In other words, it will first see if the parameter is set for the current voice. If not, it will see if it is set for the current staff. If not, it will use the value set for the entire score. Staff parameters are set in "staff" context, and voice parameters are set in "voice" context. As an example:

```
score
	staffs=3
	key=1&
staff 2
	key=2&
	clef=bass
music
1: 2f;a;
2: 2c;f;
3: 1f;
bar
```



Staff 2 will have two flats, whereas the other staffs will have one flat. Staff 2 will use the bass clef, whereas the other staffs will use treble clef (since that is the default when none is specified). All the available parameters are listed later in this User's Guide, along with information about whether they can be set for an individual staff or voice, or just for the score as a whole.

2.3 Page headers and footers

Mup allows you to specify a header and/or footer to put on the first page, as well as a header and/or footer to use on subsequent pages. These can include a page number that will be incremented automatically as pages are printed. The headers and footers can be customized as you like, with different fonts and sizes of text and items centered or left or right justified. There is also a shortcut "title" command that can be used to create a canned format title. For example:

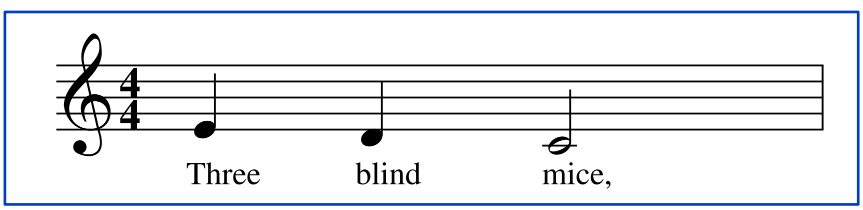
```
title "Three Blind Mice"
```

will create a centered title. You can also get left and right justified titles.

2.4 Lyrics

You can specify lyrics for as many verses as you like. They are specified somewhat like notes. As an example:

```
1: 4e;d;2c;
lyrics 1: 4;;2; "Three blind mice,";
bar
```

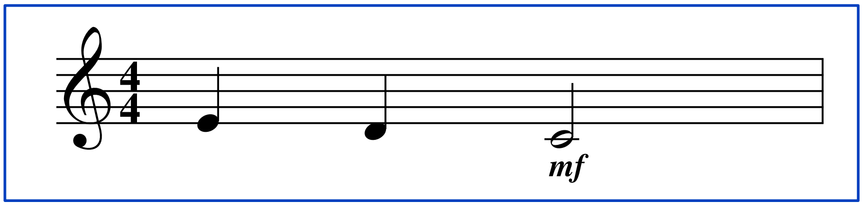


This example describes the lyrics to go with staff 1. There are three lyric syllables, having time values of quarter note, quarter note, and half note. The actual syllables are given inside the double quotes. Incidentally, since in this example the time values for the lyrics are the same as those of the notes, the time values don't actually need to be specified; if no lyrics time values are given, Mup assumes they match the note time values.

2.5 Miscellaneous

Mup provides a way to print arbitrary text (like "allegro") and musical symbols (like a fermata). It can also print phrase marks, piano pedal marks, etc. The placement of these items is specified in terms of "counts" into the measure. For example:

```
boldital below 1: 3 "mf";
```



tells Mup to print "mf" in bold-italics below staff 1, at count 3 of the measure. You can also print marks associated with specific chords. All of these facilities are described in detail in following sections.

2.6 Displaying, printing, and playing music

Once you have an input file, you can run Mup on it to get the printed version of the music. Entering:

`mup` *myfile*

from a command line prompt or selecting Run > Display from Mupmate will cause Mup to read *myfile*, which should contain text in the Mup input language. If there are no errors in *myfile*, PostScript output will be produced, which can be displayed on the screen or printed via Mupmate or other programs. Mup can also produce MIDI output, which can then be played on your speakers.

If you are using Ghostscript, but without Mupmate, two utility programs are included with Mup for displaying and printing music using Ghostscript. These are described in more detail in the next section.

3. RUNNING MUP

There are two basic ways to run Mup: directly from a command line or via the Mupmate program. You can use either approach, or switch between them as you wish. The Mupmate program just provides a more menu-driven environment on top of the Mup program itself.

You can create a Mup file using any ordinary text editor, and then run the Mup program on the file you created. On Windows, Notepad is a typical editor choice, and on Linux, editors like vim and emacs are commonly used, but pretty much any text editor (not word processor) can be used. Many people, however, prefer to be able to edit, display, and play from a single integrated and more graphical interface, and for them, a helper program called "Mupmate" is provided. The Mupmate program helps lead you through some of the steps, and you can easily access this User's Guide from its Help menu.

3.1 Mupmate

Mupmate is currently only supported on Microsoft Windows, Apple Mac OS X, and Linux. Since the source code is available, and it is based on the cross-platform FLTK toolkit, it would probably be fairly easy to make it run on any system supported by FLTK.

Once you have installed Mup and Mupmate on Windows, double clicking a .mup file in Windows Explorer will run Mupmate on that file. Or, you can run Mupmate by going to the Start menu, and choosing Programs, then Arkkra, and then Mupmate. If you would like an icon on the desktop, you can create one by right clicking the Mupmate choice in the Arkkra menu, choosing "copy", right clicking somewhere on the desktop, and choosing "paste".

On Linux, you can just type the mupmate command in a terminal window, optionally followed by the name of a Mup input file. Or you can add mupmate to your favorite window manager's menus.

On Mac OS X, you can double click on the MupMate.app in Finder. You should also be able to double click any file with a .mup suffix, which should then run Mupmate on that file. For setting paths in the Preferences, several "magic" variables are set automatically, if you have not already set them to something else. $APPL is set to the top of the application directory hierarchy. $RSRC is effectively set to $APPL/MupMate.app/Contents/Resources. $HOME is set to your home directory. $SUPP is set to your applications support folder (which is typically $HOME/Library/Application Support). $DOCS is set to your document folder (which is typically $HOME/Documents).

Mupmate provides five top level menus: File, Edit, Run, Config, and Help. The File menu provides commands for opening new files and saving the file you are working on, as well as exiting the program. The Edit menu provides the kinds of things you would expect in a editor: commands to find a pattern, or find and replace; to select text; to copy, cut, and paste; to go to a specific line; and to undo the previous operation, if you make a mistake or change your mind. The Run menu lets you set runtime options, and then run the Mup program on your input in various ways. You can either just generate a PostScript, PDF, or MIDI file, or display the PostScript or play the MIDI. The Config menu lets you specify what application program you want to use to view PostScript files and which you want to use to play MIDI files, and well as specify locations for other Mup files. Mupmate will try to find reasonable default values, but you may want to check that they are what you want, and tweak them if they aren't. For paths, you can include environment variables to be expanded, by giving their name preceded by a dollar sign. A tilde by itself will be expanded to your home directory, whereas a tilde followed by the name of a user will be expanded to that user's home directory. The Help menu lets you browse this User's Guide, view some startup hints, view the Mup license, or see the current version number of Mup and Mupmate.

Mupmate does not directly provide a print facility. Almost any PostScript viewer already provides this ability, so you can simply select "Display" from the Run menu and use the viewer's print capabilities. Alternately you can use "Write PostScript File" from the Run menu to create a PostScript file that you can print as you would any other PostScript file.

3.2 Mup Options

Mup accepts a number of options. When invoking Mup from a command line, the options are specified by a dash followed by a letter. On Windows/MS-DOS systems, you can substitute a slash instead of the dash. If you are using

Mupmate, you will use the "Set Options" form off of the "Run" menu to set the options. You just fill values into the form, and Mupmate will take care of the details of running Mup with your values, so you won't use the dash and letter shown below at all. Some of the options listed below are not available from Mupmate, either because they are meant for debugging, and thus not generally of interest to most users, or because Mupmate handles the appropriate details automatically. The options to the mup command (in alphabetical order) are:

`Command line:` **-c** *N*
`Mupmate:` Run > Set Options > Enable Auto Multirest and Min Measures to Combine

Combine consecutive measures of all rests or spaces into multirests (multiple measures of rest printed as a single measure, usually with the number of measures of rest printed above the staff). Any time there are *N* or more measures in a row that consist entirely of rests or spaces, they will be replaced by a multirest. The combining of measures stops when there is a visible staff that contains notes or lyrics, or that contain text or musical symbols after the first beat of the measure, or when there are parameter changes on a visible staff or in score context that change clef, key, or time signature, or when there is a bar line other than an ordinary bar. This option is most likely to be useful when printing a subset of staffs, where the particular staff(s) you are printing have long periods of rests. See information about the -s option and the "visible" parameter below. This option overrides the restcombine parameter.

`Command line:` **-C**
`Mupmate:` Option not available (only used for debugging).

This option is only used in connection with the -E option. It specifies that comments are to be passed through rather than deleted.

`Command line:` **-d** *N*
`Mupmate:` Option not available (only used for debugging).

Print debugging information. *N* is a bitmap, so you can turn on multiple debugging levels by adding up the flag values. For example, if you want to turn on both level 2 and level 4 tracing, *N* would be 6 (because 2+4=6).

1	input syntax/grammar analysis tracing
2	high level parse phase tracing
4	low level parse phase tracing
8	reserved
16	high level placement phase tracing
32	low level placement phase tracing
64	reserved
128	contents of the main internal list
256	high level print or MIDI phase tracing
512	low level print or MIDI phase tracing

N can be specified in decimal, octal (by using a leading zero), or hex (by using a leading 0x). This information is intended for debugging of Mup itself and thus is not likely to be of use to the average user, and is not available from Mupmate.

`Command line:` **-D** *MACRO[=macro-def]*
`Mupmate:` Run > Set Options > Macro Definitions

Define the macro *MACRO*. The macro name must consist of uppercase letters, digits, and underscores, beginning with an uppercase letter. The *macro_def* is optional, and gives the text of the macro. On UNIX, Linux, or similar systems, if it contains any white space or other special characters, it must be quoted. On other systems, white space may not be allowed. The -D option can be specified multiple times, if you wish to define more than one macro.

`Command line:` **-e** *errfile*
`Mupmate:` Option not needed. Mupmate automatically saves and displays error output.

Place the error message output into *errfile* instead of writing it to the standard error output stream.

`Command line:` **-E**
`Mupmate:` Option not needed (only used for debugging).

Rather than produce PostScript or MIDI output, just expand macros and includes, and write the result to the standard output stream. Comments in the input are deleted, unless the -C option is also specified.

`Command line:` **-f** *outfile*
`Mupmate:` Option not needed. Mupmate automatically creates appropriate output file.

Place the PostScript output into *outfile* instead of writing to the standard output.

`Command line:` **-F**
`Mupmate:` Run > Write PostScript File

This is like the **-f** option, except the name of the output file is derived from the name of the Mup input file. If the name of the Mup input file ends with a ".mup" suffix, the generated PostScript output file will end with a ".ps" suffix instead. If the name of the Mup input file ends with a ".MUP" suffix, the PostScript file will end with a ".PS" suffix. Otherwise, a ".ps" suffix will be appended to the end of the Mup input file name. If multiple input files are listed, the last is used. If none are specified (input is read from standard input), the name "stdin.ps" will be used for the output file.

`Command line:` **-l**
`Mupmate:` Help -> License

Show the Mup license and exit.

`Command line:` **-m** *midifile*
`Mupmate:` Option not needed. Mupmate automatically creates appropriate output file.

Instead of generating PostScript output, generate standard MIDI (Musical Instrument Digital Interface) output, and put it in *midifile*. This option also causes the macro "MIDI" to become defined.

`Command line:` **-M**
`Mupmate:` Run > Write MIDI File

This is like the **-m** option, except the name of the MIDI file is derived from the name of the Mup input file. If the name of the Mup input file ends with a ".mup" suffix, the generated MIDI file will end with a ".mid" suffix instead. If the name of the Mup input file ends with a ".MUP" suffix, the MIDI file will end with a ".MID" suffix. Otherwise, a ".mid" suffix will be appended to the end of the Mup input file name. If multiple input files are listed, the last is used. If none are specified (input is read from standard input), the name "stdin.mid" will be used for the MIDI file.

`Command line:` **-o** *pagelist*
`Mupmate:` Run > Set Options > Pages to Display

Print only the pages given in *pagelist*. The *pagelist* can begin or end with optional qualifiers. A qualifier of "odd" will restrict printing to only odd numbered pages, while a qualifier of "even" will restrict to even numbered pages. A qualifier of "reversed" will cause pages to be printed in reverse order, which may be useful for printers that stack output in backwards order. A comma-separated list of pages and/or page number ranges can also be specified, where a range is two numbers separated by a dash. For example,

---

* UNIX is a registered trademark of X/Open Company Limited.

-o1,7-9,12-14 would print pages 1, 7, 8, 9, 12, 13, and 14. Adding a qualifier, -oodd,1,7-9,12-14 would print pages 1, 7, 9, and 13, while -o1,7-9,12-14,even,reversed would print pages 14, 12, and 8 in those orders. -oreversed would print all pages backwards, while -oreversed,odd would print all odd pages backwards. There is also a special page number of "blank" which will result in a completely blank page being output. This is most likely to be useful when the panelsperpage parameter. is set to 2. For example, to print a one-page song on the right-hand panel rather than the left, you could use -oblank,1

`Command line:` **-p** *N*
`Mupmate:` Run > Set Options > First Page's Page Number

Start numbering pages at *N* instead of at 1. This can be set inside the Mup input file with the "firstpage" parameter, but the command line option will override the parameter. The page number can optionally be followed by a comma and either leftpage or rightpage, to specify whether any header, footer, top, or bottom on the first page should use the left or right page versions, if they are different. This would, of course, also control whether the left or right versions of header2, footer2, top2, and bottom2 are used on subsequent pages. If **-o** and **-p** are used together, the page numbers given in the **-o***pagelist* must be the printed page numbers. For example, if you use -p10 and want to print just the second page, you would need to specify -o11.

`Command line:` **-q**
`Mupmate:` Option not needed.

Quiet mode. Omits printing of version and copyright notice at startup.

`Command line:` **-s** *stafflist*
`Mupmate:` Run > Set Options > Staffs to Display/Play

Only print the staffs that are included in *stafflist*. The *stafflist* can be a comma-separated list of staff numbers or ranges, such as "1,5" or "1-3,7-8" but no spaces are allowed in the list. If the -m or -M option is also used, to produce MIDI output, this option controls which staffs are played rather than which are printed. If you want only a single voice to be printed or played, you can follow a staff number or range with **v1** or **v2** or **v3** to restrict to voice 1, 2 or 3 respectively, such as "1v2" or "1-4v1,5-6v2". Otherwise all voices on the staff are printed or played. You can't specify a list or range for voices; if you only want to make two out of three voices visible, you have to specify them separately, like "1v2,1v3". See also the "visible" parameter.

`Command line:` **-v**
`Mupmate:` Help > About Mupmate

Print the Mup version number. When invoked from command line, Mup will then exit. This document is for version 7.2.

`Command line:` **-x***M***,***N*
`Mupmate:` Run > Set Options > Extract Measures

Extract measures *M* through *N* of the song. This allows you to print or play a part of a song. The comma and second value are optional; if not specified, the default is to go to the end of the piece. Positive values specify the number of measures from the beginning of the piece, while negative values are relative to the end, with -1 referring to the last measure of the song. So -x1,-1 means the entire song, if the song doesn't have a pickup measure. If the song has a pickup measure, that is specified by 0. So for a song with a pickup, -x0,-1 would mean the entire song, and -x0,0 would mean just the pickup measure. As other examples, -x-1,-1 means just the final measure of the song, -x2 means starting after the first full measure, -x3,4 means only measures 3 and 4, and -x6,6 means just measure 6. The starting measure is not allowed to be inside an ending. A common use for this option might be to generate a MIDI file for just a few measures. For example, if you were trying to tweak tempo values for a ritard in the last 2 measures of a song, you could use -x-2 to listen to just those measures.

When invoked from command line, the options, if any, can be followed by one or more *files* in the format described in this User's Guide. If no *files* are specified, standard input is read. If several *files* are listed, they are effectively concatenated together and treated as one big file. Since there are some things (such as header and footer) that are only allowed to occur once, if you have several independent pieces, Mup should be called on each individually rather than trying to print them all with one command. If a specified file does not exist, and its name does not already end with .mup or .MUP, then Mup will append .mup to the specified name and attempt to open that.

If you just want to create a PostScript output file, for printing on a PostScript printer, or viewing with a tool such as GSview, you can use the -f option, as in:

```
mup -f outfile.ps infile.mup
```

Or on Unix, Linux or MS-DOS command window, you could redirect the output into a file using the > character, as in:

```
mup infile.mup > outfile.ps
```

For more debugging, in addition to the -d option, if the environment variable MUP_BB is set to "bcfgnsu" or any subset of those letters, the generated output will include "bounding boxes" for the things Mup internally calls bars (b), chords (c), feeds (f), grpsyls (g), header/footer and top/bottom (h), notes (n), staffs (s), and stuff (u). While this is intended for use in debugging Mup itself, it may also help you understand why Mup places things the way it does, since in general, Mup only allows bounding boxes to overlap according to specific rules. If viewed with a color PostScript viewer (not Mupdisp, which is covered below), these boxes will be in color.

3.3 Mupdisp and Mupprnt utility programs

If you are using Ghostscript, but prefer to not use Mupmate, there are two utility programs included with Mup for displaying and printing music.

`mupdisp` *Mup_options myfile*

will display music to your screen, while

`mupprnt` *Mup_options myfile*

will print the output to your printer. See the installation instructions for more details on configuring the Mupprnt program for your printer.

The *Mup_options* can be any of the options listed in the section on "Command line arguments" except -C, -E, -f, -F, -l -m, -M, or -v, which don't send PostScript output to the standard output.

The Mupdisp program allows you to view pages in any order, with either a version small enough to fit on your screen or a near-actual-size version that you can scroll if it doesn't fit on your screen. Mupdisp will run under MS-DOS/Windows or will run under UNIX with TERM of AT386, linux, or xterm (under X windows).

Mupdisp begins in partial page mode, which displays output at approximately actual size (depending on the size of your monitor). In this mode, it is possible that not all of the page fits on the screen, so the scrolling commands can be used to move up and down to view different parts of the page. In full page mode, a small version of the entire page is displayed. This is useful for seeing overall page layout, but is generally too small to see much detail. This mode is now somewhat of a relic of the days when screens were typically much smaller than they are today, and is thus becoming less useful.

Once the music has been drawn on the screen, you can enter various commands to view different pages or parts of the current page. The commands are:

num<Enter> Go to page number *num*.

+ or <space> or <control-E> or <control-F> move forward on the page by about 1/8 of an inch (partial page mode only)

- or <backspace> or <control-Y> or <control-B> move backward on the page by about 1/8 of an inch (partial page mode only) b or <control-U> or <control-P> or <upward-arrow-key> move backward on the page by about an inch (partial page mode only)

f or <Enter> or <control-D> or <control-N> or <downward-arrow-key> move forward on the page by about an inch (partial page mode only)

h or ? display help screen

m toggle between partial page and full page modes.

n or <PageDown> go to next page

p or <PageUp> go to previous page

q or ZZ quit

r Repaint the page (useful for exiting help page)

When in X windows, the mouse can be used for scrolling. The left button scrolls downward like the f command, while the right button scrolls backwards like the b command.

4. MUP FILE STRUCTURE

Mup files do not have to follow any naming convention, although on systems that use file name suffixes to associate a file with an application, it is traditional to use .mup for the suffix. It can also be useful to put a special "magic string" on the first line of Mup files. This magic string is completely optional, but having it there makes it easy for both people and programs to identify the file as Mup input. The standard recommended value for this string is:

```
//!Mup-Arkkra
```

with exactly that spacing and capitalization. If the file uses features of newer versions of Mup, and thus would not work with older versions, you can add a dash and the minimim version number the file requires, as in:

```
//!Mup-Arkkra-7.2
```

4.1 Mup General Syntax

Any number of spaces and tabs can be put in almost anywhere except in the middle of a word. Each statement goes on a separate line. If for some reason you wish to split a statement onto several lines, each but the last must end with a "\" (backslash) character, to tell Mup to treat the next line as a continuation of the current line. Blank lines can be put between statements to make things easier to read.

Comments begin with two slashes and continue to end of line. For example:

```
// Note: in some early manuscripts, this chord had an accent
```

Comments will be ignored by Mup, and are for your own use to remind yourself of something. (There are a couple exceptions, covered under Special #file and #line comments in the Hints section.)

Staffs are numbered from top to bottom, starting at staff 1.

Several different units are used for distances. One is inches or centimeters. (There is a units parameter that is used to select which you want to use.) Another is "stepsizes." One stepsize is half the distance between two staff lines. In the horizontal dimension, "counts" are sometimes used. A "count" refers to the musical duration of a note with a duration of the denominator (bottom number) of the time signature. The actual distance on the page will vary depending on how Mup determines notes should be placed. When you use a time signature with two or more fractions added together, as in 3/4 + 3/8, the "count" is the largest denominator, which would be 8 in the example just given.

Uppercase and lowercase letters are not interchangeable. Thus, for example, "SCORE" is not the same as "score."

Most lines of input end with a semicolon. There are some kinds of input that do not require an ending semicolon, but Mup will allow semicolons on most of those too, so that if you can’t remember if a given command requires a semicolon or not, you can just use one anyway. For the examples in this User’s Guide, semicolons are not used when they are not necessary.*

In music context, newlines are required to separate commands: music data, bar lines, rolls, commands to print strings, dynamic marks, lyrics, etc. all must each end with a newline. In grids context, shapes context, and headshape context, each pair of strings must end with a newline. In other contexts, there are a few cases where newlines between commands are optional, although you may wish to use them anyway to improve readability.

Mup supports macros and conditionals (like ’if’ and ’ifdef’) that can be placed anywhere in input, except in the middle of words, numbers, or strings; they needn’t be on separate lines.

In a number of statements, Mup expects a text string. All strings must be enclosed in double quotes. For example:

---

* For setting, unsetting, saving, or restoring of parameters, either a semicolon or a newline is required. For bar, multirest, print, left, right, center, title, line, curve, newscore, newpage, postscript, and paragraph a newline is required, but it may optionally be preceded by a semicolon. For commands for changing context, semicolon and newline are both optional, but a newline is traditionally used to improve readability.

```
"This is a string."
"Allegro"
```

A string can contain any combination of letters, numbers, spaces, and punctuation. It can also contain various things that will cause printing of special music characters, change font and size, and so forth. Those things are covered in the chapter on text strings.

4.2 Contexts

There is always a current Mup "context" that is in effect. When Mup begins reading input, it is operating in "music" context, which is where music, lyrics, barlines, and other related things are described. You can change to another context by entering its name. A context remains in effect until another context is named. The contexts are:

header to define what goes at the top of the first page, typically the title, composer, etc.

footer to define what goes at the bottom of the first page, typically a copyright notice, performance notes, etc.

header2 to define what is to be printed on the top of pages after the first page.

footer2 to define what is to be printed on the bottom of pages after the first page.

top to define what is to be printed on the top of page. This gets printed below the header (or header2), if any. If the output is not already at the beginning of a new page, a new page is started. Unlike header, which can only be used once, and is used only on the very first page, top can be used multiple times. In a song with multiple movements, you might use top to put a title at the beginning of each movement.

bottom to define what is to be printed on the bottom of page. This gets printed above the footer (or footer2), if any. If the output is not already at the beginning of a new page, a new page is started. Unlike footer, which can only be used once, and is used only on the very first page, bottom can be used multiple times.

top2 to define what is to be printed on the top of pages, after the page that uses "top." If the output is not already at the beginning of a new page, a new page is started.

bottom2 to define what is to be printed on the bottom of pages after the page that uses "bottom." If the output is not already at the beginning of a new page, a new page is started.

block to define a block that contains text rather than music.

score to define parameters that apply to the entire score.

staff *S* to define parameters to be used for staff *S*, where *S* is a number from 1 to 40. You can also specify a comma-separated list of staffs or staff ranges:

    staff 3,7 // staffs 3 and 7
    staff 1-2, 5-8, 10 // staffs 1, 2, 5, 6, 7, 8, and 10

voice *S V* to define parameters for a particular voice *V* on staff *S*. The voice *V* can be either 1, 2, or 3. *S* is a staff number from 1 to 40. Multiple staffs and/or voices can be specified:

    voice 1-4 1   // voice 1 on staffs 1 through 4
    voice 1-2 2 & 3 1 // voice 2 on staffs 1 and 2 and voice 1 on staff 3

control to save and restore parameter settings

grids to define grids (typically for guitar)

headshapes to define what notehead shapes to use for notes of various durations. This context is rarely used, and is described in the chapter on shaped notes.

shapes "*name***"** to define overrides for certain musical symbols, which could be applied on a per-staff, or even per-voice basis.

symbol "*name*" to define user-defined symbols, or override the appearance of built-in music symbols.

accidentals "*name*" to define symbols and frequency adjustments to use for accidentals. More details are given in the chapter on Custom Accidentals and Alternate Tunings.

keymap "*name*" to define a mapping from what you type in to other symbols. This is typically used to make it easier to enter strings that you want printed in another alphabet, like Cyrillic or Greek. More details are given in the keymap section of the chapter on text strings.

music to define everything else. This includes notes, lyrics, bar lines, phrase marks, tempo and dynamic marks, etc.

Most contexts are optional. An input file just needs to contain either at least one measure of music or at least one block.

All the contexts for things that go at the tops or bottoms of pages (i.e., header, footer, header2, footer2, top, bottom, top2, bottom2) can have different versions for left and right pages, by following their name with a modifier of "leftpage" or "rightpage." This is described more fully in the section on Headers and Footers. Each of the three variations of header, footer, header2, or footer2 contexts can be used only once, although they can be placed anywhere in the file. The other contexts may appear any number of times in any order, and the order in which they occur is significant in determining the output produced.

5. BASIC STANDARD MUSIC NOTATION

Music data is given in the "music" context. For each measure, there is usually one line of input for each voice. At the end of the measure, the kind of bar line to be used to end the measure is specified. This section describes the input for generating standard music notation. Mup can also generate tablature notation, and that is covered in the next section.

5.1 Specifying chords

5.1.1 Staff and voice

The description of the music for one voice begins with the staff and voice number, followed by a colon. For example:

```
3 1:
```

indicates that the remainder of the line contains musical information for voice 1 of staff 3. If the "voice" number is omitted, voice 1 is assumed. Thus

```
3:
```

is equivalent to the previous example. Both the staff and voice can be given as a list. This may be useful if several staffs have the same notes, or multiple voices on a staff have the same notes. For example:

```
1-4 2:           // voice 2 of staffs 1, 2, 3, and 4
1,2,4:           // voice 1 of staffs 1, 2, and 4,
1,3,6-7 1-2:     // voices 1 and 2 on staffs 1, 3, 6 and 7
5-8 1,2:    // voices 1 and 2 on staffs 5, 6, 7, and 8
```

If you want notes to go to one voice on some staffs and a different voice on others, this can be specified using an ampersand. For example:

```
1 1 & 3 2:
```

will cause the music to go to voice 1 of staff 1 as well as to voice 2 of staff 3. Various styles can be combined:

```
// Voices 1 and 2 on staff 2,
// as well as voice 2 on staff 4
// and voice 1 on staffs 6, 7, and 9
2 1-2 & 4 2 & 6-7,9 1:
```

It is possible to have up to 40 staffs and up to three voices per staff. While there can be voice crossings, in general voice 1 should be the "top" voice, voice 2 the "bottom" voice, and voice 3 the "middle" or "extra" voice. With the first two voices, Mup tries hard to avoid any "collisions" between notes, rests, and other things. Since voice 3 is an extra voice, there are some cases when overlap with the other voices is basically unavoidable, but there are some techniques discussed later that let you tweak placement when necessary.

As an alternative to this voice-at-a-time input style, there is also a chord-at-a-time input style, which will be covered later.

5.1.2 Chord duration information

The rest of the line contains a list of chords, with a semicolon at the end of each chord. Each chord has a time value. The time values of all the chords in the line must add up to no more than the time signature. Time values are given as follows:

Input	Meaning
1/8	octuple whole
1/4	quadruple whole
1/2	double whole
1	whole
2	half
4	quarter
8	eighth
16	sixteenth
32	thirty-second
64	sixty-fourth
128	128th
256	256th

Any of these time values can be followed by one or more dots, to indicate a dotted note. Each dot increases the time value by 50% of the preceding note or dot.

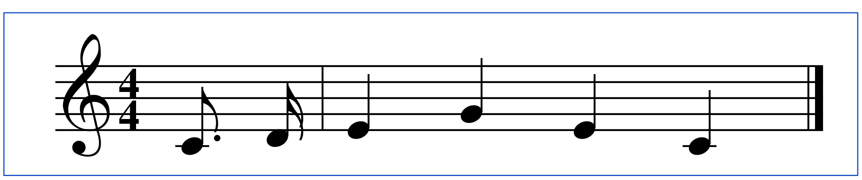
It is also possible to specify time as two or more times to be added together. For example, 2+8 would indicate the time of a half note plus the time of an eighth note, or in other words, a half note tied to an eighth note. The expression can also include subtractions, like 2.-16. When subtractions are present, it isn't clear what time values you want Mup to use, so it will start with the largest possible time value and add enough additional chords to add up to the total. For example, if you were to use 1-4, indicating a whole note minus a quarter note, Mup will use a dotted half note, even though there are a number of other ways to represent that total time, such as a half note tied to a quarter note.

If a time value is not specified for the first chord in a measure for a given voice, the default timeunit value is used. You can set the default value using the "timeunit" parameter as described in the "Parameters" section. If that parameter is not set, the default is the denominator (bottom number) of the time signature. For chords after the first, if a time value is not specified, the time value for the previous chord is used.

5.1.3 Notes, rests, or spaces

There are three kinds of "chords." The first type consists of one or more pitches, given by the letters "a" through "g". Parentheses can be placed around the pitch if you want the note to be printed in parentheses. (If the pitch is modified by an accidental or octave, which will be discussed later, those must also be included inside the parentheses.) The second is a rest, which is designated by the letter r. The third type is a "space," designated by the letter s. Space is basically a placeholder that takes up time, but doesn't print anything. It is useful when a certain voice only has notes during part of the measure. It can also be useful for specifying "pickup" measures to account for the time before the first note in the measure. For example:

```
// a pickup measure
1: 2.s;8.c;16d;
bar
1: e;g;e;c;
endbar
```



If all voices contain spaces, no space is actually taken up on output. Most of the time, this will be what you want. For example, when you are using space for a pickup, the space is just to add up to a measure, and you don't want any actual blank space at the beginning of the piece. Once in a while, however, you may want space to actually be allocated on output, perhaps to be able to allow space for some special notation. In that case, you prefix the "s" with a "u" to indicate an uncollapsible space. For MIDI, normal space is squeezed out to take no time, whereas uncollapsible space essentially becomes a rest.

If a given voice is omitted for a particular measure, Mup normally defaults to a measure of space, but you can make it default to something else (most commonly a measure of rest) by setting the emptymeas parameter.

5.1.3.1 Measure duration

There is a special duration of "m," which means an entire measure. It can only be used with a rest, space, or "rpt" (repeat). For example:

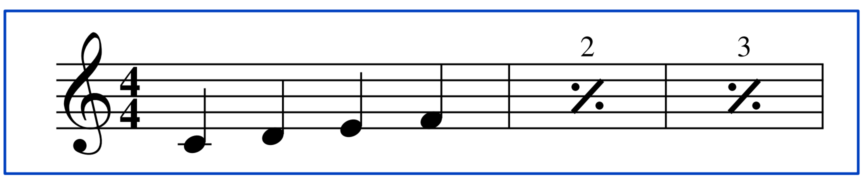
```
1 1: mr;
1 2: ms;
```

A measure rest looks like a whole rest, but is centered in the measure. It should be used when an entire measure is a rest, regardless of the time signature. However, if you want to force use of a symbol other than the whole rest symbol, you can specify a duration before the mr, and the rest symbol for that duration will be drawn instead.

```
4mr;   // use a quarter rest symbol
1/4mr;  // use a quadruple whole rest symbol
2.. mr;  // use a double-dotted half rest
```

Using "m rpt" will cause the measure repeat symbol to be printed, indicating the measure is just like the previous measure. Measures repeats will automatically be numbered, unless the numbermrpt parameter is set to n. If there is more than one voice, you only need to specify the mrpt on voice 1. If you do specify other voices as well, they must be either a mrpt or ms.

```
1: c;d;e;f;
bar

// another measure just like the first
1: m rpt;
bar

// the space between m and rpt is optional
1: mrpt;
bar
```



It is also possible to specify "dbl m rpt" for a double measure repeat, or "quad m rpt" for a quad measure repeat. The subsequent measures that are part of a multi-measure repeat must be either all spaces ("ms" or "mus") or not specified at all. There must, of course, be at least two measures of music before a dbl and at least four before a quad, as well as at least one measure after a dbl and at least three measures after a quad.

The measures associated with a mprt are not allowed to contain changes in time signature, key, clef, transpose, addtranspose, or vscheme, and the number of staffs cannot be reduced to eliminate the staff with the mrpt. For the purposes of counting measures, invisbar counts just like visible ones. Bar types of repeatstart, repeatend, or repeatboth are not allowed.

5.1.3.2 Accidentals

Each pitch letter in a chord may be followed by up to 4 accidentals, although typically no more than one is used. (Probably the most common uses of more than one would be either something like n# to remind the user that the note had had a double sharp earlier, or when using non-standard tunings.) The standard accidentals are:

Input	Meaning
#	sharp
&	flat
x	double sharp
&&	double flat
n	natural

Note that a double flat or double sharp counts as a single accidental, so this is one place where the use of spaces matters: & & with a space between is two flat accidentals, whereas && with no space between is one double flat accidental. It is possible to define your own custom accidentals, which is covered in the chapter on Custom Accidentals and Alternate Tunings. The accidental(s) can be placed inside parentheses if you want them to be printed within parentheses. All accidentals have to be within a single set of parentheses in that case; you can't put just a subset in parentheses.

5.1.3.3 Octave

An octave indicator may be specified after the pitch letter or optional accidentals. The octave can be specified in either of two ways: absolute or relative. A number from 0 to 9 is used to specify an absolute octave. Octaves run from C up to B. Octave 4 is the octave starting on middle C. Octave 3 is the octave below that, etc. A relative octave is specified by one or more plus or minus signs, and indicates that number of octaves above or below the default octave. For example:

```
c   // c in the default octave
e++ // e two octaves above default octave
f#---    // f# three octaves below default octave
b&6 // b flat in octave 6
```

The default octave can be set using the "defoct" parameter, which is described in the "Parameters" section. If that parameter is not set, the default octave is the octave containing the note associated with the middle line of the staff, based on the current clef. In other words, the default octaves are:

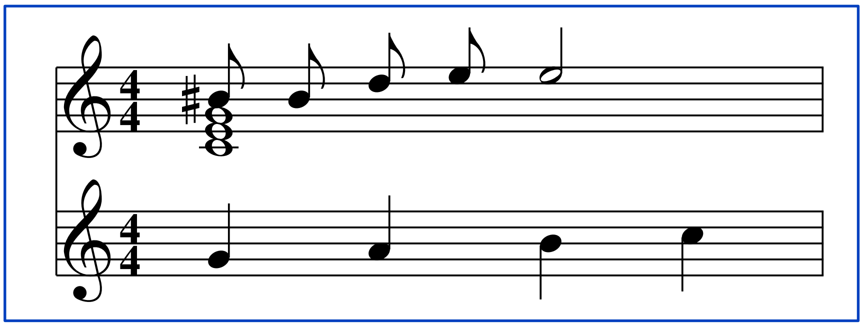
Clef	Octave
frenchviolin	5
8treble	5
treble	4
soprano	4
mezzosoprano	4
alto	4
8bass	4
treble8	3
tenor	3
baritone	3
bass	3
bass8	2
subbass	2

5.1.4 Shorthand notations

If a chord is omitted, then most of the values for the previous chord are reused. This includes duration, pitch/rest/space, and octave, as well as notehead size and shape (which are described in the next section). For normal, 5-line staffs, the pitches for the first chord of every measure must always be specified, since there is no previous chord.

Putting all these things together, here are some examples:

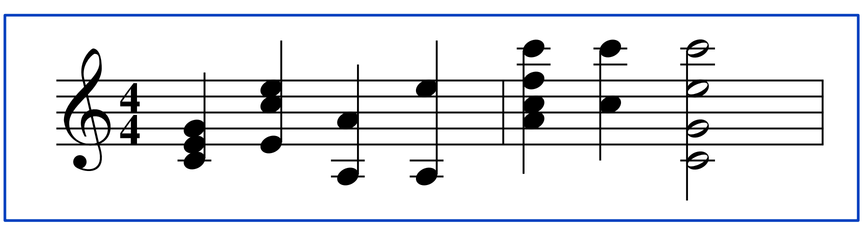
```
// Two eighth notes, each b#, followed by an eighth
// note d, eighth note e, and half note e, with the
// last 3 notes being in the next higher octave.
1 1: 8b#; ; d+; e+; 2;

// Whole note C-E-G chord in default octave
// for voice 2 of staff 1
1 2: 1ceg;

// Four quarter notes on staff 2, voice 1.
// The last is in the octave above the default octave.
2: 4g; a; b; c+;
```



For 1-line staffs, it is never necessary to specify a pitch, since all notes go on the single staff line. You can, however, specify a pitch if you wish. The pitch will be ignored for the purposes of printing, but will be used for MIDI output. If you don't specify a pitch, it is arbitrarily set to middle C.

If you have several notes in a chord, it is generally convenient to input them in order, either top-down, or bottom-up, as you prefer. If you like to always use a particular order, there is an alternate input mode, set via the noteinputdir parameter, that may save some typing. If this is set to "up" or "down" then the octave of each note in a chord after the first is determined relative to the previous note in that chord. If a note letter is an octave or less from the previous, you list it without any octave specified. Mup will then use the next instance of that letter in the appropriate direction. If it is farther than that, you use plus signs (when noteinputdir=up) or minus signs (when noteinputdir=down) to say how many octaves away it should be. Here is an example:

```
score
  noteinputdir=up
music

1: ceg;ece;a-a;a-e+;
bar

1: acfc;c+c;2cgec;
bar
```



5.1.5 Note attributes

There are several optional attributes that can be specified for each note. Any or all of these may appear on any note in any order after the letter, accidental, and octave specifications. They include:

— small notehead

— tie

— slurs

— headshape

— noteleft string

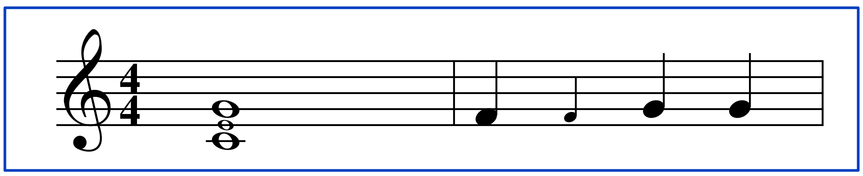
— location tag

5.1.5.1 Small notehead

A note specification can be followed by a "?" to indicate the note is to be printed with a small notehead, rather than the normal size. (Note: if you have several notes in a chord and want all of them to be small, the "cue" construct, described later, may be preferable.)

Examples:

```
// print the "e" as a small note
1: 1c e? g;
bar

// make the second note small
1: f; ?; g; ;
bar
```

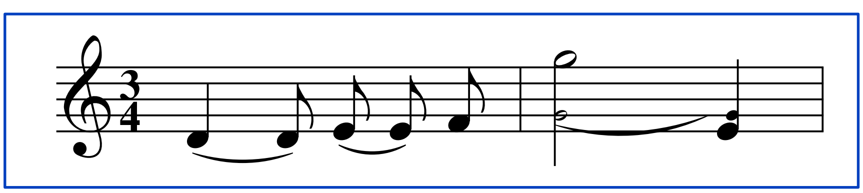


5.1.5.2 Note ties

A "˜" can be used to indicate the note is to be tied to the note of the same pitch in the following chord. That following chord need not be in the same measure, but it must contain a matching note. (Note: if a chord with several notes is to have all the notes tied to the following chord, the "tie" construct described later may be preferable.) The ˜ may be preceded by the word "dotted" or "dashed" if you want a dotted or dashed tie; otherwise a normal, solid tie is drawn. The ˜ may be followed by the word "up" or "down" to specify the direction for the curve's bulge. If neither is specified, Mup will determine an appropriate direction, so you only need to give a direction if you wish to override Mup's choice. After the aforementioned items (if any), you may put "to voice *N*" to tie to the matching note in voice *N* rather than in the current voice.

Examples

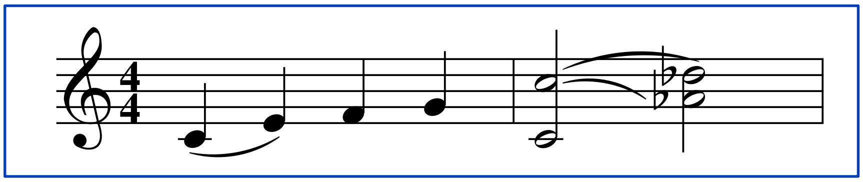
```
1: d˜; 8; e˜; ; f;
bar

// tie the g (which also happens
// to be a "small" note)
1: 2g+g?˜; 4eg?;
bar
```



5.1.5.3 Slurs

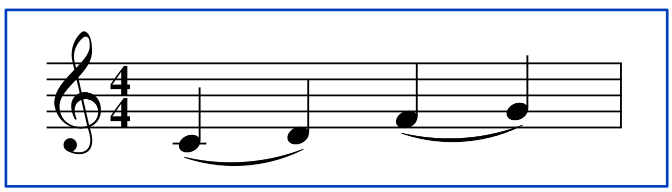
A note can also be followed by a list of notes to be slurred to, enclosed in angle brackets. A slur will be drawn from the note to each of the notes listed in the angle brackets. The notes inside the brackets are specified by a pitch and optional octave. No accidentals are specified, even if the note to be slurred to has an accidental. Examples:

```
// slur from c of first chord to e of second chord
1: c<e>; e; f; g;
bar

// slur from c+ of first chord to a& of second chord
// and from c+ of first chord to d&+ of second chord
1: 2cc+<ad+>; a&d&+;
bar
```



If there is only one note in the following chord, it isn't necessary to explicitly state it within the angle brackets; "<>" will suffice.

```
// slur from c to d, and f to g
1: c<>; d; f<>; g;
bar
```



The < may be preceded by the word 'dotted' or 'dashed' if you want a dotted or dashed slur, otherwise a normal, solid slur is drawn. The > may be followed by the word "up" or "down" to specify the direction for the curve's bulge. If neither is specified, Mup will determine an appropriate direction, so you only need to give a direction if you wish to override Mup's choice. After the aforementioned items (if any), you may put "to voice *N*" to slur to the matching note in voice *N* rather than in the current voice.

There are four special "slurs" which are really slides to/from an indefinite note. They are most commonly used on tablature staffs, but are allowed on ordinary staffs too. They are: </n> <\n> <n/> and <n\> for sliding upward into the note, downward into the note, upward out from the note, and downward out from the note respectively. These have to be in angle brackets by themselves, so if you want to have both one of these slides and another slur on the same note, multiple sets of angle brackets must be used, as in

```
1: 2c</n><d>;d;
```

5.1.5.4 Head shape

Sometimes you may want to mix head shapes on a single stem. For example, you might want to use a diamond to designate a harmonic, with other notes in the same chord being normal shape. There will be an entire section on head shapes, so only a simple example is given here. To make a single note have a different head shape, use hs followed by the name of the shape in quotes.

```
1: 2e e+ hs "diam";g;
```

5.1.5.5 Noteleft string

An underscore followed by a quoted string will cause that string to be printed to the left of the note and any accidentals. Typically this would be used for fingering, but the string can be anything.

```
1: 2f _"3";g _"4";
```

There are noteleftfont, noteleftfontafmily, and noteleftsize parameters for adjusting the string appearance.

5.1.5.6 Note location tag

Finally, a "location tag" can be associated with a note. This would enable you to draw things relative to the note. A note location tag is set by using an "=" followed by a name. The name can be either:

— a single lowercase letter

— an underscore followed by one or more letters, numbers, and underscores in any combination.

The name is arbitrary, and is used as a tag that can be referred to later. Examples:

```
// associate tag p with note e&
3: 2c; e& =p g;

// associate tag _end with note f
2: 1f =_end;
```

Location tags can only be used when defining a single voice.

5.1.6 Chord attributes

There are optional attributes that are associated with an entire chord rather than an individual note. These are put inside square brackets before the list of notes and the time value. There are several classes of information:

— chord style (grace, cue, diam, or xnote)

— head shape

— symbols to be printed with a chord

— slashes

— stem direction

— stem length

— padding

— chord location tag

— horizontal offset

— rest distance

Each class of information is separated from the other by a semicolon. All classes are optional, and can occur in any order.

5.1.6.1 Chord style

Several chord styles can be specified. Grace note chords are designated by the word "grace," or cue note chords by "cue." Grace notes can be specified as any undotted time value of eighth note or shorter, but are treated as taking no time. The time value given is merely used to specify how many flags or beams to put on the grace note. You can also specify a quarter note or longer grace note, which will be printed as a small, stemless note, as is used to show a "prebend" in guitar music. (Actually, you can force a stem if you want, by specifying a len, as is described later.) Cue notes have time like regular notes, but they are printed smaller. If there are several grace notes in a row, they will be beamed automatically. However, you cannot mix quarter (stemless) grace notes with shorter ones. Cue notes follow the same beaming rules as regular notes.

It is also possible to specify "diam," or "xnote." In the case of "diam," the chords will be drawn with diamond-shaped notes, while with "xnote," the chords will be drawn with X-shaped notes. A diamond shaped notehead will be used for xnotes that are half note or longer. Here are some examples:

```
[grace]
[xnote]
[cue; xnote]
[diam]
```

5.1.6.2 Head shape

If you want to have the noteheads in a chord use a shape other than the normal shapes, you use hs followed by the name of the head shape in quotes. There will be an entire section on shaped notes, so only the basics are covered here. The xnote and diam described above are really just shorthands for two common head shapes. The method using hs lets you use many other notehead styles, such as a slash, or triangle.

```
1: [hs "righttri"]cf; [hs "slash"]fa;
```

The section on shaped notes lists all the built-in head shape names, and explains how you can define your own.

5.1.6.3 Symbols to be printed with a chord

It is possible to specify one or more musical symbols or text strings to be printed with a chord. This is typically used for things like staccato and accent marks. The word "with" is followed by one or more things to print. If there are more than one, they are separated by commas. They will be printed outwards from the notes in the order specified.

The "with" list is not allowed on space groups. On rests, the side on which the items are printed is determined as if the rests had a stem, so if they are not placed on the side you want, in some cases forcing the "stem" direction will produce what you want.

The list can include symbols from the following table:

Symbol	**Meaning**
.	staccato
−	legato
>	accent
^	accent

A quoted string can also be specified (e.g., "ff", "adagio", etc.). This will be printed in the font and size specified by the "withfontfamily," "withfont," and "withsize" parameters. Any of the special music characters can be printed by using the usual convention, e.g., \(ferm) for a fermata. (This is described in a later section.) However, it is usually preferable to use the printing of text strings relative to beats. This will be described in a later section.

Here are some examples:

```
[with ., -]
[with ^, "\(ferm)"]
[with "sfz"; cue]
```

If the symbols acc_hat, ferm, or wedge are used by themselves in a "with" list item, they are handled specially. If the stem direction is such that the upside-down versions of these characters should be used, the upside-down version will be used.

At the end of the list, you can specify "above" or "below" to force a side. Otherwise Mup will normally place them on the notehead side, unless there is more than one voice.

```
[with . below; with sfz above]
```

5.1.6.4 Slashes

Diagonal lines to be drawn through the stem of the group can be specified using

```
slash N
```

where *N* is the number of slashes to draw. In the case of grace notes, *N* can only be 1, which will produce a grace note with a slash through it. In the case of other notes, *N* can be 1 or greater, and is used to specify tremolo or repetition of the note group. Examples:

```
[slash 2]
[grace; slash 1]
```

5.1.6.5 Stem direction

The chord stem direction can be specified as "up" or "down". Normally Mup chooses the stem direction, but once in a while you may want to override its choice. There are some restrictions. All chords beamed together and the pair of chords in an alternation pair (described later) must have the same stem direction. Examples:

```
[up]
[down]
```

For voice 3, stem direction works a little differently. By default, the stem direction for voice 3 is up, but for voice 3 on any given staff, when you specify a stem direction, that direction will remain in effect until explicitly changed. Thus, for example, if you want all voice 3 stems on a given staff in an entire song to be down, you only need to specify "[down]" on the first chord. Grace notes on voice 3, however, are always stem up unless explicitly forced down.

5.1.6.6 Stem length

Normally, Mup sets stem lengths as appropriate, but sometimes you might want to make a stem longer or shorter than normal. This is done with the keyword "len" followed by a length in stepsizes. Some examples:

```
[ len 14 ]
[ len 7; up ]
```

You cannot specify stem length on chords inside of beams. The len can be set to 0 to produce a notehead with no stem at all.

There is a stemlen parameter that can be used to set the default stem length. If you don't want any stems, setting that parameter to 0 is more convenient than setting len to 0 on each chord individually.

There are certain circumstances in which Mup will normally shorten stems slightly on beamed chords or on chords whose stems protrude from the staff. There is a stemshorten parameter that allows you to control how Mup handles those cases.

5.1.6.7 Padding

Sometimes it may be desirable to space notes somewhat differently than Mup would normally place them. It is possible to specify "padding" before any note group. This is done by specifying the word "pad" followed by a number of stepsizes of white space padding to add before the chord. The number can be fractional, as in 0.5 or 2.25. For example:

```
[ pad 2.5 ]
```

The padding can also be negative. Padding affects the minimum amount of width allocated to a chord, so you can use positive padding values to force additional room in front of a chord, or negative padding to allow things closer together.

5.1.6.8 Chord location tag

It is possible to set a location tag which is associated with an entire chord rather than an individual note. This tag is specified like location tags for notes, with an "=" followed by a name, but is inside the square brackets rather than after a note.

```
3: 2cf; [=h] egc+;
1: [cue; with >; =_ch] fa;
```

5.1.6.9 Horizontal offset

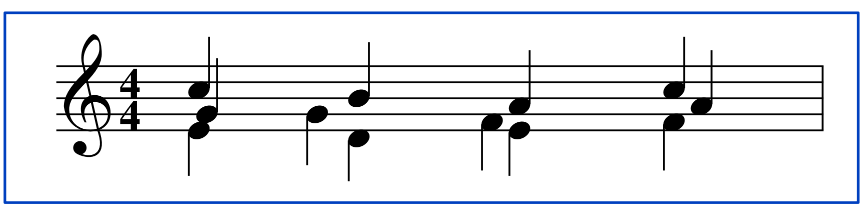
A horizontal offset can be applied to a chord by specifying "ho" followed by a number of stepsizes. The number can be positive or negative, and can include a fractional part. A positive number will cause the chord to be placed that far right of where the horizontal center of the chord would normally be; a negative number will place it that far to the left. It is also possible to simply specify a "+" or "-" which says to place the chord to the right or left, as close as possible without running into something.

A horizontal offset is most commonly used on the third voice of a staff, although it can be used on other voices too. By default, Mup places third voice chords as close as it thinks is safe for avoiding most collisions. Sometimes it may actually be okay visually to place the chord a little closer than that, or perhaps there is a better "nook" to tuck it into on the left side. By specifying a horizontal offset, you can place chords anywhere you want. Mup makes no checks to avoid collisions when you specify a horizontal offset, so this attribute needs to be used with some care.

The ho can be used on grace groups, and has generally similar effects. It only has an effect if there are multiple voices having grace notes at the same point in time, and specifies how to move the grace note relative to where it would be if no ho had been specified.

Here is an example of the use of horizontal offset and stem direction on voice 3.

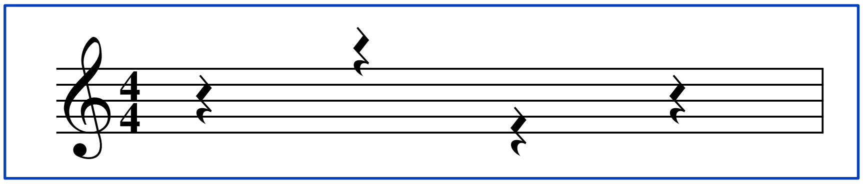
```
score
  vscheme=3f

music
1 1: c+;b;a;c+;
1 2: e;d;e;f;
1 3: [ho+1] g; [ho-5.2; down]; [ho-] f; [ho+; up] a;
bar
```



5.1.6.10 Rest distance

Usually Mup's placement of rests is satisfactory, but once in a while you might want to force a rest to be placed a little higher or lower than Mup would place it. You can specify a "dist" which is the number of stepsizes from the center of the staff to place the rest symbol, with positive numbers being above the center and negative numbers below. If the staff is transposed, the placement of the rest symbol will be adjusted accordingly.

```
// The first and last rests are in normal position.
// The second is forced higher, the third is forced lower.
1: r; [dist 6]; [dist -4]; r;
bar
```

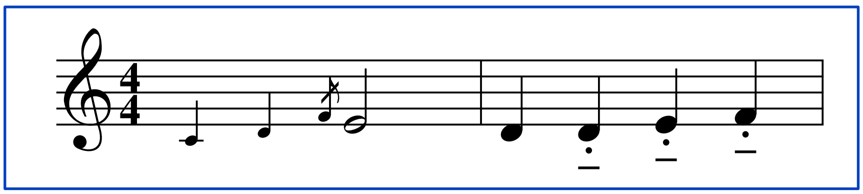


Note that the alignrests parameter can also be used to affect how rests are placed.

5.1.6.11 Repeated attributes

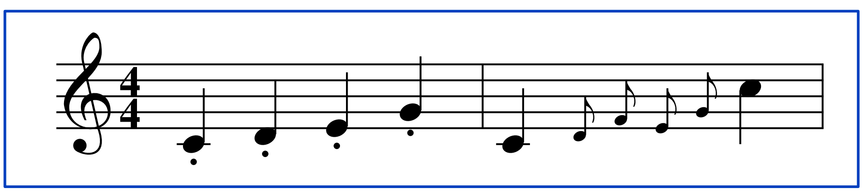
If two or more chords in a row have the same bracketed attributes, there are two shorthand notations. Specifying an empty set of brackets "[]" means to use the same attributes as were specified on the previous chord. The previous chord must be within the same measure. Some examples:

```
1: [cue] 4c;[]d;[grace; slash 1] 8f;2e;
bar

1: d; [with .,-]d; []e; []f;
bar
```



The other shorthand is to put 3 dots after a set of bracketed items. This means to repeat the same bracketed items on all chords for the rest of the measure, or until a new set of bracketed items is specified. Items in a "with" list are omitted on rests, since usually they are things that don't make sense for rests. If you wish to cancel this before the end of a measure but without specifying a new set, the special set "[-]" can be used.

```
// put staccato marks on all notes
1: [with .]... c; d; e; g;
bar
// put several cue size notes in the middle of a measure
1: c; [cue]... 8d; f; e; g; [-] 4c+;
bar
```

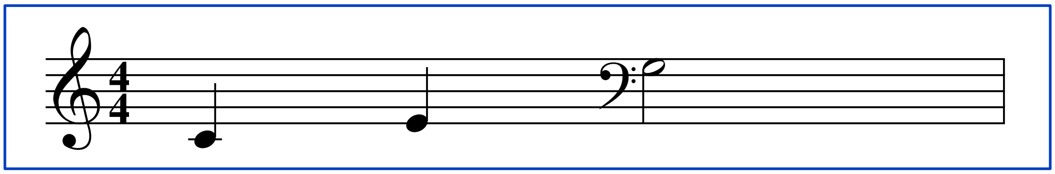


You can use more than one set of brackets on a single chord. For example "[len 5][down]" and "[len 5;down]" are equivalent. However, [-] can only be used by itself, and if you want to use [] along with another bracketed set of items (to duplicate the items on the previous chord plus add some more), the empty [] needs to be first, before the other set of items.

5.1.7 Mid-measure parameter changes

The group attributes can optionally be preceded by mid-measure parameter changes. This is not used very often, but the most common use is to change the clef in the middle of a measure. There are only a few other parameters that can be changed inside a measure: alignrests, defoct, release, and vcombine. The change is enclosed in double angle brackets. After the opening angle bracket is the context to which the changes apply (score, staff, or voice), followed by one or more parameter changes.

```
1: c; e; <<score clef=bass>> 2g;
bar
```



If there is more than one parameter listed, they are separated by semicolons.

```
<< staff clef = tenor ; defoct = 4 >>
```

If you wish to change parameters in more than one context, multiple angle-bracketed items can be given:

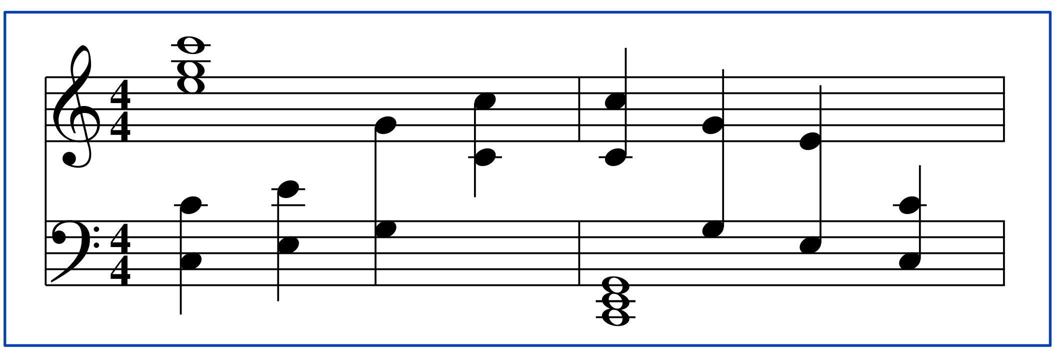
```
2-3 2: 2c; <<score release=100>><<staff clef=alto>><<voice defoct=3>> c;
```

If you specify "staff" or "voice" context, the specific staff(s) or voice(s) to which the changes apply is based on which staff(s) or voice(s) are being described on the input line. So in the example above, the clef is changed to alto on staffs 2 and 3, and the defoct is changed to 3 for voice 2 on both of those staffs. The release change applies to the entire score.

5.1.8 Cross-staff stems

Sometimes, on music for instruments that use more than one staff, you may want some of the notes of a chord to be printed on the staff above or below, rather than using a lot of ledger lines. You can specify this by putting the word "with" before the notes that are to go on the other staff, and following those notes with "above" or "below" to specify which is the other staff. For example:

```
score
    staffs=2
staff 2
    clef=bass
music

1: 1e+g+c++;
2: cc+; ee+; g with g+ above; with c+c++ above;
bar

1: cc+; g with g- below; e with e- below; with cc- below;
2: 1c-e-g-;
bar
```



The notes for the other staff have to follow the notes on the normal staff in the input. As is shown in the example, it is possible to have *all* the notes on the "other" staff, if you wish. The octave is specified as if the note were on the normal staff; Mup will automatically adjust appropriately for the other staff's clef. Once in a while, Mup may not be able to figure out how to completely avoid colliding with other notes; in that case you can use the horizontal offset that was described earlier. Or, if the collision is with a beam on the other staff, the collision may be avoided by forcing the stems in that beamed set to point the other direction (using '[up]' or '[down]') to move the beam to the other side.

For MIDI purposes, only the normal staff's key signature and accidentals are considered, so if the other staff has a different key signature or accidentals that should really apply to these notes, you will have to supply accidentals explicitly.

5.1.9 Inter-chord attributes

In addition to the chord attributes that can appear in the square brackets before time and pitch information, there are a few attributes that are specified after the time and pitch information. These are attributes that start or end on the chord but also affect other chords. There are several such attributes:

— chord ties

— chord slurs

— custom beaming

— cross-staff beams

— alternation

— slope

— phrase

If several of these are specified on a single chord, they may be in any order, separated by commas.

5.1.9.1 Chord ties

If all notes in a chord are to be tied to the following chord, the keyword "tie" can be placed at the end of the chord. As was mentioned earlier, individual notes can be tied using a "~" symbol. Thus:

```
ceg tie;
```

is equivalent to

```
c~e~g~;
```

The word "tie" may be preceded by the word "dotted" or "dashed" to produce dotted or dashed ties. Otherwise, normal, solid ties are drawn. The word "tie" may be followed by the word "up" or "down" to specify the direction of each curve's bulge. If neither is specified, Mup will determine an appropriate direction, so you only need to give a direction if you wish to override Mup's choice. After the aforementioned items (if any), you may put "to voice *N*" to tie to the chord in voice *N* rather than in the current voice.

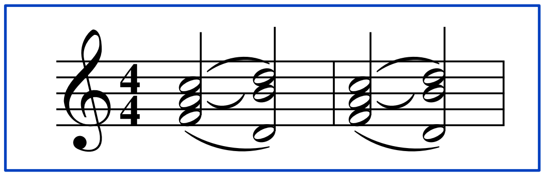
You can also produce the effect of tying chords by using additive time values. For example, the following lines produce the same output:

```
1: 2ceg tie;8;;4;  // explicit tie

1: 2+8ceg;8;4;   // tie implied by added time values
```

5.1.9.2 Chord slurs

The keyword "slur" can be placed at the end of a chord to indicate that each note in the chord is to be slurred to the corresponding note in the following chord. In other words, the top note of the chord is slurred to the top note of the following chord, the second from the top note in the first chord to the second from the top note in the second chord, etc. Thus the following 2 measures give equivalent output:

```
1: 2f<d> a<b> c+<d+>; dbd+;
bar

1: 2fac+ slur; dbd+;
bar
```

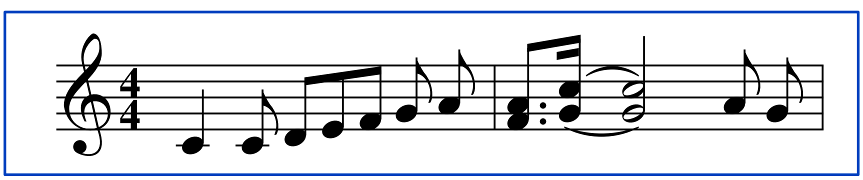


The chord with the "slur" keyword and the chord that follows it must have the same number of notes. The word "slur" may be preceded by the word "dotted" or "dashed" to produce dotted or dashed slurs. Otherwise, normal, solid slurs are drawn. The word "slur" may be followed by the word "up" or "down" to specify the direction of each curve's bulge. If neither is specified, Mup will determine an appropriate direction, so you only need to give a direction if you wish to override Mup's choice. After the aforementioned items (if any), you may put "to voice *N*" to slur to the chord in voice *N* rather than in the current voice.

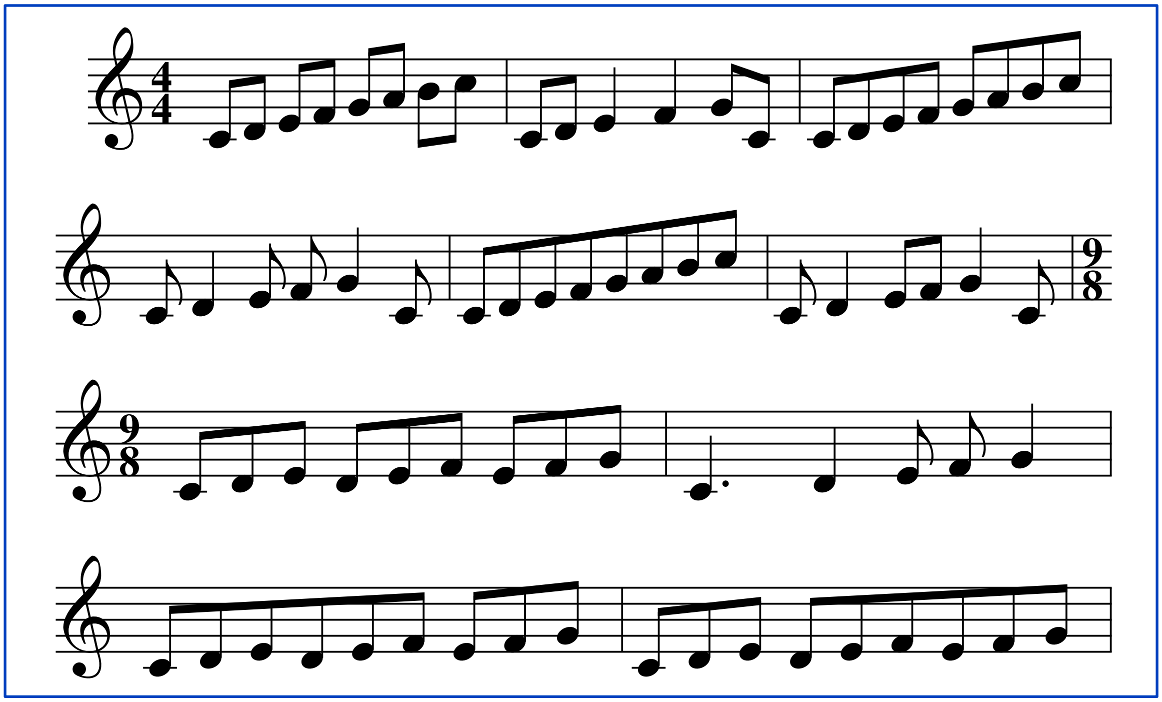
5.1.9.3 Custom beaming

Normally, notes of eighth or shorter duration are automatically beamed according to the specification of the "beamstyle" parameter. Occasionally, you may wish to override the default beaming style for a particular situation. This is done using the "bm" and "ebm" keywords. The "bm" (short for "beam") is placed at the end of the chord that is the first to be beamed. The "ebm" (short for "end beam") is placed at the end of the last chord. Both chords must be in the same measure. If there is any custom beaming specified for a given voice in a given measure, only what you explicitly specify to be beamed will be beamed. An example:

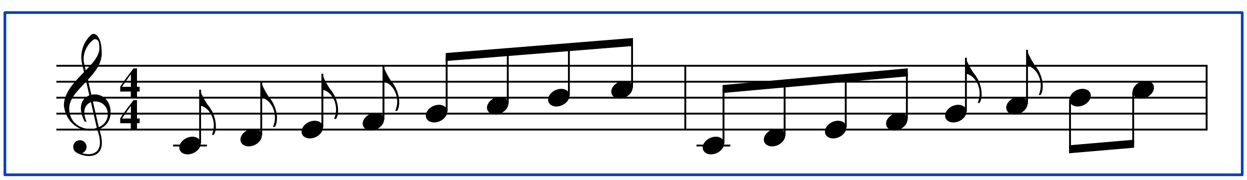
```
// The d, e, and f will be beamed together,
// but other 8th notes will not be.
1: 4c; 8; d bm; e; f ebm; g; a;
bar

// First two chords beamed together.
// Second chord is tied to third chord.
1: 8.fa bm; 16gc+ tie, ebm; 2; 8a; g;
bar
```



In the absence of custom beaming, Mup will beam notes together using the beamstyle parameter, if that parameter is set. The beamstyle parameter is a list of time values that add up to a measure. Each time value tells how many chords to beam together. For example, a 2 means to beam a half note worth of chords together, whereas 1.. would indicate that a double dotted whole note worth of chords should be beamed together. Here are some examples of how the beamstyle parameter works.

```
// beam each quarter note worth of notes together,
// breaking the beaming at each quarter note boundary
score beamstyle = 4,4,4,4
music
1: 8c;d;e;f;g;a;b;c+;
bar
1: 8c;d;4e;f;8g;c;
bar

// beam each half note worth of notes together,
// breaking the beaming at each half note boundary
score beamstyle = 2,2
music
1: 8c;d;e;f;g;a;b;c+;
bar
// the middle two eighth notes will not be beamed together,
// because they are on opposite sides of the half note boundary
1: 8c;4d;8e;f;4g;8c;
bar

// beam each whole note worth of notes together
score beamstyle = 1
music
1: 8c;d;e;f;g;a;b;c+;
bar
1: 8c;4d;8e;f;4g;8c;
bar

// in 9/8 time, beam each dotted quarter note worth of notes together
score time = 9/8 ; beamstyle = 4., 4., 4.
music
1: 8c;d;e;d;e;f;e;f;g;
bar
// the eighth notes will not be beamed together,
// because they are on opposite sides of the dotted quarter boundary
1: 4.c;4d;8e;8f;4g;
bar

// in each measure, beam the first dotted half worth of notes together,
// then beam the remaining dotted quarter worth of notes together
score beamstyle = 2., 4.
music
1: 8c;d;e;d;e;f;e;f;g;
bar

// in each measure, beam the first dotted quarter worth of notes together
// then beam the remaining dotted half worth of notes together
score beamstyle = 4., 2.;
music
1: 8c;d;e;d;e;f;e;f;g;
bar
```

It is possible to apply the automatic beaming to just a portion of a measure, by specifying abm ("automatic beaming") on the first group where it is to be applied, and eabm ("end automatic beaming") on the last such group.

```
score beamstyle=2,2
music

1: 8c;d;e;f;g abm;a;b;c+ eabm;
bar

1: 8c abm;d;e;f eabm;g;a;b bm;c+ ebm;
bar
```



The value of the beamstyle parameter is remembered for any later changes back to the same time signature. For example, suppose you set

```
time=4/4
beamstyle=4,4,4,4
```

then later in the piece switched to

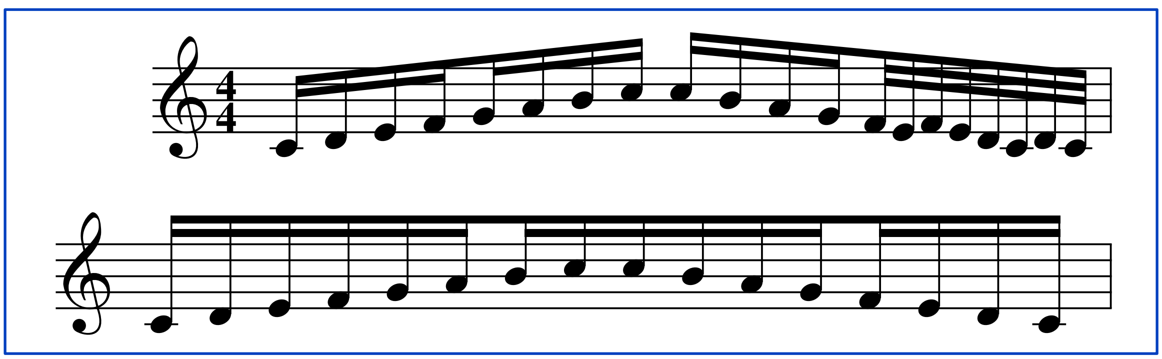
```
time=3/4
beamstyle=4,4,4
```

Then any time you went back to 4/4 or 3/4, the beamstyle you had set for that time signature would automatically be set as well. You could, of course, override the automatic setting with a new beamstyle if you wished.

Normally Mup will break beams whenever it encounters a rest or space, but if you specify an "r" at the end of the beamstyle parameter, it will beam across rests of eighth note or shorter duration. Similarly, specifying an "s" at the end of the beamstyle parameter will cause it to beam across spaces of eighth note or shorter. Specifying both (in either order) will result in beaming across both. Mup will also beam across eighth note or shorter rests or spaces inside of custom beams.

Sometimes, if there are many short notes beamed together, you may wish to subdivide the beams into smaller groupings, where the outer, or primary beam remains unbroken, but the inner, or secondary beams are broken periodically. When using custom beaming, this is specified by "esbm" ("end subbeam" or "end secondary beam") on the chord after which you want the break to occur. When using beamstyle, parentheses are used to indicate what sets of secondary beams are to be included in a given outer beam.

```
score
beamstyle=(4,4),(4,4)
music

// use the beamstyle parameter
1: 16c;d;e;f; g;a;b;c+; c+;b;a;g; 32f;e;f;e;d;c;d;c;
bar

// use custom beaming
1: 16c bm;d;e;f;g;a esbm;b;c+;c+;b;a;g esbm;f;e;d;c ebm;
bar
```

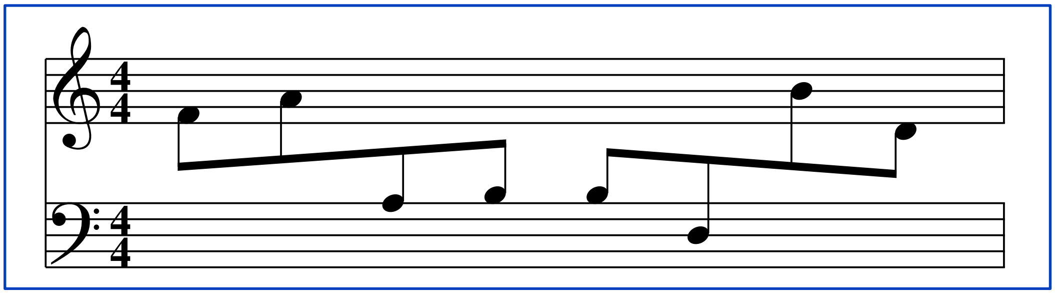


5.1.9.4 Cross-staff beams

In keyboard music, sometimes notes on adjacent staffs are beamed together. Mup will do this with a variation on custom beaming. A bm and ebm must be specified on both staffs, and in each case the bm is followed by a qualifier: on the first of the two staffs, "bm with staff below" must be specified, while on the second staff you use "bm with staff above". For every point in time for the duration of the beam, one staff must have a chord with notes in it, and the other staff must have a space chord. (This is somewhat different than ordinary, non-cross-staff beams controlled by the beamstyle parameter, where spaces are not allowed unless beaming across spaces is specifically requested.) The two "bm" marks must occur at the same time in the measure. Similarly, the two "ebm" marks must occur at the same time in each staff. Grouping subbeams using "esbm" is not supported on cross-staff beams.

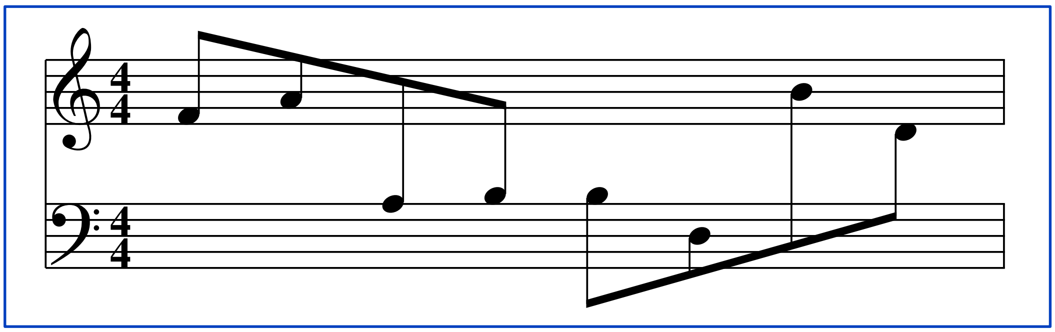
Some examples:

```
1: 8f bm with staff below; a; 4s ebm; 4s bm with staff below; 8b; d ebm;
2: 4s bm with staff above; 8a; b ebm; b bm with staff above; d; 4s ebm;
bar
```



Normally the beam will be drawn between the staffs, but you can force the beam to be above or below all the notes by specifying a stem direction. You can also adjust the appearance of the beam by giving stems lengths for the first and last chords in the beam.

```
1: [up]8f bm with staff below;a;4s ebm;4s bm with staff below;8b;[len 11]d ebm;
2: 4s bm with staff above;8a;b ebm;[down;len 14]b bm with staff above;d;4s ebm;
bar
```



If a cross-staff beam includes tuplets, the tuplet numbers will not be printed. You would have to print them yourself using either a "print" statement or a "boldital" statement.

It is possible for cross-staff beams to collide with other items, such as dynamic marks. There may be undesired interactions between cross-staff beams and cross-staff stems. In these cases, you may need to move other items. Another thing you might try is specifying stem lengths to alter where the beams get placed, or printing a blank string between the staffs to cause them to get placed further apart. If you specify a slope, you have to specify it on the staff having notes in the first chord, not the staff with space. Slashes are not allowed on cross-staff beams.

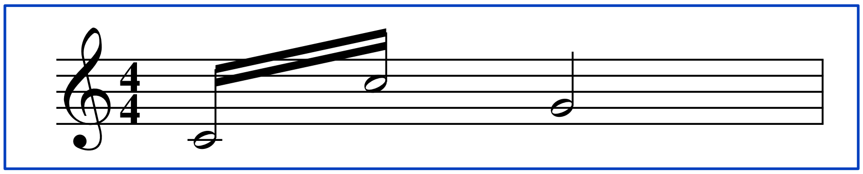
Mup doesn’t directly allow cross-staff grace note beams. However, you may be able simulate the effect by using cue note chords along with invisible time signature changes and possibly invisible bar lines. See the section on "Special uses of invisbar" for more details.

You may want to also look at the section on cross-staff stems, for an alternate way to handle some cases where you might use cross-staff beams.

5.1.9.5 Alternation

Alternation pairs can be specified using "alt *N*" where *N* is a number. An alternation pair is two chords that are to be played alternately in quick succession but are not written out as such. This is shown by drawing *N* beams between the stems of the chords. The note value you specify for each chord must be the same, and the time value of each must equal the time taken by the pair. For example, if the pair takes up the time of a half note, each chord would be a half note. Here is an example:

```
// Alternate between c and c an octave
// higher. Total time taken is that of
// a half note. Two "beam-like" lines
// will be drawn to show the alternation.
1: 2c alt 2; 2c+;2g;
bar
```

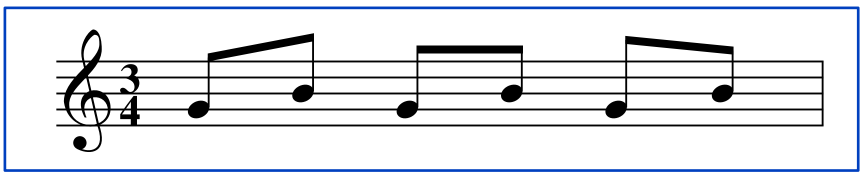


Alternation is not allowed on cross-staff beams.

5.1.9.6 Slope

On the first chord of a set of chords that are beamed together, you can specify a beam angle from -45 to 45 degrees. This will override whatever angle Mup would have used.

```
1: 8g slope 11; b; g slope 0; b;  g slope -5.75; b;
bar
```



5.1.9.7 Phrase marks

Phrase marks can be specified by putting "ph" on the chord where you want the phrase to begin, and "eph" on the chord where you want it to end. The ph can optionally be preceded by "dotted" or "dashed" to specify the line type, and optionally followed by "above" or "below" to specify the side for the phrase mark. The "ph" and its matching "eph" do not have to be in the same measure. There is also an alternate way to specify phrase marks, described in the section on Tempo, Dynamic Marks, Ornaments, etc.

5.1.10 Tuplets

Tuplets are specified by giving a list of chords within curly braces. The ending brace is optionally followed by a side ("above" or "below"). This is followed by a number or pair of numbers, and a semicolon.

The "side" tells Mup whether to print the tuplet number and bracket above or below the chords. It only applies on voice 1 or 2 when the other voice (1 or 2) is nonexistent or a space. If there are at least two voices, the number and bracket will always be put above on voice 1 and below on voice 2. You can always force the side on voice 3. If you don't specify a side for a tuplet, Mup will choose the side that seems best.

The first number is the number that should be printed with the tuplet. If it is followed by an "n," the number (and bracket) will not actually be printed. If it is followed by a "y," the number and bracket will always be printed, unless there is only a single chord in the tuplet, in which case only the number will be printed. If it is followed by "num," the number will always be printed, but the bracket will never be printed. If none of those modifiers are specified, the number will always be printed, but the bracket will be omitted in cases where all the notes in the tuplet are beamed together and the beam does not include any notes not in the tuplet.

The second number, if any, is separated from the first by a comma, and tells the time unit that the tuplet is to take up. This number can be dotted if necessary. It can also be a time expression, like 2+8, although that is very rarely likely to be useful. If no second number is given, the default is to fit in the next shorter un-dotted note. For example, three eighth notes would be fitted into the time of a quarter note, or five eighth notes would be fitted into the time of a half note.

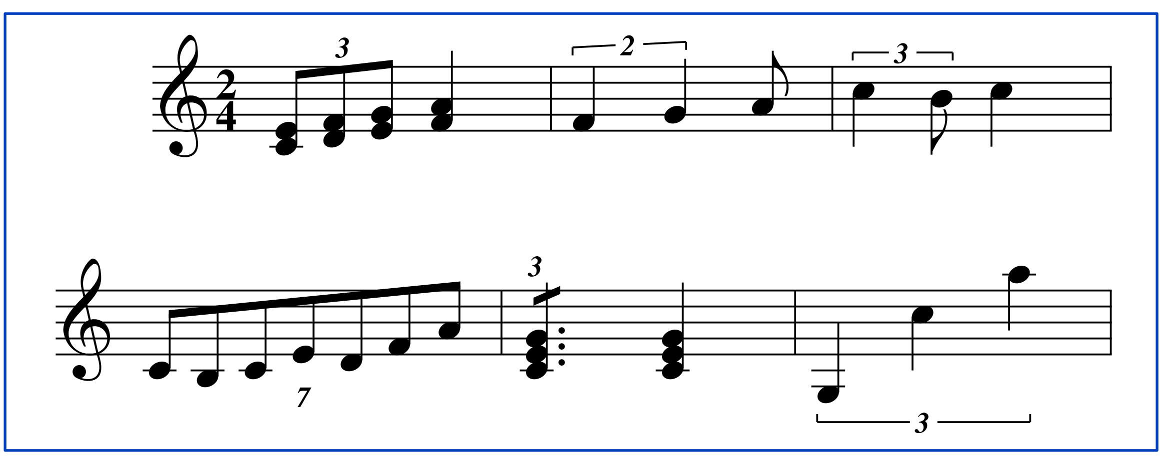
Occasionally in music, as a shorthand, a single note or chord is printed with a tuplet number to indicate the note or chord is to be repeated several times as a tuplet. Mup will allow this; simply use a tuplet with only one chord, and add one or more slashes.

The final item that can optionally be specified is the word "slope," followed by an angle in degrees, for the the angle of the tuplet bracket, if any. If that is omitted, the angle is calculated based on the tupletslope parameter.

Nested tuplets are not allowed.

Here are some examples of tuplets:

```
// This has a triplet, where 3 eighth
// notes take as much time as a
// normal quarter note.
1: { 8ce; df; eg;}3; 4fa;
bar

// This has a dublet, in which 2 quarter notes
// take the time of a normal dotted quarter
1: { 4f; g; } 2, 4.;8a;
bar

// A quarter note and eighth note that
// make up a triplet the length of a
// normal quarter note,
// with the 3 printed above the notes
1: { 4c+; 8b; } above 3; 4c+;
bar

// A septuplet in the time of a half note
// with the "7" printed below the notes
1: { 8c; b-; c; e; d; f; a; } below 7;
bar

// A single chord tuplet
1:  { [slash 1] 4.ceg;}3; 4;
bar

// Forcing a tuplet bracket to be flat
1: {g-;c+;a+;}3 slope 0;
bar
```



5.2 Chord-at-a-time input style

There is an alternate input style, in which you enter music a chord-at-a-time, rather than a voice-at-a-time. In this style, the specification before the colon gives one or more patterns that tells how to map notes to staffs and voices.

Suppose you want to print some music in a style common for many traditional hymns: it is to be printed on two staffs, each staff will always have exactly two notes, and the rhythm is exactly the same for all voices. With chord-at-a-time input, you specify, for each note in a chord, which staff and voice to map that note. Since there are four notes in each chord, there will be four mappings listed. You want the bottom two notes to get mapped to staff 2, and the top two notes to staff 1. This is shown as

```
// Bass to staff 2 voice 1
// |    Tenor to staff 2 voice 1
// |    |    Alto to staff 1 voice 1
// |    |    |    Soprano to staff 1 voice 1
[  2 1; 2 1; 1 1; 1 1 ]
```

Each item in the semicolon-separated list tells how to map one note. So the first note in each chord will get mapped to staff 2 voice 1. The second note in each chord will also get mapped to staff 2 voice 1. The third and fourth notes will get mapped to staff 1 voice 1.

Since voice 1 is, as always, the default, this could be written more compactly as just:

```
[ 2; 2; 1; 1 ]
```

If you wanted to input notes from top to bottom instead (in soprano-alto-tenor-bass order, rather than bass-tenor-alto-soprano order), you could use:

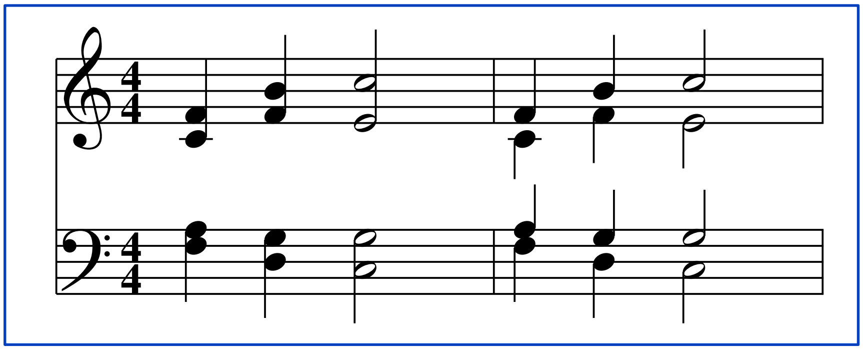
```
// Soprano
//    Alto
//       Tenor
//          Bass
[  1; 1; 2; 2 ]
```

If you wanted to use two separate voices on each staff (going back to bottom-to-top order), you could specify:

```
// Bass to staff 2 voice 2
// |    Tenor to staff 2 voice 1
// |    |    Alto to staff 1 voice 2
// |    |    |    Soprano to staff 1 voice 1
[  2 2; 2 1; 1 2; 1 1  ]
```

Now let's put these mappings with music data.

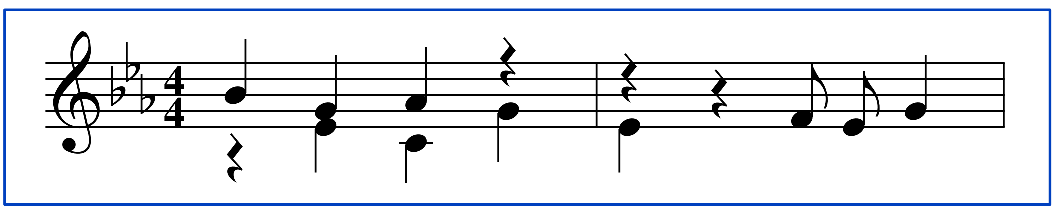
```
score
    staffs=2
    vscheme=2f
staff 2
   clef=bass

music

[ 2; 2; 1; 1 ] : facf;dgfb;2cgec+;
bar

[ 1 1; 1 2; 2 1; 2 2 ] : fcaf;bfgd;2c+egc;
bar
```



In the first measure, the first two notes listed in each chord are mapped to staff 2, voice 1, while the third and fourth notes listed in each chord are mapped to staff 1 voice 1. In the second measure, two voices are used and notes are entered in descending order. Note that each note takes on the correct default octave for whichever staff it is mapped to.

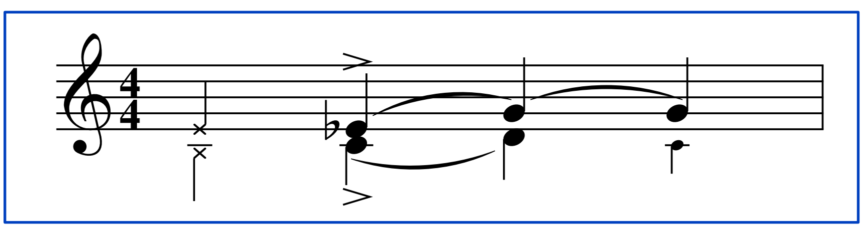
It is also possible to use rests or spaces for some of the voices.

```
score
  key=3&
  vscheme=2f

music

[ 1 2; 1 1 ]: rb; eg; ca; gr;
bar

[ 1 2; 1 1 ]: er; sr; 8sf; se; 4sg;
bar
```



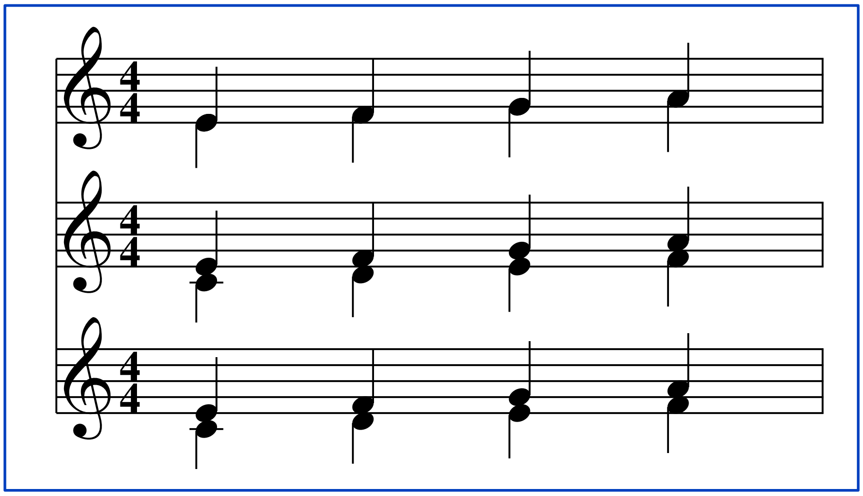
Chord attributes and interchord attributes (like tie, slur, xnote, len, and alt) can be specified just like for voice-at-a-time input. Note attributes (like ? and ˜) apply to the note wherever it gets mapped, and items that apply to the chord as a whole will be applied to all the notes.

```
[1 2; 1 1] : [cue; xnote; len 6] b-e; [with > ] ce& slur; dg˜; c?g;
```



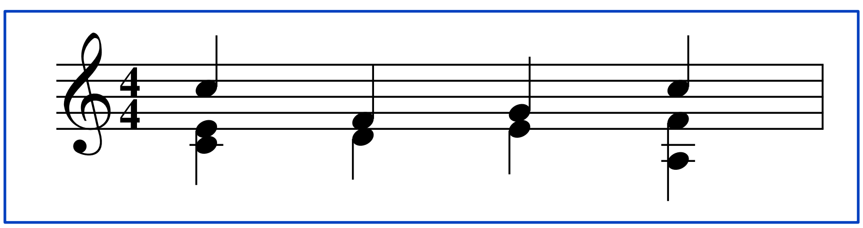
It is possible to map a note to more than one place by using ranges, and/or by giving a list of staffs and voices, separated by ampersands. This may be useful, for example, if several voices are in unison. In the next example, the first note in each chord will be mapped to voice 1 of staffs 1 through 3, as well as to voice 2 of staff 1, while the second note in each chord will be mapped to voice 2 of staffs 2 and 3.

```
[ 1-3 1  &  1 2;  2-3 2 ] : ec;fd;ge;af;
```



It is also possible to specify more than one bracketed mapping. Each must include a mapping for a different number of notes. So, for example, if some chords in a measure have two notes and others have three, you can define two maps: one for two notes, and one for three. The example below demonstrates placing alto and soprano as two voices on one staff, but sometimes the alto part splits.

```
// For chords with two notes,
// map the first to staff 1 voice 2 (alto),
// and the second to staff 1 voice 1 (soprano).
// For chords with three notes,
// map the first two notes to staff 1 voice 2
// (first and second alto part),
// and the third to staff 1 voice 1 (soprano).
[ 1 2; 1 1 ] [ 1 2; 1 2; 1 1 ]: cec+;df;eg;a-fc+;
```



If mappings of different chords need to vary by something other than the number of notes in the chord, then you will have to use the voice-at-a-time input style.

Since the mapping specifications can get rather complex, and they may be used many times during a song, it is usually best to define macros for them, and possibly even put the macro definitions in an "include" file. Macros and include files are described in a later section.

You can use the different input styles in different measures of a single song, and use different mappings in different measures. You can even mix the two input styles within a measure, but a given staff/voice can only appear on one line of input per measure. So, for example, you could choose to input staffs 1 and 2 of a song in chord-at-a-time input style, and staff 3 in voice-at-a-time style. The noteinputdir parameter is ignored on chord-at-a-time input. Location tags cannot be used with chord-at-a-time input, since often that would be ambiguous. Mid-measure parameter changes are not supported on chord-at-a-time input.

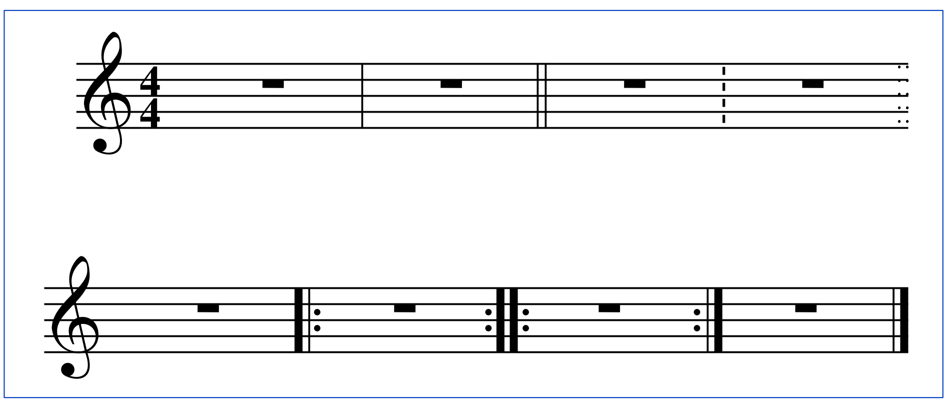
5.3 Bar lines

Each measure must be ended with a "bar" of some kind. There are several kinds:

Keyword	Meaning
bar	ordinary bar line
dblbar	double bar
repeatstart	beginning of repeated section
repeatboth	end of one repeated section and beginning of another
repeatend	end of repeated section
endbar	heavy double bar line used at end of song
invisbar	no bar line printed
restart	end staffs and begin anew

In addition, the bar and dblbar types can be preceded by the word "dashed" or "dotted" to produce dashed or dotted bar lines.

```
1: mr;
bar
1: mr;
dblbar
1: mr;
dashed bar
1: mr;
dotted dblbar
1: mr;
repeatstart
1: mr;
repeatboth
1: mr;
repeatend
1: mr;
endbar
```



When a repeatstart would end up at the end of a score, it will automatically get moved to the beginning of the following score. In that case, it isn't clear what kind of bar line you would like to have put at the end of the original score. By default, Mup will use an ordinary bar, but in some cases—like if the repeatstart begins a new section, or there is also a key change—you might want to have a dblbar instead. You can get a dblbar there by specifying dblbar in parentheses before the repeatstart:

```
(dblbar) repeatstart
```

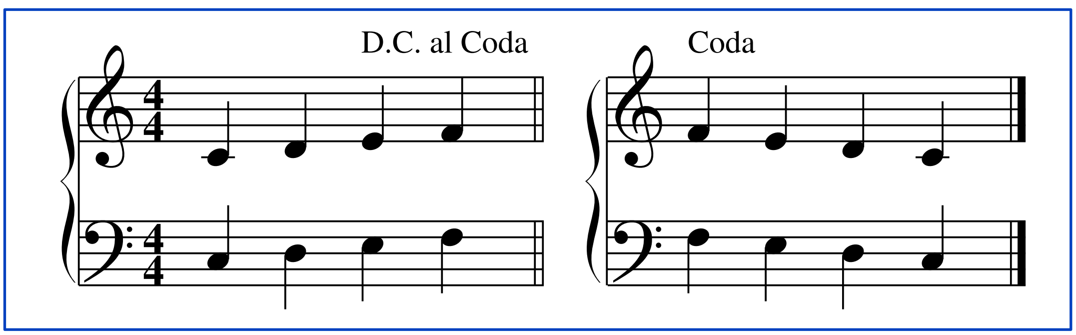
Actually, you are allowed to specify any bar type in the parentheses, including dashed or dotted types, but dblbar or possibly invisbar are probably the only ones likely to be useful. If the repeatstart ends up in the middle of a score, so that it doesn't need to get moved to the beginning of the next score, the parenthesized bar type is not needed, so it is just ignored.

There is a bracketrepeats parameter that can be set that will cause Mup to draw brackets around repeated sections, to make them more obvious to the performer, which may be helpful when playing in dimly lit areas.

Sometimes music is printed without bar lines. The "invisbar" can be used in this case, to fulfill Mup's requirement to specify some sort of bar after every measure without actually printing bar lines.

The restart is a special kind of bar. It follows immediately after another bar line without any intervening music data. (That is, you can't have any notes. You can use score or staff context things, like changing time or key signature.) It would be most commonly used for something like a short coda.

```
score
    staffs=2
    brace=1-2
staff 2
    clef=bass
music

1-2: c;d;e;f;
rom above 1: 3 "D.C. al Coda";
dblbar

// The restart follows a bar
// without any music data in between.
restart

rom above 1: 1 "Coda";
1-2: f;e;d;c;
endbar
```



There are several optional directives that can follow the bar line keyword. They can be specified in any order and are described below.

5.3.1 Bar line padding

One optional directive is padding. It can be used to force Mup to place extra white space to the left of the bar line. It is specified by the keyword "pad" followed by a number of stepsizes of padding to add. For example:

```
dblbar pad 3
```

would add 3 stepsizes of padding. One possible use would be if you wanted a different amount of gap on a restart than what Mup uses by default:

```
restart pad 10
```

5.3.2 Bar line location tag

It is also possible to associate a location tag with a bar line. For example:

```
// Associate tag "_bar6" with bar line
bar =_bar6

// Do double bar with an extra stepsize of
// padding, and associate tag "q"
// with the bar line
dblbar pad 1 =q
```

The y coordinate of a bar line is not very useful. Special rules apply if a bar line happens to be placed at the end of a score. Any locations taken relative to the bar that would be to the right of the bar are treated as if the bar line were at the beginning of the following score, just beyond the clef and time and key signatures, if any.

5.3.3 Endings

First and second endings, etc. can be designated at bar lines. This is done with the keyword "ending," followed by a quoted string to use as the label for the ending that should begin at the bar line. An ending will span bars until either another ending is specified, the piece ends, or the special keyword "endending" is used. Examples:

```
bar ending "1."
repeatend ending "2-3"
endbar endending
```

The ending label will always be forced into 12-point Times roman font. If you change font or size or include special characters in the ending label, the output is not likely to be aligned properly.

The "endingstyle" parameter (described in the section on parameters) controls where endings are placed. Endings cannot start at or cross over a restart bar.

When doing an endending, Mup will draw a vertical line at the end unless it is at a plain bar or an invisbar that is not at the end of the piece. Once in a while, you may want to override that. To do so, instead of specifying endending, you can specify openendending (to force Mup to not draw the final vertical), or closedendending (to force it to draw the final vertical).

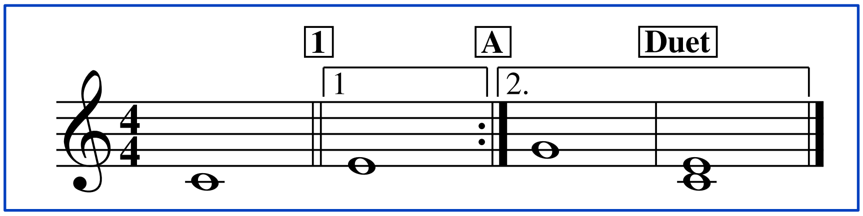
5.3.4 Rehearsal marks

Rehearsal letters or numbers can be specified on any bar line. There are four formats:

rehearsal let
rehearsal num
rehearsal mnum
rehearsal "*label***"**

In the first example, a rehearsal letter will be placed on the bar. The first occurrence of this will become rehearsal "A", the next "B", and so forth. The second format works in a similar fashion except that numbers are used rather than letters. With the third format, the current measure number is used. With the last format, any arbitrary string within the quotes will be used. The keyword "rehearsal" can be abbreviated to "reh" if you wish. Only one rehearsal mark is allowed on any one bar, but the types can be mixed throughout the composition. Note, however, that mixing "num" with "mnum" is likely to be very confusing, as would using "num" while the measnum parameter is set to "every *N*." The rehearsal marks can be intermixed with other bar options:

```
1: 1c;
dblbar reh num ending "1"
1: 1e;
repeatend pad 1 =_xyz reh let ending "2."
1: 1g;
bar rehearsal "Duet"
1: 1ce;
endbar endending
```



The rehearsal marks are normally put directly above the bar line. However, if the bar line falls at the end of a score, the rehearsal mark will be placed at the beginning of the following score, after the clef, key signature, and time signature. The "endingstyle" parameter (described in the section on parameters) is used to determine which staffs get rehearsal marks. The "rehstyle" parameter specifies whether to put rehearsal marks inside a box or a circle or leave them plain.

By default, rehearsal marks are printed in 12-point Times bold, but the rehearsal keyword may be followed by a specification for *fontfamily*, *font*, and/or **(***size***)**. Once specified, these remain in effect for any future rehearsal marks, until explicitly changed. Some examples:

```
bar rehearsal helvetica bold (14) let
repeatend reh newcentury num
invisbar reh rom (10) mnum
```

After all the options listed above, you may specify "dist *N*" where *N* is some number of stepsizes. This will override the dist parameter for determining how close to the staff to place the rehearsal mark. If the number is followed by a "!" Mup will place the mark exactly that far from the top of the staff, even if it overwrites other things; otherwise the value specifies a minimum distance.

```
rehearsal num dist 5     // at least 5 stepsizes away
reh bold "A1" dist 2 !   // exactly 2 stepsizes away
```

5.3.5 Setting the measure number

Mup keeps track of measure numbers automatically, but sometimes you may want to override this to set the current measure number to some specific value. This is done by adding mnum=*number* on a bar line, which will set the current measure number to the given *number*.

```
// Set the current measure number to 50
dblbar mnum=50
```

You can also tell Mup to adjust up or down from what it would otherwise count, by using a plus or minus sign instead of the equals sign, following by the number of measures to adjust by. So, for example, suppose you use invisbar in some way that causes Mup to not count something as a measure that you actually want to be counted. You could then use

```
invisbar mnum+1
```

5.3.6 Setting rehearsal letter or number

If you are using the "rehearsal let" or "rehearsal num" styles, you can set those to specific values too, similarly to how mnum can be set. This might be useful, for example, if you have a single input file that contains multiple songs or movements, and you want the rehearsal marks to start over at the beginning of each song or movement.

The rehearsal number can be set on any bar line using num=*N*, where *N* is a number, typically 1.

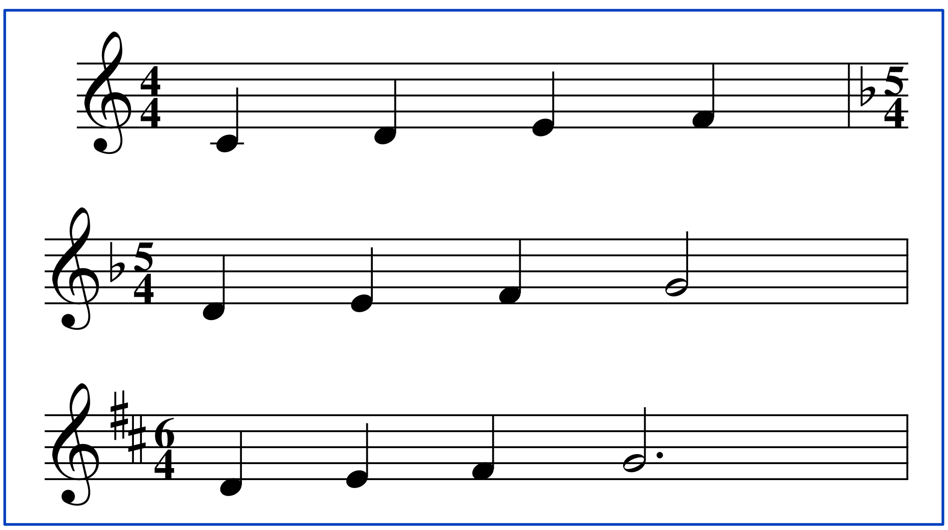
```
bar num=1
```

The rehearsal letter can be set in a similar way, using let="*X*" where *X* is either a single uppercase letter, A to Z, or two uppercase letters, AA to ZZ.

```
dblbar let="A" reh let
```

5.3.7 Hiding time/key signature and clef changes

Generally in printed music, when a time signature or key signature change occurs at the beginning of a score, these changes are also printed at the very end of the previous score, to make it clear there is a change coming up. Mup normally does this, but occasionally you may not want that behavior. Sometimes you may want to make a new score appear like the beginning of a new piece. If you wish to suppress the printing of time signature, key signature, and clef changes at the end of the preceding score, you can use the "hidechanges" keyword on the bar line at which the changes occur. Hidechanges cannot be used on a restart bar.

```
1: c;d;e;f;
bar
// change time/key with normal treatment

newscore
score time=5/4 ; key = 1&
music
1: d;e;f;2g;
// don't show changes at end of previous score
bar hidechanges

newscore

score time=6/4 ; key = 2#
music
1: d;e;f;2.g;
bar
```

5.4 Subbars

Sometimes you may want to mark subdivisions of measures. For example, you may want to indicate that something in 7/4 time is to be interpreted as 3+4 or 4+3. Although you could indicate that with the time signature, this can also be done using the subbarstyle parameter. The general syntax of this parameter is

subbarstyle = *linestyle bartype appearance ranges* **time** *counts*

The *linestyle* is optional, and can be dashed or dotted. The *bartype* is either bar or dblbar, and indicates whether to draw a single line or two lines.

The *appearance* is optional. When specified, it is in parentheses, describing where the subbar should end vertically. Top and bottom specifications are given, with the word "to" between them. Each specification consists of the word "top," "middle," or "bottom," indicating which line of the staff to draw relative to, optionally followed by a plus or minus sign and a number of stepsizes.

```
(top to bottom)
```

means draw from the top line of the top staff in each range to the bottom line of the bottom staff in the range. This is the default if no appearance is given.

```
(middle to middle)
```

would mean draw from the middle line of the top staff in each range to the middle line of the bottom staff in the range. As mentioned earlier, you can also specify adding or subtacting some number of stepsizes on each. So

```
(top-2 to bottom+2)
```

would draw from 2 stepsizes below the top line of the top staff to 2 stepsizes above the bottom line of the bottom staff.

The *range* is similar to ranges for the barstyle parameter, so things like 1-2 or 1-4,8-12 or all.

Often you may want the subbar marks to span just a part of each individual staff, so the "ranges" will be just individual staffs.

The "time" keyword is required and must be followed by at least one count at which to draw the subbar. The time can include a decimal part. Multiple times are separated by commas. Examples would be time 3 or time 2.5,4.75

Subbars are only drawn on staffs where there is actually a note or rest at that count.

Multiple styles can be given. For example:

```
subbarstyle=bar 1-4 time 2 dotted dblbar (top-1 to bottom+1) 5,6 time 4
```

Finally, you can use "between" as the appearance, to draw between staffs in the range.

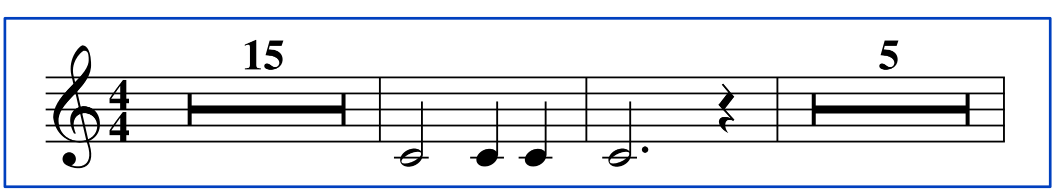
```
subbarstyle=dotted bar between all time 3
```

Once between is specified, it applies to all future styles in the parameter value, so if you want a mixture of between and not, specify all the not between first.

5.5 Multirest

Multiple measure rests can be indicated using the "multirest" statement. It is followed by the number of measures of rest. The number must be greater than 1, and no larger than 1000. There must not be any note or lyric information specified for a multirest measure. Examples:

```
multirest 15
bar
1: 2c;4;;
bar
1: 2.c;4r;
bar
multirest 5
bar
```



See also the -c command line option, and the restcombine parameter.

Normally, the number of measures of rest is printed, but it is possible to turn this off using the printmultnum parameter. An alternate notation can be specified using the restsymmult parameter.

5.6 Lyrics

Lyrics are specified in a somewhat similar manner to note groups. Lyrics for each staff are specified with the word "lyrics" optionally followed by a *place*, followed by the *staff number*, followed by a colon, then the timing and syllable information.

The *place*, if specified, can be "above" or "below," to indicate whether the lyrics are to be printed above or below the given staff. The default is below.

The *staff number* can be a single number or list of ranges of numbers. It can also be given as the keyword "all," which will place the lyrics above the top visible staff or below the bottom visible staff.

The place can also be specified as "between *M***&***N*" where *M* and *N* are numbers. This will cause the lyrics to printed approximately halfway between staff *M* and staff *N*. *N* must be exactly one greater than *M*.

```
lyrics 1:
lyrics above 2,3:
lyrics between 1&2, 3&4:
```

If you want the same lyrics above some staffs but below others, you can specify several places and list of staffs, separated by semicolons. For example:

```
lyrics above 1,3 ; below 2,4 ; between 5&6 :
```

After the colon comes a list of time values. Time values are separated by semicolons and are specified as they are for notes: "2" for half notes, "8." for dotted eighth, etc.

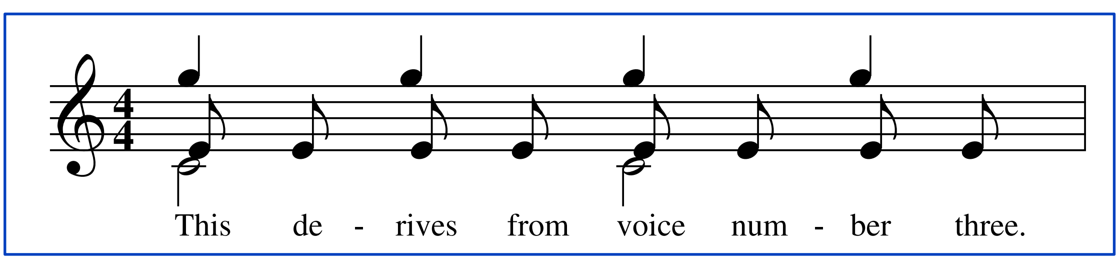
If the time values for lyrics are the same as the time values for the notes on the staff, the time values need not be specified. If you don't specify them, Mup will derive the appropriate values from the music input. If you list more than one staff, the time values will be derived from the first staff you list. Usually, the time values will be derived from voice 1, but if voice 1 music data doesn't exist or is invisible, or if the lyrics are explicitly specified as "below" and there is a visible voice 2, then voice 2 will be used. Voice 3, if any, is never used for automatically deriving time values for lyrics. Some examples:

```
1: c;d;e;f;
lyrics 1: "The sun will shine";
// The time values will be all quarter notes,
// as derived from the music on staff 1
bar

1: 8g;4.f;4g;a;
lyrics 1: "up-on the earth.";
// The derived time values will be 8;4.;4;;
bar

// In these examples, time values will be taken
// from staff 4, since it is the first one listed.
// In the last case, because "below" is specified explicitly,
// the times will be taken from voice 2
// if there is a visible voice 2; in all the other cases
// it will be taken from voice 1 if voice 1 music
// has been entered and is visible.
lyrics 4,7,9: "for-ev-er";
lyrics above 4-6: "nev-er-more";
lyrics between 4&5,6&7: "this is it";
lyrics below 4,2: "and so forth";
```

The line describing the music from which to derive the times must appear in the input prior to the lyrics line that is deriving the times. If chords in the music are tied or slurred together, Mup will treat those chords as being for a single syllable. However, there may be some circumstances in which the rules Mup uses to derive time values may not yield what you want, in which case you will have to explicitly specify the times for that measure.

Or if there is some other voice in the measure that has the times you want, you can tell Mup to use that one, by adding the keyword "using" and the staff and voice of that other voice. As usual, an omitted voice defaults to 1. This can even let you use voice 3 for time derivation, which would otherwise not be possible. Here is a simple example:

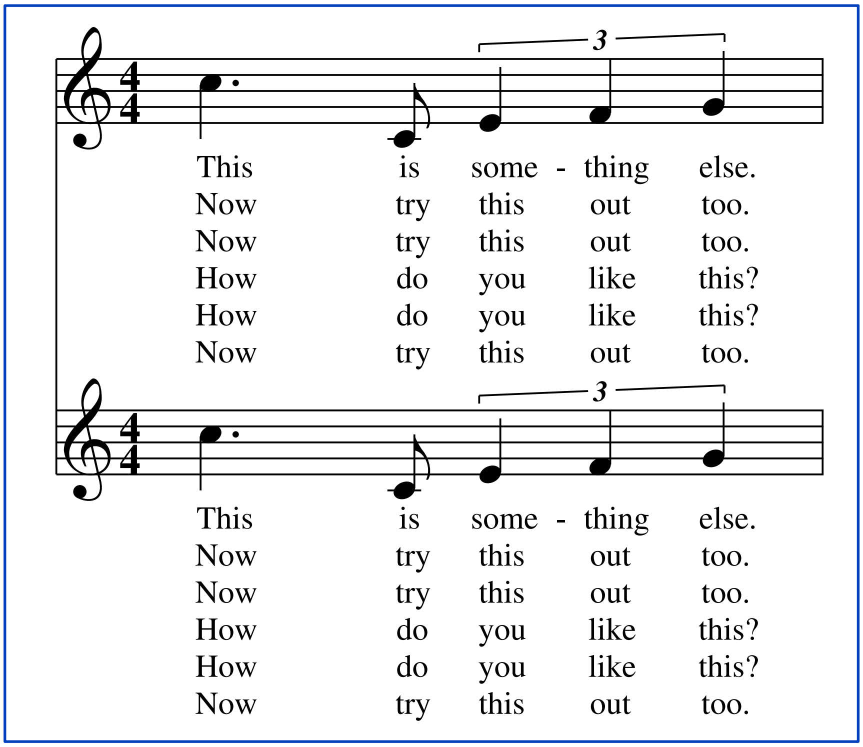
```
score
 vscheme=3f
music

1: g+;;;;
1 2: 2c;;
1 3: 8e;;;;;;;;
lyrics 1 using 1 3: "This de-rives from voice num-ber three.";
bar
```



Following the time values is a list of one or more verse numbers and lyric strings containing the words of the lyrics. The verse number(s) are given within square brackets. If no verse is specified on the first set of lyrics for a staff and place, verse 1 is assumed. On subsequent lyrics strings for that staff and place, the verse number is assumed to be one more than the previous verse number. Thus you only need to explicitly specify a verse number if you want to skip over a verse or supply them out of order.

Verse numbers need not be consecutive. The staff number and verse number can be given as lists or ranges. Another example:

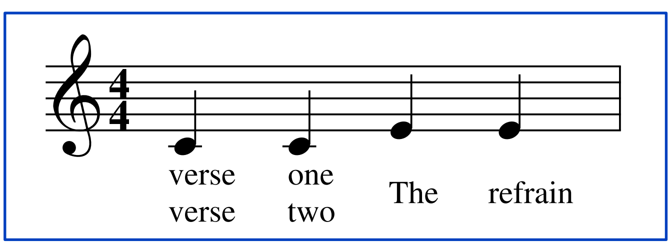
```
1-2: 4.c+;8c;{4e;f;g;}3;
lyrics below 1-2: 4.;8;{4;;;}3; \
        "This is some-thing else."; \
        [4,5] "How do you like this?"; \
        [2-3,6] "Now try this out too.";
bar
```



In this example, the lyrics will go below staffs 1 and 2. There are 5 syllables. The first is a dotted quarter in length, the second is an eighth, and the last three make up a quarter note triplet. The first string is for verse 1, since no verse number was specified. The second string will be used for verses 4 and 5, and the last string for verses 2, 3, and 6.

You can also specify a verse number of "c" which means the verse is to be centered vertically. This is useful if you have a refrain that is identical for several verses, and you don't want to have it printed multiple times.

```
1: c;;e;;
lyrics 1: ;;2s; [1] "verse one"; [2] "verse two";
lyrics 1: 2s;4;; [c] "The refrain";   // centered lyrics
bar
```

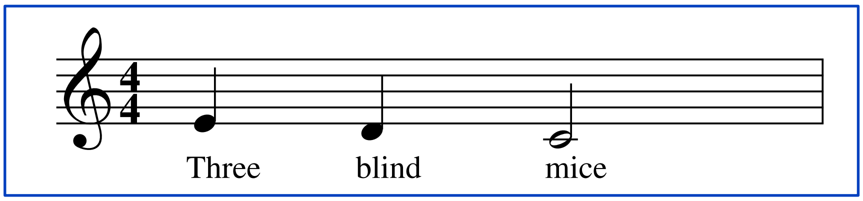


The lyrics string is followed by a semicolon. There must normally be one syllable in the lyrics string for each time value. (There is one exception, discussed later.) Syllables are separated in the lyrics string by white space, a dash, or an underscore.

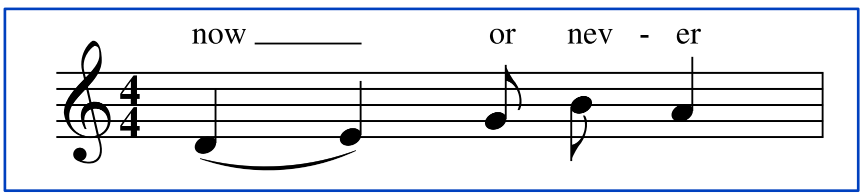
Sometimes a syllable is to be held out for several counts or over several notes. This can be indicated by dashes or underscores. If the syllable in the lyric string ends with a dash, on output the dash will be placed halfway between the given syllable and the next syllable. If the distance between the two syllables is long, several evenly spaced dashes will be printed. If a syllable in the lyric string ends with an underscore, an underscore line will be printed from the end of the current syllable to the edge of the last note associated with the syllable.

Here are some examples:

```
// Verse 1. The first two words have the
// duration of a quarter note each.
// The last word lasts a half note.
1: e;d;2c;
lyrics 1: 4;;2; [1] "Three blind mice";
bar
```

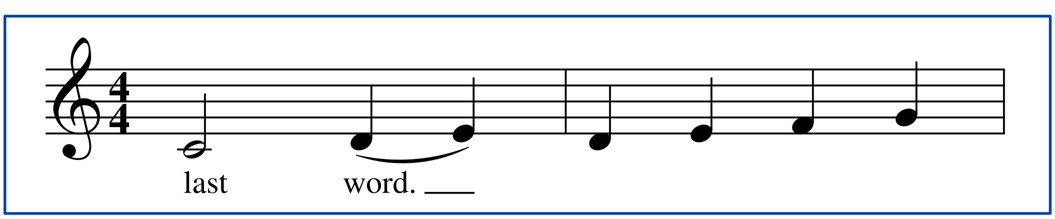



```
// examples of above lyrics with dash and underscore
1: 4d<>;e;8g;b;4a;
lyrics above 1: "now_ or nev-er";
bar
```



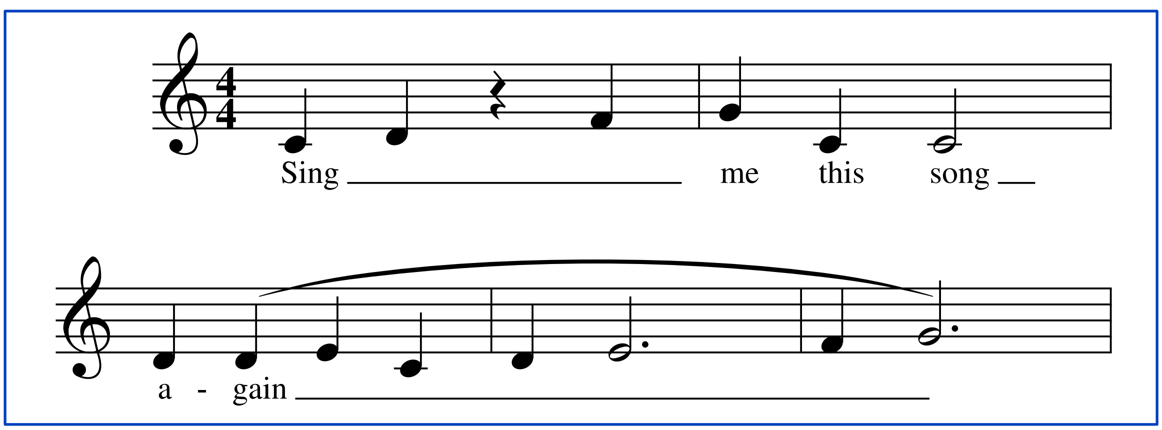
Mup does its best to figure out where to end dashes or underscores. However, if there isn't a following syllable after a dash or underscore, Mup would extend the dash or underscore to the end of the piece, which may not be what you want. There are a few other cases where Mup may be unable to properly deduce where you had intended an underscore to end. You can manually halt the dash or underscore by adding in an "empty syllable," consisting of "<>". Normally, the angle brackets are used inside lyrics to enclose special non-lyrics items, as will be described a bit later. However, if they are used by themselves with nothing between them, they essentially mean a syllable with no text.

```
1: 2c;4d<>;e;
lyrics 1: "last word._";
bar

1: 4d;e;f;g;
// add empty syllable to end the underscore
lyrics 1: 1; "<>";
bar
```



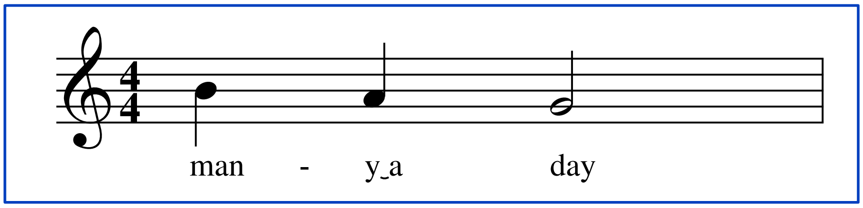
However, if necessary, it is also possible to tell Mup exactly where you want dashes or underscores to end. To do that, immediately after the underscore or dash character, specify {*M*m+*N*}, where *M* is the number of measures (i.e., the number of bar lines to continue to) and *N* is the number of beats into the final measure where it is to end. If either is zero, it can be omitted. The beats can include a decimal fraction part. Examples:

```
1: c;d;r;f;
// Force underscore to go past the rest to very end of measure.
// This example only specifies measures, not beats.
lyrics 1: 1; "Sing_{1m}";
bar

1: g;c;2c;
// Force to end just beyond count 4 in current measure.
// This example only specifies beats, not measures.
lyrics 1: "me this song_{4.1}";
bar

newscore

1: d; ph above;e;c;
// Continue across two bar lines and two more beats.
// This example specifies both measures and beats.
lyrics 1: 4;2.; "a-gain_{2m+2}";
bar

1: d;2.e;
bar

1: f;2.g eph;
bar
```



Occasionally, a single chord is used for more than one syllable. If the syllables are within the same word, it is sufficient to omit the dashes between syllables, so that Mup will treat them as a single syllable. However, if the syllables are in separate words, a "~" can be used in place of a space between the syllables. On output, Mup will replace the "~" with a space and a small, curved line below the space, indicating that the syllables on either side are to be joined. For example:

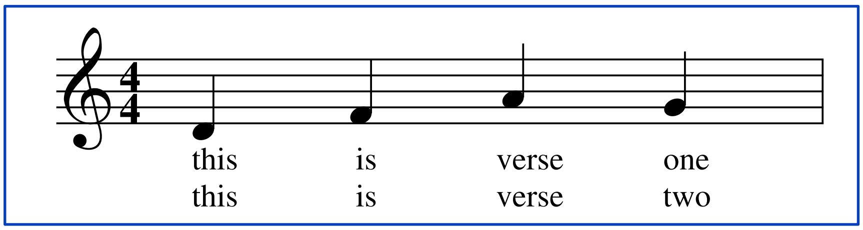
```
1: b;a;2g;
lyrics 1: ;;2; "man-y~a day";
bar
```



If you don't want the curved line, you can use the special character name "\(space)" instead of a literal space. That special space character is printed like a normal space on output, but is not considered a space for the purposes of determining syllable boundaries. Alternately, a technique using angle brackets, described later, can be used.

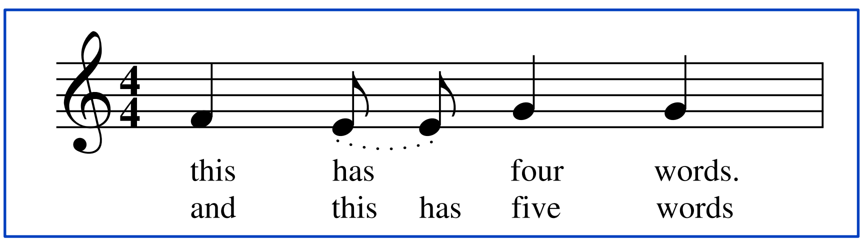
If several verses use the same time values, you can specify them all on one input line. For example:

```
1: d;f;a;g;
lyrics 1: [1] "this is verse one"; [2] "this is verse two";
bar
```



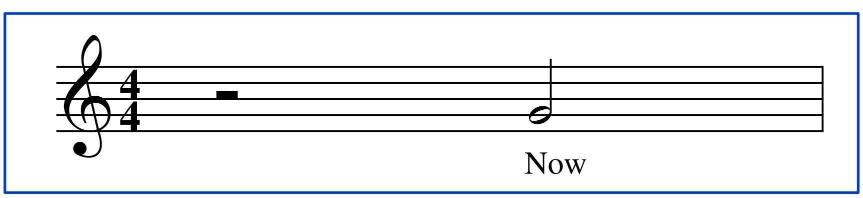
If verses have different time values, use separate lines of input:

```
1: f;8e dotted tie;;4g;;
lyrics 1: [1] "this has four words.";
lyrics 1: 4;8;;4;; [2] "and this has five words";
bar
```



Occasionally, lyrics may occur during only part of a measure. This case can be handled by specifying "space" by using an "s" after time values that have no lyric syllable associated with them. For example:

```
1: 2r;g;
lyrics 1: 2s;; [1] "Now";
bar
```



In this example, the first half note of the measure is a space, so there will be no lyric there. The second half note of the measure will have the word "Now" as its lyric. Note that the "s" does not work quite the same way with lyrics as it does with notes. With notes, "2s;;" would mean two half note spaces, because the space would be used as default for the following chord where no notes were specified. With lyrics, "2s;;" means a half note space, followed by a half note lyric; the space is not carried forward as a default.

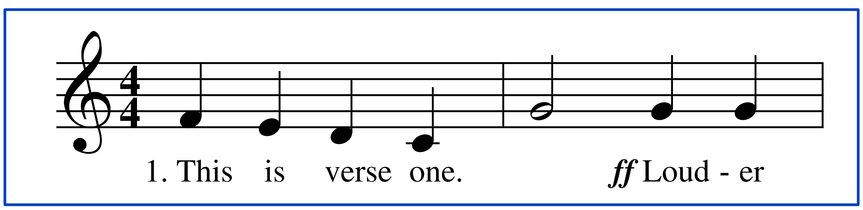
If you don't specify any time values, relying on Mup to derive the time values from the corresponding music time values, any rests and spaces in the music will be translated to lyric spaces. If the first note entered in a chord is tied to or slurred to the following chord, that following chord will also be translated to a lyric space, since you most likely want a single syllable to span both chords in that case.

In addition, if you set the extendlyrics parameter, Mup will deduce where to add in underscores, based on the locations of ties and slurs.

Occasionally, you may want to print something within a lyric string which isn't really a lyric syllable. The most common example of this would be that you may want to print verse numbers. Other possibilities may include associating a dynamic mark (e.g., "mf") with a particular verse, or marking a section for a subset of the singers (e.g., "Men:" or "Solo:"). These extra things are specified within angle brackets. They can be placed immediately before or after any syllable. Mup will not consider them when lining up the syllable with notes. Normally, it will assume there is enough space to print them rather than reserving space for them. If you want Mup to ensure there is enough space to print them, put a "^" immediately after the "<".

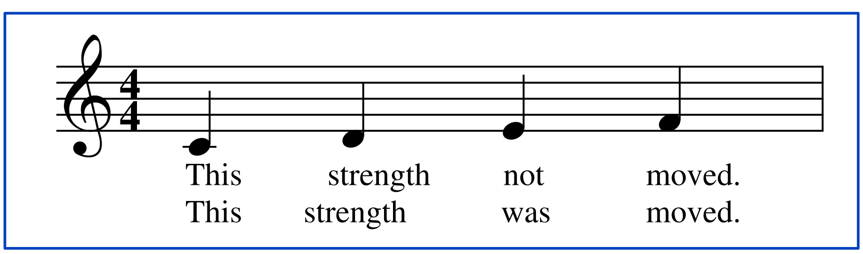
Some examples:

```
1: f;e;d;c;
lyrics 1: "<1. >This is verse one.";
bar
1: 2g;4;;
lyrics 1: 2s;4;; "<^\f(TX)ff\f(PV) >Loud-er";
bar
```



The < > construct can also be used to fine-tune the placement of syllables. The placement of syllables is governed in general by the lyricsalign parameter. Sometimes, however, you may want to adjust the placement of specific syllables. Suppose you have a long syllable, like "strength" and would like to shift it leftward somewhat. Entering it as "<str>ength" would cause Mup to move the syllable farther to the left than it normally would.

```
1: c;d;e;f;
lyrics 1: "This strength not moved."; \
        "This <str>ength was moved.";
bar
```



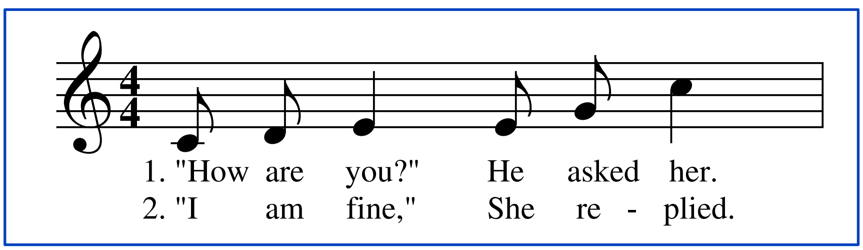
Angle brackets may also be useful for entering chant, where many words are to be associated with a single note. For example:

```
lyrics 1: "All<^ these words will be treated like one syllable.>";
```

The font and size to use for lyrics is initially determined from the "lyricsfontfamily," "lyricsfont" and "lyricssize" parameters for the appropriate staff. If "all" is used for the staff number, the score parameters are used. In the case of "between," the parameters for the staff above are used for determining the font and size. The usual \f and \s forms can be used to change these values for a given verse. Lyrics fontfamily, font and size values are maintained separately for each combination of staff number, verse number, and place, and are carried forward from one measure to the next. Thus, for example, if you want one verse to be printed in roman and another in italics (perhaps one is in English and the other in another language), you need only put a "\f(TI)" at the beginning of the syllable string for the first measure of the second verse, and all subsequent syllables for that verse will be in italics. Setting the "lyricsfontfamily," "lyricsfont" or "lyricssize" parameters will reset the values for all verses of the staff (if set in staff context), or the entire score (if set in score context).

Another way to align a syllable in a particular way is to precede the syllable with a |. If the | is preceded by a number (optionally signed), the left edge of the syllable will be placed that many points from the horizontal "middle" of the chord. If there isn't a number before it, the value is obtained from the sylposition parameter. Negative values are to the left, and will usually be what you want. This alignment method is particularly useful for lining up multiple verses, either to get verse numbers to line up or to align words at the beginning of a poetic line. The | goes after anything in angle brackets.

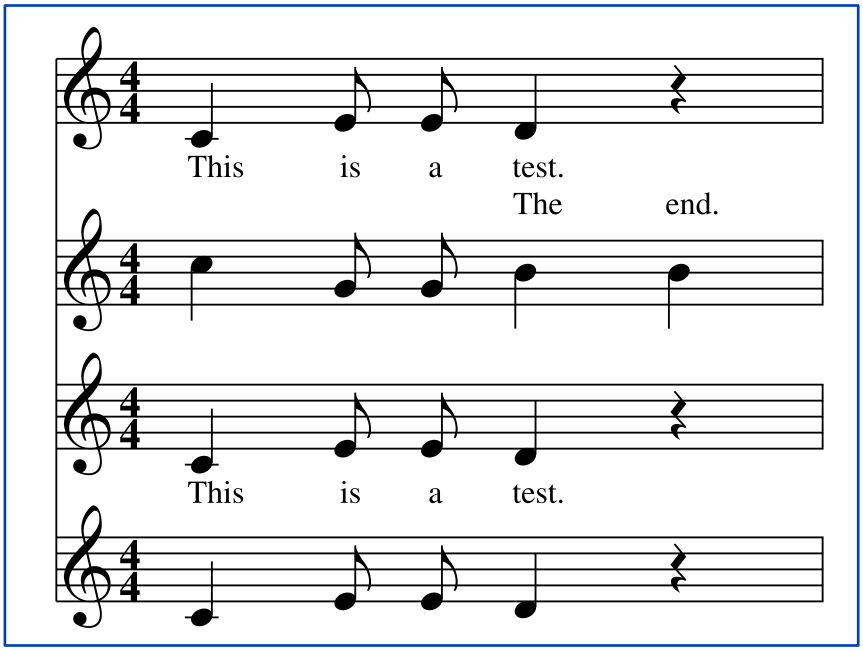
```
1: 8c;d;4e;8e;g;4c+;
lyrics 1: "<1. >|\"How are you?\" -7|He asked her."; \
          "<2. >|\"I am fine,\" -7|She re-plied.";
bar
```



Location tags can be set on lyrics syllables by adding \=(*tagname*) to the syllable. As with all tags, the *tagname* must be either a single lowercase letter, or an underscore followed by any number of uppercase or lowercase letters, digits, or underscores. The area associated with the tag will include the syllable itself and anything inside <ˆ >, but excluding things inside < >. While Mup will recognize a tag like this almost anywhere within a syllable (except inside < > or before any initial < >), it is recommended that it be placed at the end of the syllable, before the dash or underscore, if any, because that's less confusing to read than putting it in the middle of a syllable. An example showing common uses of lyric tags in given in the Hints section.

A few more examples of lyrics:

```
1,3-4: c;8e;;4d;r;
2: c+;8g;;4b;;
lyrics between 1&2,3&4: 4;8;;4;s; "This is a test.";
lyrics above 2: 2s;4;; "The end.";
bar
```


This is a test.
The end.
This is a test.

6. TABLATURE NOTATION

Mup can print tablature staffs. Tablature notation is commonly used for guitar and several other instruments. Mup supports tablature for 1 to 9 strings.

Mup will automatically create a normal 5-line staff above each tablature staff, giving the equivalant music on that staff. This staff will be referred to as the "tabnote staff." Normally, you will input music just for the tablature staff, and the tabnote staff will be derived from that. However, for any given measure, you can explicitly input music for a tabnote staff, which will turn off the automatic translation for that measure. If you only want either the tab or tabnote staff to be printed, you can mark the other one as invisible (see the "visible" parameter). You should specify the appropriate clef and key signature for the tabnote staff, so that Mup can make the best derivation.

6.1 Specifying a tablature staff

To set up a tablature staff, you use the stafflines parameter. To get a standard 6-line guitar tablature staff, you can just say

```
score
     staffs=2
staff 2
     stafflines = tab
```

There are 2 staffs, because there is both the tablature staff and the automatically generated tabnote staff. The tablature staff is always immediately below the corresponding tabnote staff. Setting the stafflines parameter to "tab" marks staff 2 as a tablature staff.

If you want tablature for an instrument other than a 6-string guitar with standard tuning, you specify the pitches of the strings from top to bottom within parentheses after the "tab." For example:

```
stafflines = tab (e a d& g)
```

would define some instrument that had 4 strings, with the string on the top line of tablature staff being an e string, the next a, the next d flat, and the bottom g. As shown in the example, pitches can include a # or & if necessary. Strings are assumed to be in octave 4 unless otherwise marked. You can specify a different octave by specifying an octave number after the string's pitch (using either an absolute octave number or pluses or minuses). If the instrument has more than one string with the same pitch (even if they are in different octaves), they must be distinguished by adding one or more ' marks after the pitch. The tablature definition for standard guitar is

```
stafflines = tab (e5 b g d a3 e'3)
```

This specifies that the top string on the staff is e in octave 5. The next three strings are in the default octave of 4, and the last two strings are in octave 3. Since there are two different strings with pitch letter of e, the lower e is marked as e'. Note that the octaves given are how they should be printed on the tabnote staff. A standard guitar actually sounds an octave lower than written. If you just use "stafflines = tab" without specifying any strings, Mup not only assumes the standard guitar layout, and prints in the appropriate octave, but it also automatically transposes the MIDI output to the actual pitches an octave lower. If you specify strings explicitly, you will have to specify the octaves and any desired MIDI transposition values yourself. Stated another way,

```
score
     staffs = 2
staff 2
     stafflines = tab
```

is equivalent to

```
score
     staffs=2
staff 2
     stafflines = tab (e5 b g d a3 e'3)
staff 1
     ifdef MIDI transpose = down perfect 8 endif
```

Note from this last example that MIDI is taken from the tabnote staff, not the tablature staff, so MIDI directives should be placed with the tabnote staff.

Mup should be able to handle almost any instrument with up to 9 strings. Several strings can have the same pitch; you just need to distinguish them with ' marks. The strings must be listed in whatever order the strings are to appear on the tablature staff. As another example,

```
stafflines = tab (g3 d3 a2 e2)
```

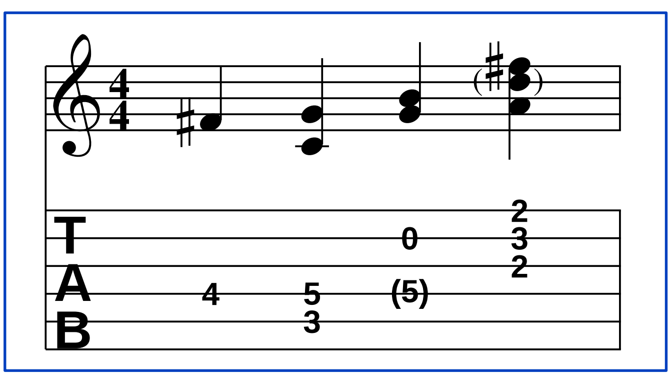
would define a standard bass guitar. Note that a bass guitar also sounds an octave lower than written, so it should be transposed for MIDI purposes.

The list of strings, if any, can optionally be followed by y or n. Using y will cause the word TAB to be printed vertically at the beginning of every score. Using n will cause that to never to be printed. Not specifying either will cause it to be printed only on the very first staff, which is the most common way of printing tablature.

6.2 Music input on tablature staffs

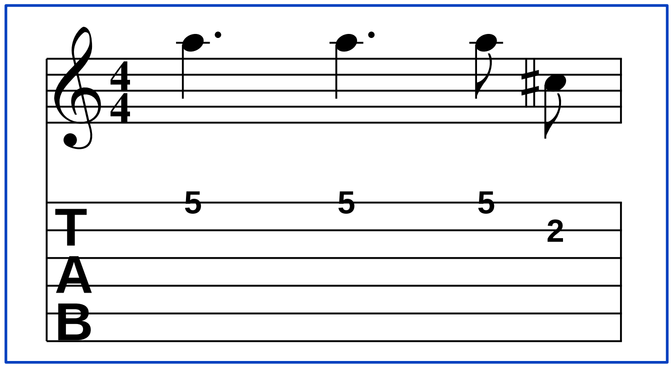
Input of tablature is similar to that on regular staffs. Each note in a chord is specified by a string and a fret. So fret 3 on the g string is designated by g3, or fret 0 on the e' string is designated by e'0. If your open string pitch includes an accidental, that would be included as well, so if you had an instrument with an f# string and wanted to play the 6th fret on that string, it would be f#6. Fret numbers can range from 0 to 99. If you want fret numbers to be placed inside parentheses, put parentheses around them in the input. You can also put both the string and fret within a set of parentheses, in which case, the note generated on the tabnote staff will have parentheses around it. Here are some examples of chords:

```
2: d4; d5a3; b0 d(5); e2(b3)g2;
bar
```



Time values are specified just like on non-tablature staffs: a number optionally followed by one or more dots (like 4 or 8..), or a list of such numbers added together (like 2+8), preceding the chord pitches. Tablature staffs follow the same rules as non-tablature staffs for using time values from the previous chord if no time value is specified on the current chord. Similarly, if no notes are specified for a chord, the same notes used in the previous chord are reused. As an example:

```
2: 4.e5; ;8;b2;
bar
```



Since nothing is specified for the second chord, both time value (dotted quarter) and note (e string fret 5) are copied from the first chord. The third chord in the measure has only a time value (an eighth note) so the note (string/fret) is repeated. The final chord has only a note, so the previous time value is used.

Rests and spaces can be specified with r and s, just like on non-tablature staffs. They result in nothing being printed on the tablature staff, and rest or space being printed on the tabnote staff.

6.3 Chord attributes

Any chord on a tablature staff can be preceded by a list of chord attributes in brackets, just like with non-tablature staffs. If you want a chord to be printed with X (for a muffled string) rather than a fret number, use [xnote]. In this case, the fret number will be used only to determine where to put the X on the tabnote staff, and what note to use for MIDI. Using [diam] will have no effect on the tablature staff, but will cause diamond notes to be used on the tabnote staff. Using [grace] or [cue] will cause the fret numbers to printed in a smaller size, and the corresponding notes on the tabnote staff to be printed in a smaller size. Using [slash *N*] will cause *N* slashes to be printed below the chord on the tablature staff, as well as slashes on the corresponding chord on the tabnote staff. You can use the other chord attributes (with, pad, and =tag) just like on a non-tablature staff.

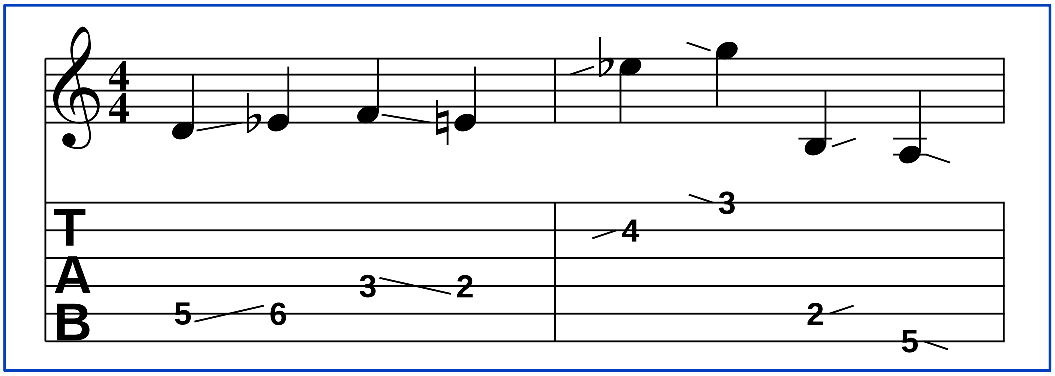
6.4 Ties

From an input point of view, ties work much the same on tablature staffs as on other staffs. On output, tie marks on tablature staffs are normally not printed, and the frets are not printed on the tied-to chord. If a tie goes across to a new score, the frets are printed, but in parentheses, to indicate the chord is really just the continuation of a tie.

6.5 Slides

In Mup input, slides are shown with <>, rather like slurs on non-tablature staffs. There is a special variation used for slides that come from nowhere in particular or go to nowhere in particular. These are shown with <\n>, </n>, <n\>, or <n/>. A slide is indicated on output on tablature staffs as a slanted line between 2 fret numbers on a string. On tabnote staffs, they as drawn as a slanted line between 2 notes. See the examples below:

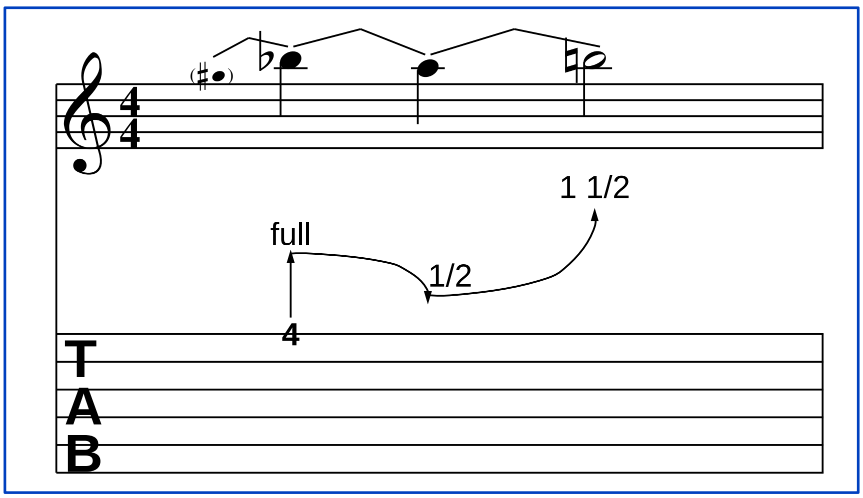
```
2: a5<>; a6; d3<>; d2;
bar

2: b4</n>; e3<\n>; a2<n/>; e'5<n\>;
bar
```



6.6 Bends

A bend (stretching a string to make it sound higher than normal) is specified on tablature staffs by putting a text string after the string. The text string can be the word "full" or a number and/or fraction. Some examples:

```
2: e4 "full"; e "1/2"; 2e "1 1/2";
bar
```

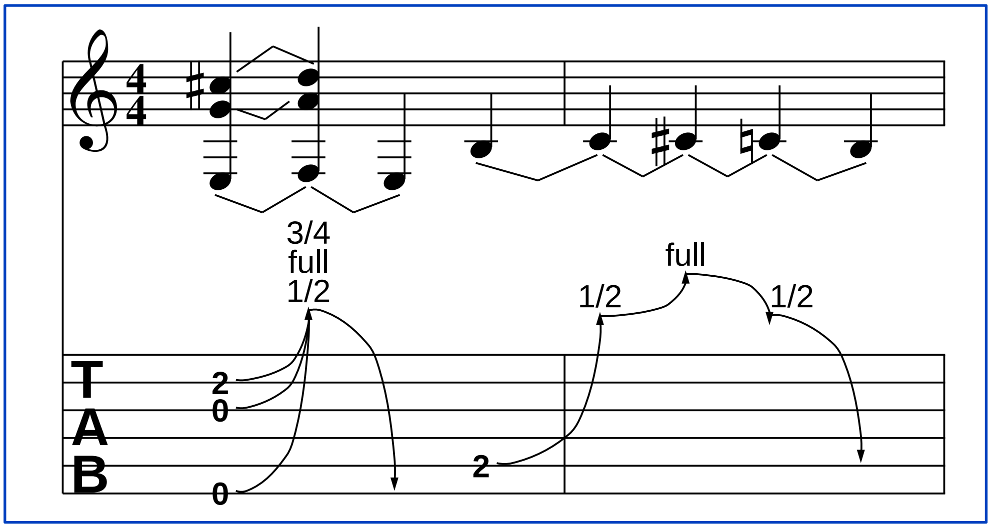


If you want to bend back down to having no bend (a release), the text string specifying the bend is just an empty string of "". Note that if both a fret and bend other than "" are specified, this indicates a prebend, whereas if only a bend is specified, this indicates a normal bend.

Bends need not be a whole number of half steps, but if you specify a bend that falls between half steps, when it is used on the tabnote staff, the bend is rounded to the nearest half step, or rounded downward if it is exactly in the middle of a half step. When possible, bends of 1/4 step or less are shown on the tabnote staff by a small curve rather than by a separate note.

You can specify bends on more than one string at once, but a continuation bend (bending to a new distance without replucking the string) is only allowed on a single string at a time. Here are some examples:

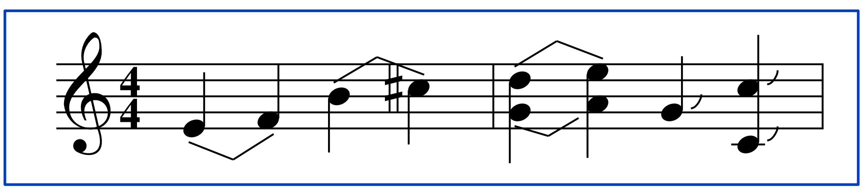
```
2: g0b2e'0; e' "1/2" g "full" b "3/4"; e' ""; a2;
bar

2: a "1/2"; a "full"; a "1/2"; a "";
bar
```



If you have more than one voice on a tablature staff, bends on other than the top voice may collide with other voices in some cases, and if there are bends from more than one voice simultaneously, they may collide.

On non-tablature staffs, bends are specified with a ^ symbol followed by the note to bend to. The note to bend to is specified by letter and optional octave (accidentals are not specified). If you want just a small bend (1/4 step), this is specified by ^/. For example:

```
1: e ^f; f; b ^c5; c#5;
bar

1: d+^e+ g^a; e+a; g ^/; c ^/ c+^/;
bar
```



6.7 Miscellaneous

Note attributes of ~ for tie, ? for a small note, and = for a tag work the same on tablature staffs as on non-tablature staffs.

If bm, ebm, esbm, abm, or eabm are given on tablature staffs they are transferred to the tabnote staff and used for custom beaming or automatic beaming there, as appropriate. Items in "with" lists inside [ ] are also copied to the tabnote staff. In general, unless otherwise specified here, if you want items such as tempo and dynamic marks to appear on both tablature and tabnote staff, you have to specify them for both staffs.

The "alt" inter-chord attribute is not allowed on tablature staffs.

When there is a tablature staff, the MIDI is actually generated from the tabnote staff. There are some limitations. Currently, no MIDI pitch bends are generated; slides and bends are instantaneous rather than gradual.

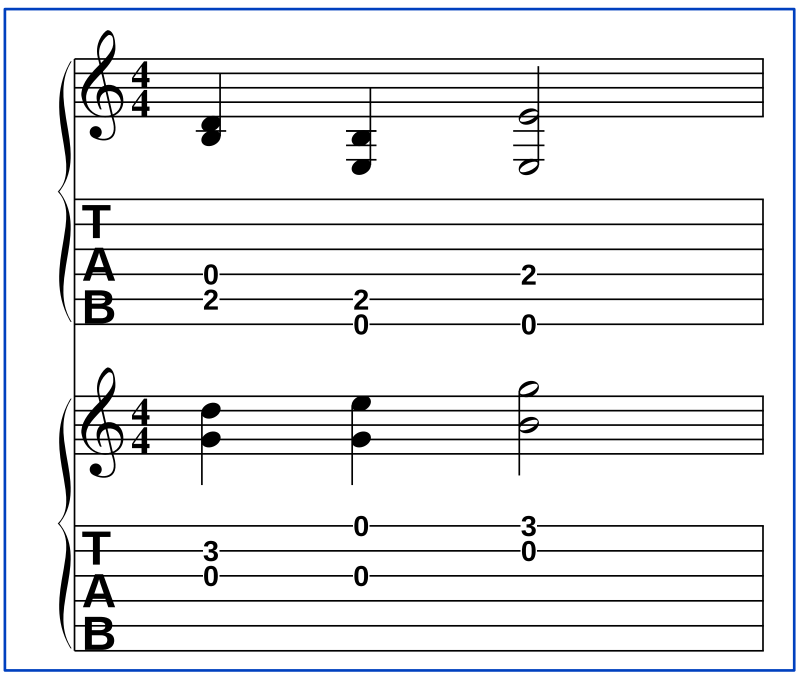
If a capo is being used, it probably better to declare a transposition of the tabnote staff rather than declaring the strings to be different notes. This is because most people would rather think of the original string letters when entering the tab staff information; and if you decide to put the capo on a different fret later, you only have to change the "transpose" line. The automatic MIDI octave transposition for standard guitar will also then still apply, on top of your transposition.

If you specify an octave mark (discussed in a later section) on a tabnote staff, the derived music will be printed with the notes raised or lowered the appropriate number of octaves to correspond to the given octave mark interval.

Since ties are not printed on tablature staff, and input notation for tablature slides is like what is used for slurs on ordinary staffs, if you want a curved line like a tie or slur on a tablature or tabnote staff, you have to use a phrase mark. See the section on phrase marks for more details.

The chord-at-a-time input style can also be used for tablature. The staffs being mapped to from a single input line must either be all tablature or all non-tablature, however.

```
score
    staffs=4
    brace=1-2, 3-4
    tabwhitebox=y
staff 2
    stafflines=tab
staff 4
    stafflines=tab
music

// The first two notes in each chord go to staff 2,
// and the last two go to staff 4.
[ 2; 2; 4; 4] : a2d0 g0b3; e'0a2 g0e0; 2 e'0d2 b0e3;
bar
```



7. SHAPED NOTES

If you intend to only use the usual system of noteheads, you can skip over this section on shaped notes. If you wish to use less common notehead shapes, like X-shaped notes, rather than normal noteheads, or want "shaped note" music that is often used for "Sacred Harp" style music, sometimes also called "fasola notation," then this section will explain how you can do that.

7.1 Headshapes context

The headshapes context is used in conjuction with the noteheads parameter to determine what shapes will be used when printing noteheads. It is rarely necessary to include a headshapes context in your music, since Mup already has the most common values built in, so generally you just need to use the noteheads parameter to access them. However, it is still important to understand what the headshapes context can contain, to understand how the built-in values work.

The headshapes context defines, for a given head shape name, what specific notehead characters to use for the notes of various durations. It contains one or more pairs of strings. The first string in the pair gives a name for a set of notehead shapes. The second string contains a space-separated list of the names of 4, 5, or 6 notehead characters to use for that head shape name. The first shape in the list is used for quarter notes and shorter, the second for half notes, the third for whole notes, and the fourth for double whole notes. If there is a fifth, that is used for quadruple whole notes. If there is a sixth, that is used for octuple whole notes.

If an upside-down version of the character is to be used for stem-down notes, the name is prefixed by "u?" (The "u" stands for "upside-down" and the question mark is intended to be mnemonic for the fact that the upside-down version will only be used part of the time, namely for stem-down notes.)

As was mentioned above, the most common mappings are already built into Mup. Here is the list of the pre-defined values. Pay particular attention to the first string on each line, since those are the names you will use in the noteheads parameter and for overriding chord and note shapes.

"norm"	"4n 2n 1n dblwhole quadwhole octwhole"
"x"	"xnote diamond diamond dwhdiamond quadwhole octwhole"
"allx"	"xnote xnote xnote xnote xnote xnote"
"diam"	"filldiamond diamond diamond dwhdiamond quadwhole octwhole"
"blank"	"blankhead blankhead blankhead blankhead"
"righttri"	"u?fillrighttriangle u?righttriangle u?righttriangle u?dwhrighttriangle quadwhole octwhole"
"isostri"	"fillisostriangle isostriangle isostriangle dwhisostriangle quadwhole octwhole"
"rect"	"fillrectangle rectangle rectangle dwhrectangle quadwhole octwhole"
"pie"	"fillpiewedge piewedge piewedge dwhpiewedge quadwhole octwhole"
"semicirc"	"fillsemicircle semicircle semicircle dwhsemicircle quadwhole octwhole"
"slash"	"fillslashhead slashhead slashhead dwhslashhead quadwhole octwhole"
"allslash"	"fillslashhead fillslashhead fillslashhead fillslashhead fillslashhead fillshlashhead"
"mensural"	"mensurfilldiamond mensurdiamond mensurdiamond mensurdblwhole"

You can redefine these or define new ones if you wish. The name (the first of the two strings in the pair) can be almost anything you want. The four to six names in the second string must be taken from the list of valid notehead characters given below, or be characters you have defined yourself and for which you have provided a ystemoffset value (described in the section on user-defined symbols).

4n	2n	1n
dblwhole	quadwhole	octwhole
filldiamond	diamond	dwhdiamond
fillisostriangle	isostriangle	dwhisostriangle
fillpiewedge	piewedge	dwhpiewedge
fillrectangle	rectangle	dwhrectangle
fillrighttriangle	righttriangle	dwhrighttriangle
ufillrighttriangle	urighttriangle	udwhrighttriangle
fillsemicircle	semicircle	dwhsemicircle
fillslashhead	slashhead	dwhslashhead
xnote	altdblwhole	blankhead
mensurfilldiamond	mensurdiamond	mensurdblwhole

The righttriangle shape names can be prefixed by u? when used in the headshapes context, to indicate the upside-down version of them should be used when the stem is down. Note that u? cannot be used on any other notehead characters, since none of the others have a corresponding upside-down version.

As an example of how you could use the headshapes context, suppose you wished to use xnote for half notes and shorter, and diamond for longer notes. That is different than either of the built-in values "x" or "allx," but you could override one of them:

```
headshapes
     "x" "xnote xnote diamond diamond"
```

Another common case is if you want to use the alternate double whole note symbol. There are two very commonly used forms for double whole. One has a single vertical line on either side of an ellipse, the other has two vertical lines. Mup uses the one-line form by default, but if you prefer the two-line form, you can get that via:

```
headshapes
     "norm" "4n 2n 1n altdblwhole"
```

The blankhead does not print any head at all, it just leaves space as if there were a notehead. It might be used if for some reason you just want stems.

The use of "mensural" will also cause stems to be centered, as is appropriate for those noteheads.

7.2 Noteheads parameter

The noteheads parameter describes which notehead shape to use for each pitch in the scale. It can be specified in score, staff, or voice contexts. If you want to use the same shape for all pitches, (as is the case with standard notation), only one shape name is specified. Thus the default value for this parameter is

```
noteheads = "norm"
```

If you want to use different shapes for different pitches, the noteheads value needs to be a string containing a list of 7 shape names. They are listed from the "tonic" of the major key up the scale, with the key based on the number of sharps or flats in the key signature. The shape names must be from the list of pre-defined head shapes (norm, xnote, diam, blank, righttri, isostri, rect, pie, semicirc, slash), or names that you have given as the first in a pair of strings in the headshapes context.

You can get the most common shaped note system using 4 shapes using:

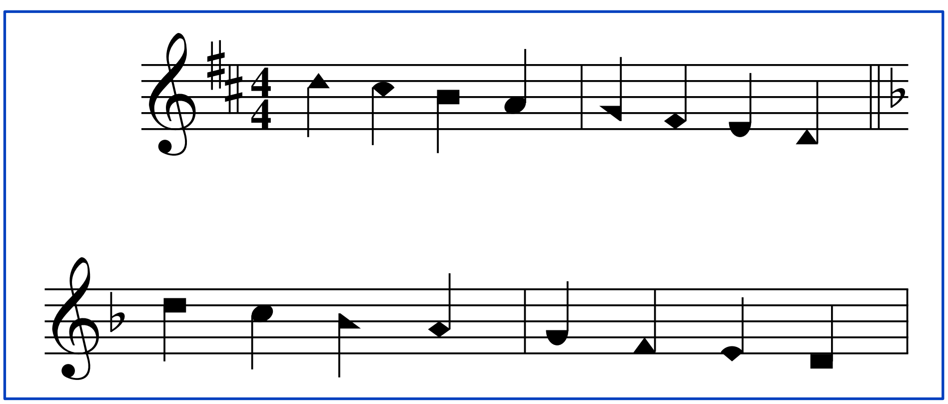
```
noteheads = "righttri norm rect righttri norm rect diam"
```

There are several shaped notes systems using 7 different shapes. One such system is specified by:

```
noteheads = "isostri semicirc diam righttri norm rect pie"
```

Once the noteheads parameter is set, you specify your music just like you would for standard notation, but Mup will use the appropriate noteheads based on your specifications.

```
score
   // Use a 7-shaped system.
   noteheads = "isostri semicirc diam righttri norm rect pie"

   // Noteheads are given in order for the major key,
   // so we'll start out with an example in major.
   key = d major

music

// Do a descending scale in D major
1: d+;c+;b;a;
bar

1: g;f;e;d;
dblbar

score
   // Now we switch to D minor, where the "tonic"
   // will start at the rect and then wrap around.
   key = d minor

music

// Do a descending scale in D minor
1: d+;c+;b;a;
bar

1: g;f;e;d;
bar
```



7.3 Overriding chord noteheads

It is possible to override what note shape to use for a chord, by giving "hs" followed by a head shape name from the headshapes context, inside square brackets. Thus

```
[hs "righttri"]
```

would use the "righttri" headshape. The specific character to use would be based on the group's duration. In other words, if the chord was a quarter note or shorter, the "fillrighttriangle" notehead would be used, but if the chord was a half or whole note, a "righttriangle" notehead would be used, and a "dwhrighttriangle" would be used for a double whole note.

The hs specification can be used along with other things that can go in the square brackets. For example,

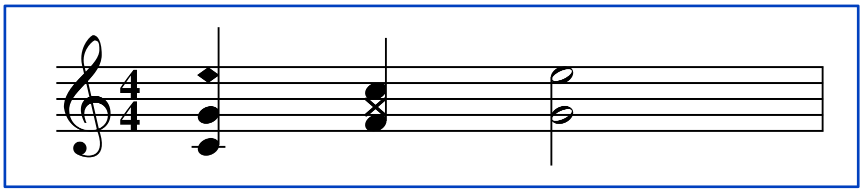
```
[hs "blank"; len 0]
```

would use blankheads and no stem, resulting in no chord being printed at all!

7.4 Overriding individual noteheads

If you want to override the notehead shape to be used for one specific note in a chord, you use hs followed by the head shape name as a string after the note.

```
1: cg e+ hs "diam"; f a hs "x" c+; 2ge+;
bar
```

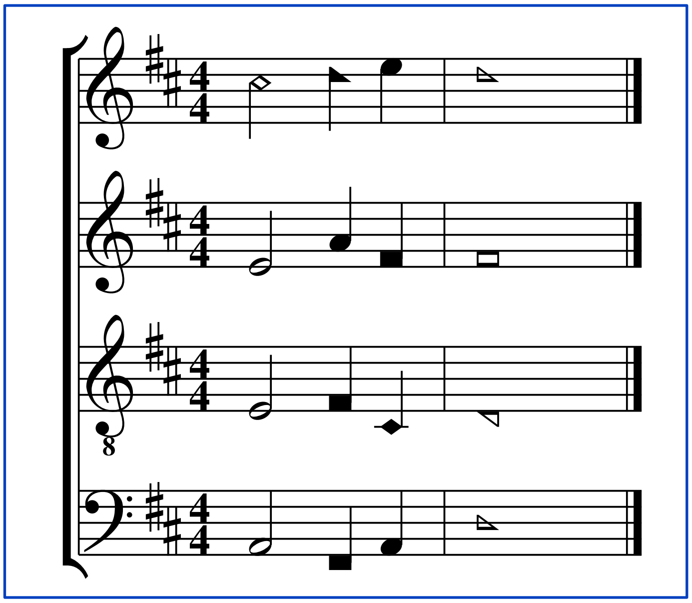


7.5 Putting it all together to use shaped notes

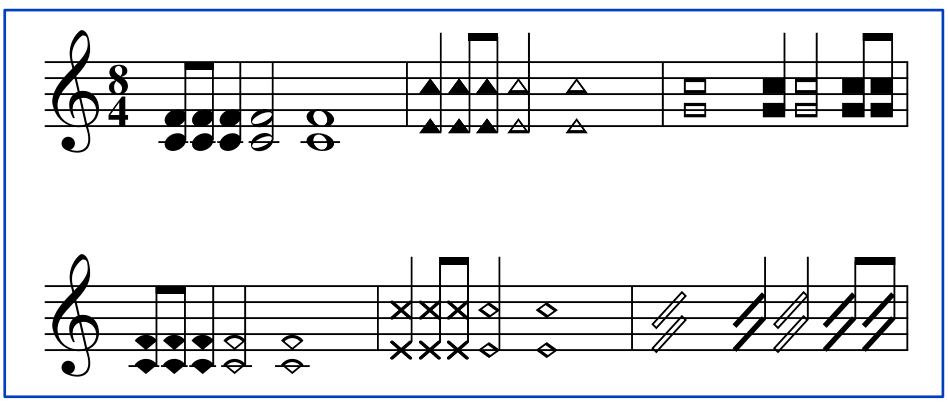
In summary, notehead shapes can be specified in five different places: per note, per chord, in voice context, in staff context, and in score context. When deciding what notehead shape to use, Mup checks for specifications in that order, using the first it finds.

Since fret numbers are used rather than noteheads on tablature staffs, the only head shape name that is allowed is "allx." which is used for "muffled" notes.

Here is a simple example of shaped notes, using the common 4-shape system.

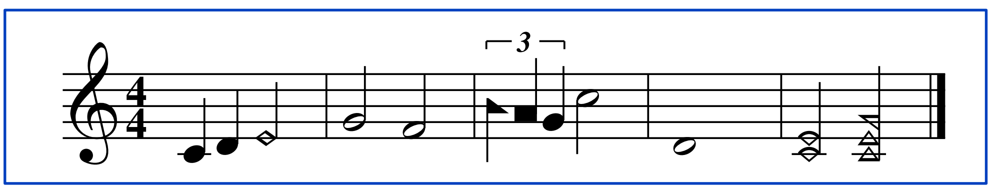
```
score
    noteheads = "righttri norm rect righttri norm rect diam"
    staffs=4
    key=2#
    bracket=1-4

staff 3
    clef=treble8
staff 4
    clef=bass
music

1: 2c+;4d+;e+;
2: 2e;4a;f;
3: 2e;4f;c;
4: 2a-;4f-;a-;
bar

1: 1d+;
2: 1f;
3: 1d;
4: 1d;
endbar
```

Here is an example of using the same notehead shape for all pitches, illustrating how the proper version—filled or open—of the notehead is used, based on the note's duration.

```
score
    time=8/4
    beamstyle=2,2,2,2

music
1: 8cf;;4;2;1;
bar

score noteheads="isostri"
music
1: 4ec+;8;;2;1;
bar

score noteheads="rect"
music
1: 1gc+;4;2;8;;
bar

score noteheads="pie"
music
1: 8cf;;4;2;1;
bar

score noteheads="x"
music
1: 4ec+;8;;2;1;
bar

score noteheads="slash"
music
1: 1gc+;4;2;8;;
bar
```



And finally, here is a somewhat silly example that demonstrates how you can use the various shaped notes features to get any kind of notehead that Mup supports anywhere you want.

```
headshapes
    // Make some user-defined head shapes.
    // These combinations don't really make sense;
    // they are just to demonstrate what you can do.
    "reg" "4n 2n 2n 2n"
    "other" "fillrectangle diamond isostriangle dblwhole"

score
    // Set notesheads, using an arbitrary mixture of built-in
    // and user-defined head shapes.
    noteheads="norm reg other reg reg other righttri"

music

1: c;d;2e;
bar

1: 2g;f;
bar

1: {b;a;g;}3;2c+;
bar

1: 1d;
bar

// do some shape overrides, both on group and note
1: [ hs "pie" ]2ce; [hs "isostri"] c e g hs "righttri";
endbar
```



8. SHAPE OVERRIDES AND MENSURAL NOTATION

The shapes context can be used to define overrides for certain music symbols. The overrides can then be applied, via the shapes parameter, either globally or to specific staffs, or, for symbols for which it could make sense, to specific voices. Most users will probably never use this facility, so if you are satisfied with the appearance of Mup's standard music symbols, feel free to skip this section.

These overrides apply only when Mup is choosing the symbols to use. I.e., they will not apply when symbols are explicitly specified in strings, like for rom, ital, bold, boldital, mussym, print, left, center, right, noteleft, or "with lists." If you want a symbol to be overridden everywhere, see the section on user-defined symbols.

The syntax of the shapes context is:

shapes "*name*"
"*symbol1*" "*replacement1*"
"*symbol2*" "*replacement2*"

The *name* is an arbitrary string which can then be used as the value of the shapes parameter. It is followed by one or more pairs of strings, with each pair on a separate line. Any subset of the allowable characters can be overridden in a given shapes context. Any not overridden will retain their normal values. As many shapes contexts as desired can be defined.

The symbols which can be overridden in score, staff, or voice context are:

upflag	dnflag	ll1rest	ll2rest
owhrest	qwhrest	dwhrest	1rest
2rest	4rest	8rest	16rest
32rest	64rest	128rest	256rest

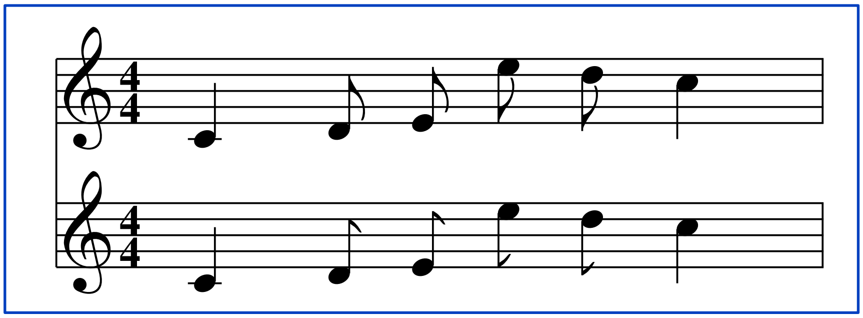
The following symbols are only used in score or staff context, so while they can be overridden in any shapes context, if that shapes context is used at voice level, the overrides of these symbols would have no effect there:

fclef	gclef	cclef
measrpt	dblmeasrpt	quadmeasrpt
sharp	flat	nat
dblsharp	dblflat	
cut	com	

Note that while Mup will take the bounding boxes of overridden symbols into account, there are certain cases where it uses special knowledge of the characteristics of built-in symbols, to improve the aesthetics of their placement. If a user-defined symbol is substituted, and it has different characteristics, its placement may not look as good as the default symbol.

The replacements will likely often be user defined symbols, though built-in symbols may also be used. One example of using built-in symbols could be if you wanted to use the more simple, mensural style flags, but only on one staff, so that overriding the flags symbols on a global level would not work. For that, you could do something like:

```
shapes "simpleflags"
  "upflag" "mensurupflag"
  "dnflag" "mensurdnflag"
score
  staffs=2
staff 2
  shapes="simpleflags"
music

1-2: c;8d;e;e+;d+;4c+;
bar
```



As an example of a user-defined symbol as the replacement, there are several variations of the C clef symbol which have been used over the centuries, and the C clef symbol is used for several clefs. Suppose you wanted only the alto clef usage to look different. You could define that different appearance symbol using the symbol context, then have a shapes context:

```
shapes "alt C clef"
  "cclef"  "mycclef"
```

then define a macro:

```
  define ALTO_CLEF  clef=alto ; shapes="alt C clef" @
```

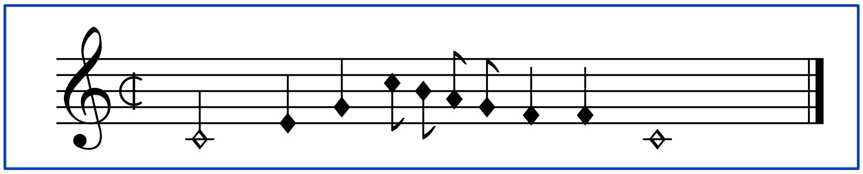
and then use that macro in the parameters section for appropriate staffs.

There is a pre-defined "mensural" shapes context, which is:

```
shapes "mensural"
    "dwhrest" "mensurdwhrest"
    "1rest" "mensur1rest"
    "ll1rest" "mensurll1rest"
    "2rest" "mensur2rest"
    "ll2rest" "mensurll2rest"
    "4rest" "mensur4rest"
    "8rest" "mensur8rest"
    "16rest" "mensur16rest"
    "upflag" "mensurupflag"
    "dnflag" "mensurdnflag"
```

So to produce mensural notation, the recommended usage is:

```
score // or just the staff(s) where mensural notation is desired
     shapes = "mensural"
     noteheads = "mensural"
     stemlen = 6
     time = 3/2  // or as appropriate
     printedtime = "\(imperfminordim)" // or one of the other 7 choices
music

1: c;4e;g;8c+;b;a;g;
invisbar

1: 4f;;1c;
endbar
```



9. TEXT STRINGS

Text strings are used in many different ways. A text string is enclosed in double quotes, and can contain any combination of letters, numbers, spaces, and punctuation. It can contain a \n to indicate a newline or \b to indicate a backspace, but otherwise cannot contain tabs or other control characters. If you want a double quote mark inside a string, it has to be preceded by a backslash, to indicate it is not ending the string:

```
"A \"word\" in quotes"
```

If you want an actual backslash in a string, it must be entered as two backslashes.

You can concatenate strings with a + sign. Thus the following are equivalent:

```
"this is a string"
"this is " + "a string"
```

9.1 Special characters

A string can also contain special music characters. These are specified by \(*xxx*), where the *xxx* is a name from the following table:

type	name	music symbol
clef	gclef	G clef (treble clef)
	fclef	F clef (bass clef)
	cclef	C clef (used for alto clef, tenor clef, etc.)
time sig	com	common time
	cut	cut time
	perfmaior	perfectum maior
	perfminor	perfectum minor
	imperfmaior	imperfectum maior
	imperfminor	imperfectum minor
	perfmaiordim	perfectum maior diminutum
	perfminordim	perfectum minor diminutum
	imperfmaiordim	imperfectum maior diminutum
	imperfminordim	imperfectum minor diminutum
accidental	flat	flat
	dblflat	double flat
	sharp	sharp
	dblsharp	double sharp
	nat	natural
note	4n	quarter (and shorter) note notehead
	2n	half note notehead
	1n	whole note
	dblwhole	double whole note
	altdblwhole	alternate double whole
	quadwhole	quadruple whole note (longa)
	octwhole	octuple whole note (maxima)
	dn2n	half note with stem down
	dn4n	quarter note with stem down
	dn8n	eighth note with stem down
	dn16n	16th note with stem down

	dn32n	32nd note with stem down
	dn64n	64th note with stem down
	dn128n	128th note with stem down
	dn256n	256th note with stem down
	up2n	half note with stem up
	up4n	quarter note with stem up
	up8n	eighth note with stem up
	up16n	16th note with stem up
	up32n	32nd note with stem up
	up64n	64th note with stem up
	up128n	128th note with stem up
	up256n	256th note with stem up
	upflag	upward flag
	dnflag	downward flag
	mensurupflag	mensural upward flag
	mensurdnflag	mensural downward flag
notehead		
	xnote	X-shaped notehead
	diamond	open diamond-shaped notehead
	filldiamond	filled diamond-shaped notehead
	dwhdiamond	double whole diamond-shaped notehead
	isostriangle	open isosceles triangle notehead
	fillisostriangle	filled isosceles triangle notehead
	dwhisostriangle	double whole isosceles triangle notehead
	piewedge	open piewedge notehead
	fillpiewedge	filled piewedge notehead
	dwhpiewedge	double whole piewedge notehead
	rectangle	open rectangle notehead
	fillrectangle	filled rectangle notehead
	dwhrectangle	double whole rectangle notehead
	righttriangle	open right triangle notehead
	fillrighttriangle	filled right triangle notehead
	dwhrighttriangle	double whole right triangle notehead
	urighttriangle	upside-down open right triangle notehead
	ufillrighttriangle	upside-down filled right triangle notehead
	udwhrighttriangle	upside-down double whole right triangle notehead
	semicircle	open semicircle notehead
	fillsemicircle	filled semicircle notehead
	dwhsemicircle	double whole semicircle notehead
	slashhead	open slash notehead
	fillslashhead	filled slash notehead
	dwhslashhead	double whole slash notehead
	blankhead	blank notehead
	mensurdiamond	mensural open diamond notehead
	mensurfilldiamond	mensural filled diamond notehead
	mensurdblwhole	mensural double whole notehead
rest	owhrest	octuple whole rest
	qwhrest	quadruple whole rest
	dwhrest	double whole rest
	1rest	whole rest
	ll1rest	ledger-less whole rest

	2rest	half rest
	ll2rest	ledger-less half rest
	4rest	quarter rest
	8rest	eighth rest
	16rest	sixteenth rest
	32rest	thirty-second rest
	64rest	sixty-fourth rest
	128rest	128th rest
	256rest	256th rest
	mensurdwhrest	mensural double whole rest
	mensur1rest	mensural whole rest
	mensurll1rest	mensural ledger-less whole rest
	mensur2rest	mensural half rest
	mensurll2rest	mensural ledger-less half rest
	mensur4rest	mensural quarter rest
	mensur8rest	mensural eighth rest
	mensur16rest	mensural sixteenth rest
pedal	begped	begin pedal mark
	endped	end pedal mark
	pedal	pedal up/down mark
ornaments	tr	trill
	mor	mordent
	invmor	inverted mordent
	turn	turn
	invturn	inverted turn
misc	ferm	fermata
	uferm	upside-down fermata
	acc_gt	accent like a greater-than sign
	acc_hat	accent like a "hat" (circumflex or "up-arrow")
	acc_uhat	accent like an upside-down hat
	leg	legato mark
	dot	dot
	wedge	wedge
	uwedge	upside-down wedge
	sign	sign for D. S.
	coda	coda mark
	upbow	up bow
	dnbow	down bow
	rr	"railroad tracks" or caesura (2 slanted lines sometimes put at the top of a staff to indicate the end of a musical thought)
	measrpt	measure repeat
	dblmeasrpt	double measure repeat
	quadmeasrpt	quadruple measure repeat
	copyright	C-in-circle copyright symbol
	dim	diminished
	halfdim	half diminished
	triangle	triangle

gclef	fclef	cclef	octwhole	quadwhole	dblwhole	altdblwhole	1n	2n	4n	upflag	dnflag
owhrest	qwhrest	dwhrest	1rest	2rest	4rest	8rest	16rest	32rest	64rest	128rest	256rest
l1rest	l2rest	dot	nat	sharp	flat	dblsharp	dblflat	xnote	dwhdiamond	diamond	filldiamond
up2n	dn2n	up4n	dn4n	com	cut	begped	pedal	endped	dnbow	upbow	wedge
uwedge	ferm	uferm	sign	coda	mor	invmor	turn	invturn	acc_gt	acc_hat	acc_uhat
tr	leg	rr	dim	halfdim	triangle	measrpt	copyright	up8n	dn8n	up16n	dn16n
up32n	dn32n	up64n	dn64n	up128n	dn128n	up256n	dn256n	dwhrighttriangle	righttriangle	fillrighttriangle	udwhrighttriangle
urighttriangle	ufillrighttriangle	dwhrectangle	rectangle	fillrectangle	dwhisostriangle	isostriangle	fillisostriangle	dwhpiewedge	piewedge	fillpiewedge	dwhsemicircle
semicircle	fillsemicircle	dwhslashhead	slashhead	fillslashhead	blankhead	perfmaior	perfminor	imperfmaior	imperfminor	perfmaiordim	perfminordim
imperfmaiordim	imperfminordim	dblmeasrpt	quadmeasrpt	mensurdiamond	mensurfilldiamond	mensurdblwhole	mensurdwhrest	mensur1rest	mensurl1rest	mensur2rest	mensurl2rest
mensur4rest	mensur8rest	mensur16rest	mensurupflag	mensurdnflag							

Any of these music character names can be prefixed by "sm" to indicate a smaller version of the character. For example, "smup4n" is a small quarter note, as might be used for a "cue" note. Small music characters are 0.65 times as big as regular characters.

Various non-ASCII characters can be included in text strings. If you have a non-United States keyboard which can produce characters listed in the table below from the "Latin-1" alphabet, you can simply type them into strings as you normally would. If you want a character that your keyboard does not support, you can put them in strings by using their names in a manner similar to the music characters. For example, you can include an "a" with an acute accent on it in a string by using \(aacute), or an upside-down question mark using \(questiondown).

There are 2-character shortcut names for many of the letters with diacritical marks. The shortcut names consist of the letter and a character representing the diacritical mark. So, for example, \(aacute) can also be specified by just \(a'), \(Egrave) can also be specified by \(E`), \(ntilde) can be specified as \(n~), and \(Ocircumflex) can be specified as \(O^). The following table lists the diacritical mark names, their shortcut symbols, and the list of available shortcut names using those symbols:

acute	'	A' a' C' c' E' e' I' i' L' l' N' n' O' o' R' r' S' s' U' u' Y' y' Z' z'
breve	(	A(a(E(e(G(g(I(i(O(o(U(u(
caron	v	Cv cv Dv dv Ev ev Lv lv Nv nv Rv rv Sv sv Tv tv Zv zv
cedilla	,	C, c, S, s,
circumflex	^	A^ a^ C^ c^ E^ e^ G^ g^ H^ h^ I^ i^ J^ j^ O^ o^ S^ s^ U^ u^ W^ w^ Y^ y^
dieresis	:	A: a: E: e: I: i: O: o: U: u: Y: y:
dotaccent	.	C. c. E. e. G. g. I. A. z.
grave	`	A` a` E` e` I` i` O` o` U` u`
macron	-	A- a- E- e- I- i- O- o- U- u-
ogonek	c	Ac ac Ec ec Ic ic Uc uc
ring	o	Ao ao Uo uo
slash	/	L/ l/ O/ o/
tilde	~	A~ a~ I~ i~ N~ n~ O~ o~ U~ u~

There are also a few special case shortcut names:

germandbls	ss
quotedblleft	``
quotedblright	''
guillemotleft	<<
guillemotright	>>

The following charts list the full names of all of the available named characters, and shows what they look like.

✁ a1	✂ a2	✄ a3	☎ a4
✆ a5	✝ a6	✞ a7	✟ a8
✠ a9	✡ a10	☛ a11	☞ a12
✌ a13	✍ a14	✎ a15	✏ a16
✑ a17	✒ a18	✓ a19	✔ a20
✕ a21	✖ a22	✗ a23	✘ a24
✙ a25	✚ a26	✛ a27	✜ a28
✢ a29	✣ a30	✤ a31	✥ a32
✦ a33	✧ a34	★ a35	✩ a36
✪ a37	✫ a38	✬ a39	✭ a40
✮ a41	✯ a42	✰ a43	✱ a44
✲ a45	✳ a46	✴ a47	✵ a48
✶ a49	✷ a50	✸ a51	✹ a52
✺ a53	✻ a54	✼ a55	✽ a56
✾ a57	✿ a58	❀ a59	❁ a60
❂ a61	❃ a62	❄ a63	❅ a64
❆ a65	❇ a66	❈ a67	❉ a68
❊ a69	❋ a70	● a71	❍ a72
■ a73	❏ a74	❑ a75	▲ a76
▼ a77	◆ a78	❖ a79	◗ a81
❘ a82	❙ a83	❚ a84	❯ a85
❱ a86	❲ a87	❳ a88	❨ a89
❩ a90	❬ a91	❭ a92	❪ a93
❫ a94	❴ a95	❵ a96	❛ a97
❜ a98	❝ a99	❞ a100	❡ a101
❢ a102	❣ a103	❤ a104	✐ a105
❥ a106	❦ a107	❧ a108	♠ a109
♥ a110	♦ a111	♣ a112	✉ a117
✈ a118	✇ a119	① a120	② a121
③ a122	④ a123	⑤ a124	⑥ a125
⑦ a126	⑧ a127	⑨ a128	⑩ a129
❶ a130	❷ a131	❸ a132	❹ a133
❺ a134	❻ a135	❼ a136	❽ a137
❾ a138	❿ a139	➀ a140	➁ a141
➂ a142	➃ a143	➄ a144	➅ a145
➆ a146	➇ a147	➈ a148	➉ a149
➊ a150	➋ a151	➌ a152	➍ a153
➎ a154	➏ a155	➐ a156	➑ a157
➒ a158	➓ a159	➔ a160	→ a161
➣ a162	↔ a163	↕ a164	➙ a165
➛ a166	➜ a167	➝ a168	➞ a169
➟ a170	➠ a171	➡ a172	➢ a173
➤ a174	➥ a175	➦ a176	➧ a177
➨ a178	➩ a179	➫ a180	➭ a181
➯ a182	➲ a183	➳ a184	➵ a185
➸ a186	➺ a187	➻ a188	➼ a189
➽ a190	➾ a191	➚ a192	➪ a193
➶ a194	➹ a195	➘ a196	➴ a197
➷ a198	➬ a199	➮ a200	➱ a201
✃ a202	❐ a203	❒ a204	❮ a205

❰	a206	á	aacute	Á	Aacute	ă	abreve
Ă	Abreve	â	acircumflex	Â	Acircumflex	´	acute
ä	adieresis	Ä	Adieresis	æ	ae	Æ	AE
А	afii10017	Б	afii10018	В	afii10019	Г	afii10020
Д	afii10021	Е	afii10022	Ё	afii10023	Ж	afii10024
З	afii10025	И	afii10026	Й	afii10027	К	afii10028
Л	afii10029	М	afii10030	Н	afii10031	О	afii10032
П	afii10033	Р	afii10034	С	afii10035	Т	afii10036
У	afii10037	Ф	afii10038	Х	afii10039	Ц	afii10040
Ч	afii10041	Ш	afii10042	Щ	afii10043	Ъ	afii10044
Ы	afii10045	Ь	afii10046	Э	afii10047	Ю	afii10048
Я	afii10049	Ґ	afii10050	Ђ	afii10051	Ѓ	afii10052
Є	afii10053	Ѕ	afii10054	І	afii10055	Ї	afii10056
Ј	afii10057	Љ	afii10058	Њ	afii10059	Ћ	afii10060
Ќ	afii10061	Ў	afii10062	а	afii10065	б	afii10066
в	afii10067	г	afii10068	д	afii10069	е	afii10070
ё	afii10071	ж	afii10072	з	afii10073	и	afii10074
й	afii10075	к	afii10076	л	afii10077	м	afii10078
н	afii10079	о	afii10080	п	afii10081	р	afii10082
с	afii10083	т	afii10084	у	afii10085	ф	afii10086
х	afii10087	ц	afii10088	ч	afii10089	ш	afii10090
щ	afii10091	ъ	afii10092	ы	afii10093	ь	afii10094
э	afii10095	ю	afii10096	я	afii10097	ґ	afii10098
ђ	afii10099	ѓ	afii10100	є	afii10101	ѕ	afii10102
і	afii10103	ї	afii10104	ј	afii10105	љ	afii10106
њ	afii10107	ћ	afii10108	ќ	afii10109	ў	afii10110
Џ	afii10145	џ	afii10193	ә	afii10846	№	afii61352
à	agrave	À	Agrave	ℵ	aleph	α	alpha
Α	Alpha	ā	amacron	Ā	Amacron	∠	angle
〈	angleleft	〉	angleright	ą	aogonek	Ą	Aogonek
≈	approxequal	å	aring	Å	Aring	↔	arrowboth
⇔	arrowdblboth	⇓	arrowdbldown	⇐	arrowdblleft	⇒	arrowdblright
⇑	arrowdblup	↓	arrowdown	⎯	arrowhorizex	←	arrowleft
→	arrowright	↑	arrowup	⏐	arrowvertex	∗	asteriskmath
ã	atilde	Ã	Atilde	β	beta	Β	Beta
⎪	braceex	⎩	braceleftbt	⎨	braceleftmid	⎧	bracelefttp
⎭	bracerightbt	⎬	bracerightmid	⎫	bracerighttp	⎣	bracketleftbt
⎢	bracketleftex	⎡	bracketlefttp	⎦	bracketrightbt	⎥	bracketrightex
⎤	bracketrighttp	˘	breve	¦	brokenbar	•	bullet
ć	cacute	Ć	Cacute	ˇ	caron	↵	carriagereturn
č	ccaron	Č	Ccaron	ç	ccedilla	Ç	Ccedilla
ĉ	ccircumflex	Ĉ	Ccircumflex	ċ	cdotaccent	Ċ	Cdotaccent
¸	cedilla	¢	cent	χ	chi	Χ	Chi
⊗	circlemultiply	⊕	circleplus	ˆ	circumflex		commaaccent
≅	congruent	©	copyright	©	copyrightsans	©	copyrightserif
¤	currency	†	dagger	‡	daggerdbl	ď	dcaron
Ď	Dcaron	đ	dcroat	Đ	Dcroat	°	degree
δ	delta	Δ	Delta	¨	dieresis	÷	divide
˙	dotaccent	ı	dotlessi	·	dotmath	é	eacute
É	Eacute	ĕ	ebreve	Ĕ	Ebreve	ě	ecaron

Ě	Ecaron	ê	ecircumflex	Ê	Ecircumflex	ë	edieresis
Ë	Edieresis	ė	edotaccent	Ė	Edotaccent	è	egrave
È	Egrave	∈	element	…	ellipsis	ē	emacron
Ē	Emacron	—	emdash	∅	emptyset	–	endash
ŋ	eng	Ŋ	Eng	ę	eogonek	Ę	Eogonek
ε	epsilon	Ε	Epsilon	≡	equivalence	η	eta
Η	Eta	ð	eth	Đ	Eth	€	Euro
¡	exclamdown	∃	existential	ﬁ	fi	ﬂ	fl
ƒ	florin	⁄	fraction	γ	gamma	Γ	Gamma
ğ	gbreve	Ğ	Gbreve	ĝ	gcircumflex	Ĝ	Gcircumflex
ģ	gcommaaccent	Ģ	Gcommaaccent	ġ	gdotaccent	Ġ	Gdotaccent
ß	germandbls	∇	gradient	`	grave	≥	greaterequal
«	guillemotleft	»	guillemotright	‹	guilsinglleft	›	guilsinglright
ħ	hbar	Ħ	Hbar	ĥ	hcircumflex	Ĥ	Hcircumflex
♥	heart	˝	hungarumlaut	í	iacute	Í	Iacute
ĭ	ibreve	Ĭ	Ibreve	î	icircumflex	Î	Icircumflex
ï	idieresis	Ï	Idieresis	İ	Idotaccent	ℑ	Ifraktur
ì	igrave	Ì	Igrave	ĳ	ij	Ĳ	IJ
ī	imacron	Ī	Imacron	∞	infinity	∫	integral
⌡	integralbt	⎮	integralex	⌠	integraltp	∩	intersection
į	iogonek	Į	Iogonek	ι	iota	Ι	Iota
ĩ	itilde	Ĩ	Itilde	ĵ	jcircumflex	Ĵ	Jcircumflex
κ	kappa	Κ	Kappa	ķ	kcommaaccent	Ķ	Kcommaaccent
ĸ	kgreenlandic	ĺ	lacute	Ĺ	Lacute	λ	lambda
Λ	Lambda	ľ	lcaron	Ľ	Lcaron	ļ	lcommaaccent
Ļ	Lcommaaccent	ŀ	ldot	Ŀ	Ldot	≤	lessequal
∧	logicaland	¬	logicalnot	∨	logicalor	ſ	longs
◊	lozenge	ł	lslash	Ł	Lslash	¯	macron
−	minus	′	minute	μ	mu	Μ	Mu
×	multiply	ń	nacute	Ń	Nacute	ŉ	napostrophe
ň	ncaron	Ň	Ncaron	ņ	ncommaaccent	Ņ	Ncommaaccent
∉	notelement	≠	notequal	⊄	notsubset	ñ	ntilde
Ñ	Ntilde	ν	nu	Ν	Nu	ó	oacute
Ó	Oacute	ŏ	obreve	Ŏ	Obreve	ô	ocircumflex
Ô	Ocircumflex	ö	odieresis	Ö	Odieresis	œ	oe
Œ	OE	˛	ogonek	ò	ograve	Ò	Ograve
ő	ohungarumlaut	Ő	Ohungarumlaut	ō	omacron	Ō	Omacron
ω	omega	Ω	Omega	ϖ	omega1	ο	omicron
Ο	Omicron	½	onehalf	¼	onequarter	¹	onesuperior
ª	ordfeminine	º	ordmasculine	ø	oslash	Ø	Oslash
õ	otilde	Õ	Otilde	¶	paragraph	⎝	parenleftbt
⎜	parenleftex	⎛	parenlefttp	⎠	parenrightbt	⎟	parenrightex
⎞	parenrighttp	∂	partialdiff	·	periodcentered	⊥	perpendicular
‰	perthousand	ϕ	phi	Φ	Phi	φ	phi1
π	pi	±	plusminus	∏	product	⊂	propersubset
⊃	propersuperset	∝	proportional	ψ	psi	Ψ	Psi
¿	questiondown	„	quotedblbase	‟	quotedblleft	″	quotedblright
‚	quotesinglbase	'	quotesingle	ŕ	racute	Ŕ	Racute
√	radical	‾	radicalex	ř	rcaron	Ř	Rcaron
ŗ	rcommaaccent	Ŗ	Rcommaaccent	⊆	reflexsubset	⊇	reflexsuperset

®	registered	®	registersans	®	registerserif	ℜ	Rfraktur
ρ	rho	˚	ring	ś	sacute	Ś	Sacute
š	scaron	Š	Scaron	ş	scedilla	Ş	Scedilla
ŝ	scircumflex	Ŝ	Scircumflex	ș	scommaaccent	Ș	Scommaaccent
″	second	§	section	σ	sigma	ς	sigma1
∼	similar		space	£	sterling	∍	suchthat
∑	summation	τ	tau	Τ	Tau	ŧ	tbar
Ŧ	Tbar	ť	tcaron	Ť	Tcaron	ţ	tcommaaccent
Ţ	Tcommaaccent	∴	therefore	θ	theta	ϑ	theta1
þ	thorn	Þ	Thorn	¾	threequarters	³	threesuperior
˜	tilde	™	trademark	™	trademarksans	™	trademarkserif
²	twosuperior	ú	uacute	Ú	Uacute	ŭ	ubreve
Ŭ	Ubreve	û	ucircumflex	Û	Ucircumflex	ü	udieresis
Ü	Udieresis	ù	ugrave	Ù	Ugrave	ű	uhungarumlaut
Ű	Uhungarumlaut	ū	umacron	Ū	Umacron		uni0162
	uni0163	Ѐ	uni0400	Ѝ	uni040D	ѐ	uni0450
ѝ	uni045D		uni048C		uni048D		uni048E
	uni048F	Ғ	uni0492	ғ	uni0493		uni0494
	uni0495	Җ	uni0496	җ	uni0497	Ҙ	uni0498
ҙ	uni0499	Қ	uni049A	қ	uni049B	Ҝ	uni049C
ҝ	uni049D		uni049E		uni049F	Ҡ	uni04A0
ҡ	uni04A1	Ң	uni04A2	ң	uni04A3		uni04A4
	uni04A5		uni04A6		uni04A7		uni04A8
	uni04A9	Ҫ	uni04AA	ҫ	uni04AB		uni04AC
	uni04AD	Ү	uni04AE	ү	uni04AF	Ұ	uni04B0
ұ	uni04B1	Ҳ	uni04B2	ҳ	uni04B3		uni04B4
	uni04B5	Ҷ	uni04B6	ҷ	uni04B7	Ҹ	uni04B8
ҹ	uni04B9	Һ	uni04BA	һ	uni04BB		uni04BC
	uni04BD		uni04BE		uni04BF	Ӏ	uni04C0
	uni04C1		uni04C2		uni04C3		uni04C4
	uni04C7		uni04C8	Ӌ	uni04CB	ӌ	uni04CC
	uni04D0		uni04D1		uni04D2		uni04D3
	uni04D4		uni04D5		uni04D6		uni04D7
Ә	uni04D8		uni04DA		uni04DB		uni04DC
	uni04DD		uni04DE		uni04DF		uni04E0
	uni04E1	Ӣ	uni04E2	ӣ	uni04E3		uni04E4
	uni04E5		uni04E6		uni04E7	Ө	uni04E8
ц	uni04E9		uni04EA		uni04EB		uni04EC
	uni04ED	Ӯ	uni04EE	ӯ	uni04EF		uni04F0
	uni04F1		uni04F2		uni04F3		uni04F4
	uni04F5		uni04F8		uni04F9	∪	union
∀	universal	ų	uogonek	Ų	Uogonek	υ	upsilon
Υ	Upsilon	ϒ	Upsilon1	ů	uring	Ů	Uring
ũ	utilde	Ũ	Utilde	ŵ	wcircumflex	Ŵ	Wcircumflex
℘	weierstrass	ξ	xi	Ξ	Xi	ý	yacute
Ý	Yacute	ŷ	ycircumflex	Ŷ	Ycircumflex	ÿ	ydieresis
Ÿ	Ydieresis	¥	yen	ź	zacute	Ź	Zacute
ž	zcaron	Ž	Zcaron	ż	zdotaccent	Ż	Zdotaccent
ζ	zeta	Ζ	Zeta				

One of the special characters is specified by \(space). This character appears as a normal space on output, but is not treated like a space when Mup is looking for word or syllable boundaries. The most common use for this is probably in lyrics when you want several words to be sung on a single note. Another use would be in cases where Mup would normally split up a long string between words in order to avoid running off the edge of a page, but you want to prevent that split.

9.2 Keymaps

Typing in the names for the non-ASCII characters can become tedious, so if you are using some of them often, defining a keymap may be helpful. The keymap context is typically used to take the letters you type in, and map them to some other alphabet, but it could be used for other kinds of mapping inside strings as well. You can define up to 100 different keymaps, and you can assign different mappings to different kinds of text. You give each keymap a name, and then can set parameters to that name to cause mapping.

A keymap context begins with a line with the word "keymap," followed by a string in double quotes, giving a name for the map. The name can be anything you like. This is followed by lines containing pairs of strings. The first in each pair is a pattern to be matched, and the second is the replacement. The pattern strings can only contain letters (uppercase or lowercase) and the equals sign. The replacement strings can only contain regular characters or the special named characters of the form \(XXX). They cannot contain other "backslash escapes" like changes in font or size. Mapping is done as strings are parsed, not when they are printed.

As an example, you could set up a mapping to allow something close to phonetic spelling on an English keyboard, but have the results come out in Cyrillic. The Mup distribution has an "include" file for one possible mapping of the full Cyrillic alphabet, but as a small example, to get the Russian word for "song" you could do

```
keymap "Russian"
    // map approximate phonetic equivalents to their Cyrillic symbols
    "e"  "\(afii10070)"
    "n"  "\(afii10079)"
    "p"  "\(afii10081)"
    "s"  "\(afii10083)"
    "q"  "\(afii10097)"

score
    defaultkeymap="Russian"

music
    // enter Russian string phonetically
    title "Cyrillic: pesnq"
```

Cyrillic: песня

In the previous example, the text string to be matched was only a single character, and the replacement string was also a single character, albeit one that normally has to be specified by a long name. Either or both can actually be multiple characters. So you could do something like

```
keymap "names"
  "Ted"  "Theodore"
  "Liz"  "Elizabeth"
```

and then "Ted and Liz" would become "Theodore and Elizabeth"

As another example, suppose you want to use various styles of arrows in different places, and would like to have shorter names. You could do something like this:

```
keymap "arrows"
    "=l"  "\(arrowleft)"
    "=r"  "\(arrowright)"
    "==l"  "\(arrowdblleft)"
    "==r"  "\(arrowdblright)"
    "=ll"  "\(arrowleft)\(arrowleft)"
    "=rr"   "\(arrowright)\(arrowright)"
score
    defaultkeymap="arrows"

block
  paragraph "=r between arrows =l";
  paragraph "==r between double arrows ==l";
  paragraph "=rr between pairs of arrows =ll";
```

→ between arrows ←
⇒ between double arrows ⇐
→→ between pairs of arrows ←←

Patterns are matched left to right, using the longest pattern that matches. Thus for a pattern/replacement list of

```
"a"   "X"
"b"   "Y"
"ab"  "Z"
```

the string "abba" would become "ZYX"

There are a number of parameters that specify what kind of text strings to map. These parameters are:

PARAMETER	WHEN USED
defaultkeymap	when nothing else applies
endingkeymap	text on endings
labelkeymap	staff and group labels
lyricskeymap	lyrics
printkeymap	print, left, right, center, title
rehearsalkeymap	user-defined rehearsal marks
textkeymap	rom, ital, bold, boldital
withkeymap	"with" lists on chords

Having these different parameters generally makes it easy to do things like set a mapping for lyrics and titles, but leave musical directions like "Allegro" unmapped. To determine which mapping to use, Mup first looks up the specific parameter, and if that is not set, then it will fall back to using the value of the defaultkeymap parameter. Thus, for example, if you wanted only lyrics to be in Greek, with everything else left unmapped, you could define a Greek keymap and set

```
lyricskeymap="Greek"
```

Or if you wanted everything except "with" lists to be Cyrillic, you could do something like

```
defaultkeymap="Cyrillic"
withkeymap=""
```

Note that the special name "" is used to mean "no mapping." You can also set these parameters to nothing at all, which effectively "unsets" them. So if you later wanted "with" lists to also be in Cyrillic like everything else, you could just do

```
withkeymap=
```

which basically "unhides" the defaultkeymap value that had been overridden by the withkeymap value.

It is possible to change keymaps in the middle of a string by using \m(keymap_name). This might be used if you want just part of a sentence in a different alphabet:

```
"This would be written \m(Russian)pesnq \m() in Russian."
```

As shown, an empty keymap name of \m() means to return to normal literal text, without any mapping.

9.3 Font and size changes

Strings can contain special codes to indicate changes in font or size. The font codes begin with \f, and can either be spelled out, as in the first column of the table below, or as the abbreviations given in the second column.

Code	Abbr	Meaning
\f(avantgarde rom)	\f(AR)	change to Avant Garde roman font
\f(avantgarde bold)	\f(AB)	change to **Avant Garde bold font**
\f(avantgarde ital)	\f(AI)	change to *Avant Garde italic font*
\f(avantgarde boldital)	\f(AX)	change to ***Avant Garde bold-italic font***
\f(bookman rom)	\f(BR)	change to Bookman roman font
\f(bookman bold)	\f(BB)	change to **Bookman bold font**
\f(bookman ital)	\f(BI)	change to *Bookman italic font*
\f(bookman boldital)	\f(BX)	change to ***Bookman bold-italic font***
\f(courier rom)	\f(CR)	change to Courier roman font
\f(courier bold)	\f(CB)	change to **Courier bold font**
\f(courier ital)	\f(CI)	change to *Courier italic font*
\f(courier boldital)	\f(CX)	change to ***Courier bold-italic font***
\f(helvetica rom)	\f(HR)	change to Helvetica* roman font
\f(helvetica bold)	\f(HB)	change to **Helvetica bold font**
\f(helvetica ital)	\f(HI)	change to *Helvetica italic font*
\f(helvetica boldital)	\f(HX)	change to ***Helvetica bold-italic font***
\f(newcentury rom)	\f(NR)	change to New Century roman font
\f(newcentury bold)	\f(NB)	change to **New Century bold font**
\f(newcentury ital)	\f(NI)	change to *New Century italic font*
\f(newcentury boldital)	\f(NX)	change to ***New Century bold-italic***
\f(palatino rom)	\f(PR)	change to Palatino roman font
\f(palatino bold)	\f(PB)	change to **Palatino bold font**
\f(palatino ital)	\f(PI)	change to *Palatino italic font*
\f(palatino boldital)	\f(PX)	change to ***Palatino bold-italic font***
\f(times rom)	\f(TR)	change to Times* roman font
\f(times bold)	\f(TB)	change to **Times bold font**
\f(times ital)	\f(TI)	change to *Times italic font*
\f(times boldital)	\f(TX)	change to ***Times bold-italic font***
\f(previous)	\f(PV)	change back to previous font
\f(rom)	\f(R)	change to roman in the current family
\f(ital)	\f(I)	change to *italics* in the current family
\f(bold)	\f(B)	change to **bold** in the current family
\f(boldital)	\f(X)	change to ***bold italics*** in the current family

Some of the special characters look better in some fonts than others. We have found that a few older PostScript interpreters unfortunately don't always implement all the special characters in all fonts, so if you have one of those, you may want to see if a newer version is available that corrects the problem.

---

* Times is a trademark and Helvetica is a registered trademark of Allied Corporation.

If you need a font other than those Mup supports directly, it is possible to override Mup's built-in fonts with other fonts. This is described later in the section on "Installing other fonts."

The point size can also be changed. (A "point" is about 1/72 of an inch.)

Code	Meaning
\s(*N*)	change to point size *N*
\s(+*N*)	increase point size by *N* points
\s(-*N*)	decrease point size by *N* points
\s(PV) or \s(previous)	revert to previous size

The point size can range from 1 to 100. A font or size change will last until changed or until the end of the string. Any subsequent strings will begin with default font and size values, except in the case of lyrics, where font and size information is maintained separately for each staff and verse, and carried forward from measure to measure, and rehearsal marks, where the information is also carried forward. The default values can be set as described in the "Parameters" section. The music symbols are affected by size changes. If a music symbol follows a specification for italics, bold, or bolditalics, it will be slanted and/or made bold, as appropriate.

Here are some examples of strings:

```
"hello"
"( \(up2n) = 100 )"
"\f(TB)this will be bold. \f(TI)this will be italics"
"A \f(TX)\s(+12)BIG\f(PV)\s(-12) word"
"\f(newcentury boldital)Allegro"
```

9.4 Horizontal and vertical motion

It is also possible to specify a string that takes up more than one line, by putting a "\n" where you want to move to a new line. The place where this is most likely to be useful is for staff labels, which you may want to make multi-line, to keep them from becoming excessively wide. For example:

```
"Violins\nI&II"
```

A "\b" can be used to specify a backspace. This might be useful for adding underlines to text. If you wish to use letters with diacritical marks, you will probably want to use the special characters, described earlier in this section, rather than trying to construct them using the plain letter, a backspace, and the mark.

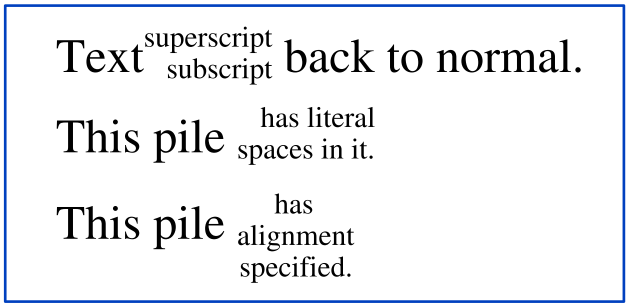
You are not allowed to back up to before the beginning of a line. Note that in all fonts except Courier, different characters have different widths, so using backspaces can be a bit tricky. For example, underlining a 5-letter word might require more than 5 underscores, if the characters in the word are wider than an underscore character. In general, it works better to enter an entire string, then backspace to add underscoring or accents, rather than doing them along the way. For example, "My\b\b___" will line up much better than "M\b_y\b_".

Vertical motion within a string can be specified using "\v(*N*)" where *N* is some percentage of the current font height. The distance may be negative for downward motion or positive for upward motion, ranging from -100 to 100. This might be used for creating superscripts and subscripts, although the next section describes an alternate way of doing that which is usually better. It might also be used just to line something up differently than Mup normally would. You cannot put newlines (with \n) in the same string with vertical motion.

9.5 Piled text, for superscripts, subscripts, etc.

It is also possible to "pile up" lines of text in a string. Some common uses of this could be for superscripts and subscripts or for figured bass notation. Some facilities to specifically deal with figured bass will be described in a later section, but here we describe the general-purpose constructs for piling text. A \: is used to indicate the beginning of piling. Any subsequent instances of \: in the same string will alternately turn piling off and on. By default, when a pile is begun, the text size is automatically made smaller, and the text baseline is moved up, so that the first line of piled text is like a superscript, and the next like a subscript. If there are additional lines, they are placed below the subscript. How much the size is adjusted is controlled by the pilescale parameter. By default, the lines in a pile are placed such that the last digit in each line will line up, or if there is no digit, the last character. However, you can force different alignment. A \| will force alignment at that point, or a \^ will force alignment at the center of the following character. A maximum of one alignment marker can be specified on each line of a pile. You cannot put newlines (with \n) in the same string with piling. A new line of the pile is started at each <space> character in the input string. If you want an actual literal space inside a piled line, it must be preceded with a backslash. As an example:

```
"Text\:superscript subscript\: back to normal.";
"This pile \:has\ literal spaces\ in\ it.";
"This pile \:h\^as align\|ment spec\^ified.";
```



9.6 Slash through number

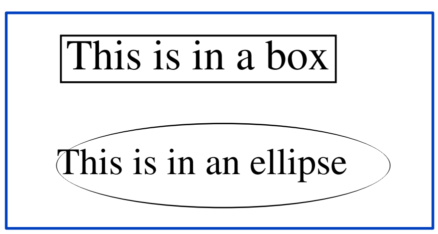
Another thing which is common in figured bass notation is to draw a slash through a number. Again, the section on figured bass will describe how to do that inside figured bass, but it is possible to put a slash through a number elsewhere by placing a backslashed slash after a number, like this:

```
"6\/"
"10\/"
```

9.7 Boxed or circled text

You can cause the text to be printed inside a box by placing a \[at the beginning of the string and a \] at the end of the string. Similarly you can cause text to be placed inside a circle (or ellipse for strings that are wide) by placing a \{ at the beginning of the string and a \} at the end of the string. Boxed or circled text are not allowed in lyrics.

```
"\[This is in a box\]"
"\{This is in an ellipse\}"
```



10. TEMPO, DYNAMIC MARKS, ORNAMENTS, ETC.

10.1 General information

There are a number of symbols and other markings that appear frequently in music and are normally associated with a particular time or place in the composition. There are several classes of such symbols, all of which are handled by Mup in a somewhat similar fashion. The general format of these statements is:

mark_type place staff(s) distance aligntag : begintime text duration;

The *mark_type* can be any of the following:

Mark_type	Meaning
rom	text in roman font
ital	text in italic font
bold	text in bold font
boldital	text in bold-italic font
mussym	music symbol (fermata, coda sign, etc.)
phrase	phrase mark
<	crescendo "hairpin"
>	decrescendo "hairpin"
octave	play 1 or more octaves higher or lower
pedal	piano pedal marks

The *place* is as for lyrics: above, below, or between. There are some restrictions, as summarized below:

	place allowed			
mark_type	above	below	between	default
rom	yes	yes	yes	above
bold	yes	yes	yes	above
ital	yes	yes	yes	above
boldital	yes	yes	yes	above
mussym	yes	yes	yes	above
phrase	yes	yes	no	varies
<	yes	yes	yes	above
>	yes	yes	yes	above
octave	yes	yes	no	none
pedal	no	yes	no	below

Octave marks must include a *place* of above or below. For all the other mark_types, the *place* is optional. For phrase, if *place* is not specified, Mup determines it on a case-by-case basis, depending on the location of the notes, unless the defaultphraseside parameter is set to force a side.

As with musical data or lyrics, the *staff* can be a single number or may include lists and ranges. In the case of "between", staff numbers must be in pairs, separated by an "&", with the second staff number one greater than the first.

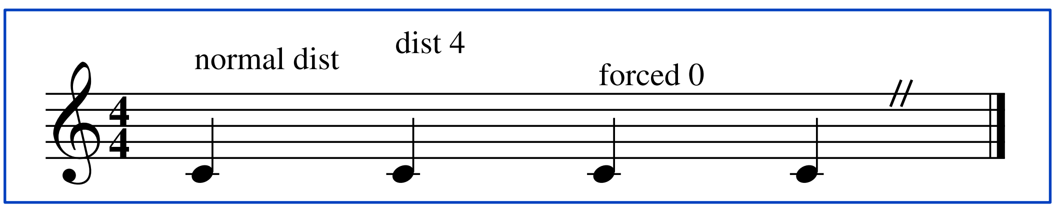
The *staff* can also be specified by the keyword "all," in which case the mark will be placed above the top visible staff or below the bottom visible staff.

Here are some examples:

```
rom 3:
boldital below 1:
< between 3&4:
pedal below 2:
octave above 3:
phrase 3,4:
mussym above 2-3, 5:
ital between 1&2, 3&4:
```

The *distance* is optional. It is specified by the keyword "dist" followed by a number. This number overrides the dist, chorddist, or dyndist parameter value that would normally apply, and is given in stepsizes. If the number is followed by a ! the items will be placed at exactly that distance from the edge of the staff, without regard for anything they might overwrite. Otherwise the normal rules apply: the dist is a minimum value, and items may be placed farther away than this to avoid colliding with other things. If the ! is used, the number is allowed to be negative, which allows you to place items inside the staff. A dist cannot be specified with phrase or between.

```
score
    dist=2
music

rom above 1: 1 "normal dist";
rom above 1 dist 4: 2 "dist 4";       // higher than usual
rom above 1 dist 0! : 3 "forced 0";   // lower than normal
mussym above 1 dist -2! : 4.5 "rr";   // forced down into staff
1: c;;;;
endbar
```



The *aligntag* is optional, and allows forcing several items to be placed at the same vertical position. This will be covered in more detail a bit later.

After the colon comes one or more items to be printed. Each item contains at least a begintime specification. Some may also contain a text string and/or a duration. Each item ends with a semicolon. A newline ends the list of items.

The begintime describes where in time the item should be printed. It is a number ranging from 0 to the numerator (top number) of the time signature plus one. 0 refers to the bar line at the beginning of the current measure, 1 refers to the first beat of the measure, 2 to the second beat, etc., with the maximum value referring to the bar line which ends the measure. A "beat" is whatever time value is given by the denominator (bottom number) of the time signature. For example, in 4/4 or 3/4 time, a beat is a quarter note, whereas in 6/8 or 9/8 time, a beat is an eighth note. Fractional values can be specified using a decimal number. Thus 1.5 is halfway between beats 1 and 2, while 2.66 is about 2/3 of the way between beats 2 and 3. For time signatures that are the addition of two or more fractions, like 4/4 + 3/8, a "beat" is given by the largest denominator.

Since grace notes effectively take zero time, special notation is used to place something relative to a grace note. The *begintime* can be followed by a number of grace notes to "back up" from the beat. The number is a negative number enclosed in parentheses.

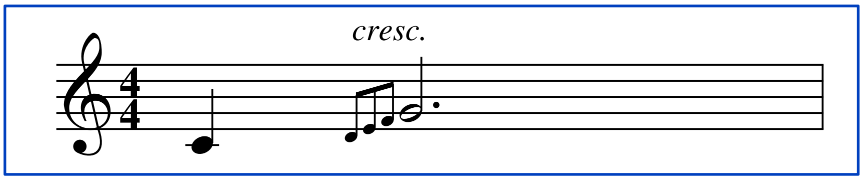
The beat time value can optionally be followed by an offset, in stepsizes. This offset is a number in square brackets. The number must begin with a + or - sign. It can optionally include a decimal point and fractional part. A negative offset will cause the item to be moved to the left, while a positive offset will cause it to be moved to the right. This offset is used as a final adjustment after all the other placement has been done. A common use for this would be if you want to place something slightly before the beginning of the first measure of a song. You could then specify something like:

```
rom above all: 0 [-3.5] "Allegro";
```

which would place the "Allegro" 3.5 stepsizes left of the beginning of the measure.

The grace note backup, if any, must occur before the stepsize offset, if any. For example:

```
1: c; [grace] 8d; []e; []f; 2.g;
ital above 1: 2(-3)[+1] "cresc.";
bar
```



would find the chord at or closest to count 2, and back up 3 grace notes from there, then move right by 1 stepsize.

For items that include a duration, the duration tells the ending point of the item. A duration is specified by:

til *M***m** + *N*

where *M* is a number of bar lines to cross, and *N* is a number of additional beats. (An "m" is used because it counts the number of measures.) If either of them is 0, it can be omitted, but at least one of them must be non-zero.

This can optionally be followed by a grace backup and/or an offset, like the start time. The grace backup is a negative number in parentheses, telling how many grace notes to back up. The steps offset is a number in square brackets, which must begin with a + or - sign, and can optionally include a decimal point and fractional part. A negative offset will cause the item to be moved to the left, while a positive offset will cause it to be moved to the right. This offset is used as a final adjustment after all the other placement has been done. Here are some examples:

```
til 3   // until beat 3 of current measure
til 1m+2   // until beat 2 of next measure
til 1m   // until next bar line
til 3m+2.8[-1.5]  // 3 bar lines and 2.8 beats, then left 1.5 stepsizes
```

Whether the text and duration are allowed or required depends on the mark_type, as summarized below:

mark_type	text	duration
rom	required	optional
bold	required	optional
ital	required	optional
boldital	required	optional
mussym	required	allowed on tr only
phrase	not allowed	required
<	not allowed	required
>	not allowed	required
octave	required	optional
pedal	* or nothing	not allowed

If there are several items, such that they have to be stacked on top of each other, all items of a particular class are placed from top to bottom according to input order. The classes are done in the order shown in the table below, unless the order is overridden by the aboveorder, beloworder, or betweenorder parameters.

Place	mark_type order
above	(from bottom up) phrase mussym octave rom bold ital boldital < > lyrics endings rehearsal marks
below	(from top down) phrase mussym octave rom bold ital boldital < > lyrics pedal
between	(from bottom up) mussym rom bold ital boldital < > lyrics

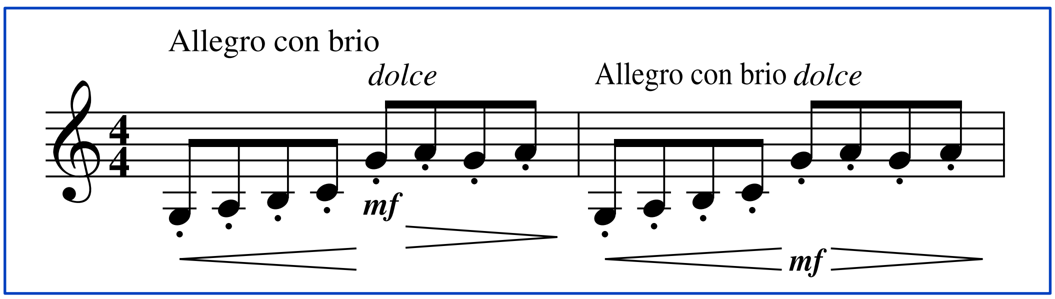
For example, for the above items, all phrases will be placed as close to the staff as possible, then mussym items above them, then octave marks above them, etc. However, if there are several mussym items that belong at the same horizontal position, the first one entered in the input will be printed above the second, the second one entered will be printed above the third, etc. All of the "above all" items of a given class will be above the items of that class for the top visible staff, and all "below all" items of a given class will be below the items of that class for the bottom visible staff.

Additional control of placement can be imposed by the optional align tag. This is the keyword "align" followed by a number between 0 and 10000 inclusive. On a given score, all marks in the same placement order level and having the same align tag value will be placed together, so that they will all be aligned at the same vertical position. An align tag can only be applied to text (rom, ital, bold, boldital) and crescendo and decrescendo marks. It can also be used on pedal, but only when the alignped parameter is set to n. Among marks with the same placement order level, those without alignment are placed first. Then those with alignment are placed, in ascending numerical order of the align tag. If any of the aligned marks also have a dist specified, the rightmost such dist applies to all with that alignment tag, otherwise the largest default distance is used. If any also have ! specified, to force an exact distance, that applies to all.

If after placing a set of aligned marks, Mup determines that any of them overlap horizontally, it will squeeze the one on the left in an attempt to avoid the overlap—to a point. If some overlap remains even after squeezing as much as is allowed by the minalignscale parameter, Mup will issue a warning. You can then adjust things yourself if you wish.

Here is an example showing the impact of using alignment. The two measures are the same, except the second uses alignment tags. Note that since in this example only a single align tag value is used for above and one for below, there is no need for specifying an order among alignment tags, so the numeric values can be arbitrary.

```
score
   beamstyle=2,2

music

rom above 1: 1 "Allegro con brio";
ital above 1: 3 "dolce";
1: [with .]... 8g-;a-;b-;c;g;a;g;a;
< below 1: 1 til 2.8;
boldital dyn below 1: 3 "mf";
> below 1: 3.3 til 4.8;
bar

rom above 1 align 17: 1 "Allegro con brio";
ital above 1 align 17: 3 "dolce";
1: [with .]... 8g-;a-;b-;c;g;a;g;a;
< below 1 align 3: 1 til 2.8;
boldital dyn below 1 align 3: 3 "mf";
> below 1 align 3: 3.3 til 4.8;
bar
```



We now discuss each of the mark_types in more detail.

10.2 Text

The four forms of Mup text statements (rom, bold, ital, and boldital) operate identically except for the font which is used.

The text statements can have several additional qualifiers. The first is a fontfamily, given before the font. It can have any of the values of the "fontfamily" parameter (avantgarde, bookman, courier, helvetica, newcentury, palatino, or times), with the default being the value of the "fontfamily" parameter.

The next optional qualifier is a point size, given in parentheses after the font type of rom, bold, etc. If this is omitted, the size is obtained from the "size" parameter for the given staff, or for the score if the staff is specified by "all."

10.2.1 Chord, analysis, figured bass, and dynamics

The final optional qualifier specifies special treatment of the text. The qualifier can be "chord", "analysis", "figbass", or "dyn". The "chord" modifier is typically used for marking chords that might be played by a guitar or other instrument. The "analysis" modifier is typically used when marking harmonic analysis like "IV" or "vii". The "figbass" modifier is for figured bass notation. For all three, distance from the staff is affected by the "chorddist" parameter. The "dyn" modifier is to mark the text as something that specifies dynamics. Mup only uses it when deciding where to place the text; something marked "dyn" will be treated like crescendo and decrescendo "hairpins." The default *place* for chord and dyn is above, whereas the default for the others is below.

The text strings used with these chord, analysis, or figbass modifiers can contain any characters, but characters that indicate accidentals ("#", "&", "x", "&&", and "n") are translated to the appropriate music character, while "o", "o/", and "^" are translated to "\(dim)", "\(halfdim)", and "\(triangle)" respectively. However, with "chord," the translation of "n" to natural sign is not done, so you have to use \(nat) if you want a natural sign. This is because a literal letter "n" tends to occur more often in chords than natural signs.

If you want to turn off the translation, to treat one of these characters literally, you can precede it with two backslashes. Thus, for example, "\\&" would yield a literal ampersand rather than a flat symbol.

If the transpose or addtranspose parameters are set, chords are transposed to match the new key: the letters "A" through "G" and any following accidentals will be transposed appropriately. The accidental can be either something like "#" or "&" or any of the special music characters for accidentals (\(sharp), \(flat), etc.). If the staff is specified as "all," the score transposition value is used. Transposition has no effect on analysis or figbass.

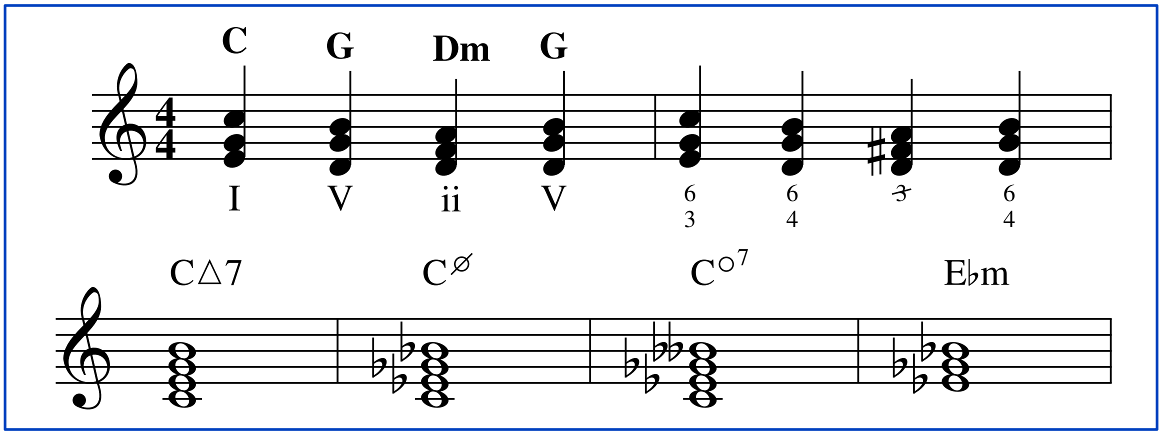
The "chordtranslation" parameter can be used to translate chords to something like DO/RE/MI, or to apply the German usage of H for B, and B for B flat.

In figbass, the string starts out in piled mode, which means that each space in the input string will cause a new line on output, and, unless you specify other alignment, each line will be aligned on the last digit in the line. Also, the meanings of / and \/ are reversed from the normal meaning. This is done since drawing a slash through a number is very common in figured bass, so you can just use a / to indicate this, but if you really want a real slash, you can still get one by entering \/.

With all three special qualifiers, the usual meanings of : and \: are reversed. This is because piling is very common in these strings, so it's handier to just put a : to indicate this, and for those rare cases when you want a literal colon, you can still get one by entering a \:.

Here is an example showing chord, analysis, and figbass:

```
1: egc+;dgb;dfa;dgb;
bold chord above 1: 1 "C"; 2 "G"; 3 "Dm"; 4 "G";
rom analysis below 1: 1 "I"; 2 "V"; 3 "ii"; 4 "V";
bar

1: egc+;dgb;df#a;dgb;
rom figbass below 1: 1 "6 3"; 2 "6 4"; 3 "3/"; 4 "6 4";
bar

1: 1cegb;
rom chord above 1: 1 "C^7";
bar
1: 1ce&g&b&;
rom chord above 1: 1 "Co/";
bar
1: 1ce&g&b&&;
rom chord above 1: 1 "Co:7";
bar
1: 1e&g&b&;
rom chord above 1: 1 "E&m";
bar
```

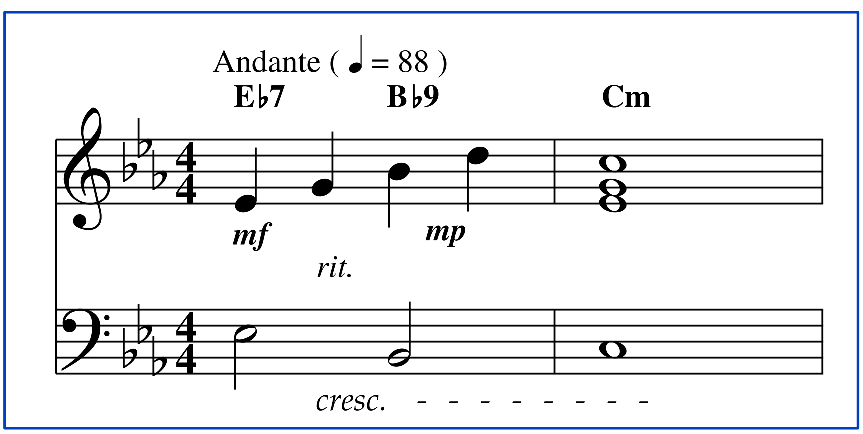


If a music symbol occurs inside an ital, bold, or boldital string, the music symbol will automatically be made to match the rest of the string.

It is possible to supply a duration on text statements. When this is done, Mup will draw a dashed line from the end of the text to the end of the duration. This may be useful if you want to indicate how long an action such as an accelerando or crescendo is to last. If the last character of the string is a "~", a wavy line will be drawn instead of a dashed line. If the last character of the string is an underscore, an underscore line will be drawn.

Here are some examples of text:

```
1: e;g;b;d+;
2: 2e;b-;
rom (12) above 1: 0 "Andante ( \(smup4n) = 88 )";
boldital (12) below 1: 1 "mf"; 3.5 "mp";
newcentury bold (12) chord above 1: 1 "E&7"; 3 "B&9";
ital between 1&2: 2 "rit.";
palatino ital below 2: 2 "cresc." til 1m+2;
bar
1: 1egc+;
2: 1c;
bold (12) chord above 1: 1 "Cm";
bar
```

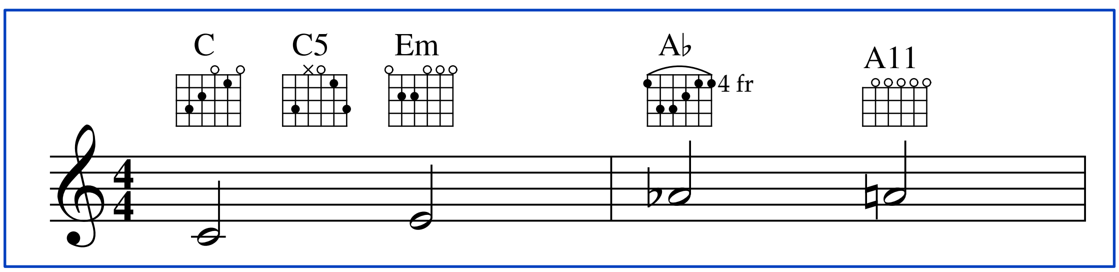


10.3 Grids

If the gridswhereused parameter is set to "y," chords will also have a grid printed. For this to work, a grid with the same name as the chord must be defined in "grids" context elsewhere in the input file.

The grids context contains lines each consisting of two text strings. The first is the name of the chord, matching the name you want to use in printing text with the "chord" qualifier. The second describes the fret pattern for that chord. It is a space-separated list, with each list element being either a fret number from 1 to 99, which will result in a dot being printed at that fret, or an "o" or "x" which will result in a circle or x respectively being printed above the grid, or a "-" in which case no mark at all will be made. The fret pattern may also contain an open and closing parenthesis to mark where a curve is to be drawn, indicating the use of a single finger to span several strings. For example:

```
score
      gridswhereused=y
      gridfret=3
grids
      "C"      " -  3  2  o  1  o "
      "C5"      " -  3  x  o  1  3 "
      "Em"      " o  2  2  o  o  o "
      "A&"      "(4  6  6  5  4  4)"
      "A11"     " -  o  o  o  o  o "
music

rom chord 1: 1 "C"; 2 "C5"; 3 "Em";
1: 2c;e;
bar

rom chord 1: 1 "A&"; 3 "A11";
1: 2a&;an;
bar
```



If transposition is in effect, the chord names in the grids context have to match the transposed names. So, for example, if you use an "A" chord in a chord statement, then transpose the staff up a major second, Mup will look for and use the grid definition called "B" to match the transposed chord name. If chordtranslation is in effect, that is applied to the chords in grid context, though transposition is not.

If you supply more than one grid definition for the same chord name, Mup uses the last one. This allows you to easily "include" a file of standard chords, but override a few of them with a special fingering for a particular song. If you really want to use more than one fingering in different places in the same song for the same chord name, you need to make the names look different, so Mup will treat them as different chords. One way to do this would be to put a space and backspace in the name. So, for example, "Am" and "Am \b" would count as separate chords and could have different grid definitions, but the chord names would still look the same on output.

Sometimes, you might want to have an alternate label printed for a chord at a particular place. An example might be that you have a C chord followed by a C/B chord, and want to abbreviate the second to just /B. Another case might be wanting a chord name in parentheses for some reason, in just one spot. This can be done by giving a string in parentheses, after the string for the chord name:

```
rom chord above 1: 1 "C"; 2 "C/B" ("/B"); 3 "F" ("(F)"); 4 "F/B" ("/B");
```

The first string is the actual chord name, which will be used to look up what grid diagram to print, while the parenthesized string specifies what label to print. This allows the same chord grid to be used with different labels, or the same label to be used for different grids.

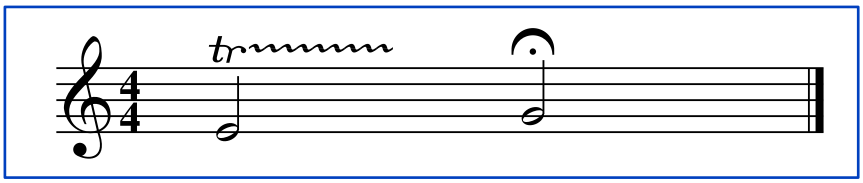
10.4 Mussym

While it is possible to place musical symbols such as fermatas and coda signs using text statements, it is perhaps a bit confusing, since music symbols are really not part of any particular font. So there is a "mussym" statement that can be used. The text strings after the colon must each consist of a single musical symbol whose name can be given without the usual \() wrapper. The following two lines produce identical results, but the second is perhaps a bit clearer:

```
rom above 1: 1 "\(ferm)";
mussym above 1: 1 "ferm";
```

A duration is not allowed on mussym statements except in one special case—if the symbol is "tr" (trill). In that case, the duration tells Mup how long a wavy line to draw from the end of the "tr" symbol.

```
1: 2e;g;
mussym above 1: 1 "tr" til 2; 3 "ferm";
endbar
```



A size can optionally be specified, inside parentheses:

```
mussym (15) above 2: "turn";
```

10.5 Phrase marks

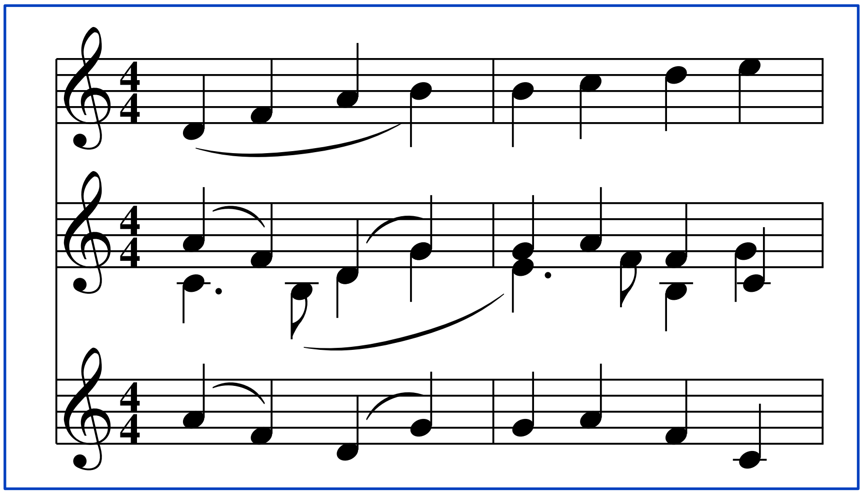
While it may often be more convenient to specify phrase marks using "ph" and "eph" in the music input, you can also use the phrase statement.

If there is only one voice, specifying *place* for a phrase just tells Mup where to draw the phrase mark. If there are two or more voices, and a *place* is specified, "above" indicates to Mup that the phrase is associated with voice 1, and "below" indicates to Mup that the phrase is associated with voice 2.

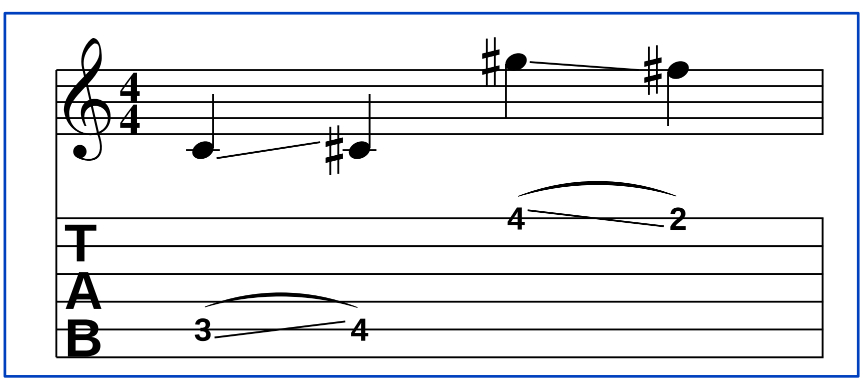
If no *place* is specified and there is only one voice with notes, Mup will decide which side would be better based on the musical data. This means the phrase mark may come out above or below. In the case where there are two voices and there are notes present in both voices, phrase marks will be drawn both above and below.

Each phrase statement item must include a begintime and duration. A phrase mark must begin and end on a chord, so Mup first takes the begintime and duration and finds the chords nearest to each of them. It then draws a phrase mark between them, shaping it to be out of the way of other things as much as possible. It is possible to specify a "grace backup" on the begintime or endtime to make the phrase start or end on a grace note. This is done by giving a negative number in parentheses, specifying how many grace notes to back up. It is possible to "nest" phrase marks (i.e., have one phrase on a subset of the chords of another phrase).

Some examples:

```
staff 2
  vscheme=2o

music

1: d;f;a;b;
2,3 1: a;f;d;g;
2 2: 4.c;8b-;4d;g;
phrase 1: 1 til 4;
phrase above 2,3: 1 til 2; 3 til 4;
phrase below 2: 2.5 til 1m + 1.5;
bar
1: b;c+;d+;e+;
2,3 1: g;a;f;c;
2 2: 4.e;8f;4b-;g;
bar
```

Phrase marks are sometimes used on tablature staffs in conjunction with slides.

```
score staffs=2
staff 2 stafflines=tab
music

2: a3<>;a4;e4<>;e2;
phrase above 2: 1 til 2; 3 til 4;
bar
```

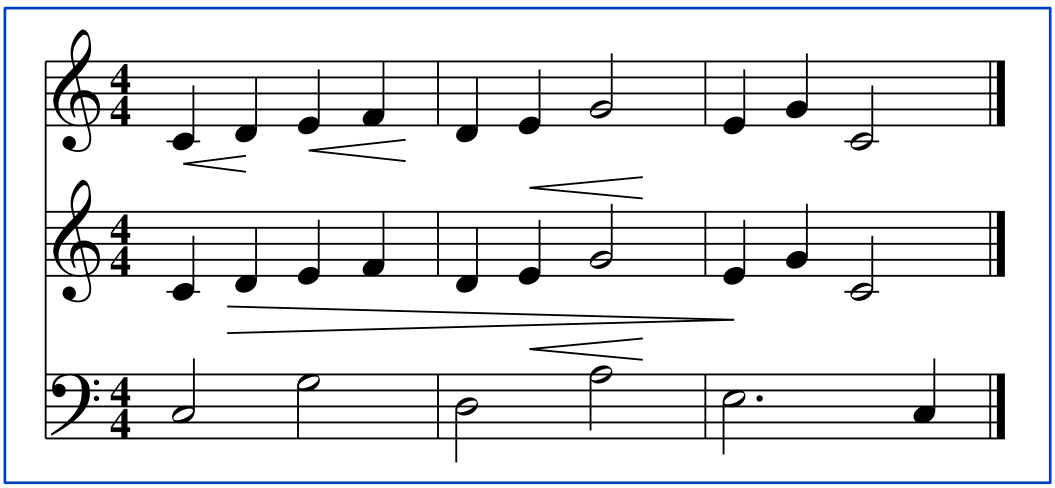


The word "phrase" can be preceded by a line type modifier: dotted or dashed. The dotted or dashed styles might be used for phrase marks that were added by an editor rather than the composer, or to show a phrase that doesn't apply to all verses.

10.6 Crescendo and decrescendo marks

The "<" and ">" statements are used to specify crescendo and decrescendo marks respectively. Each mark must include a begintime and duration. The begintime and duration can include a grace backup specification, to make the mark begin or end on a grace note. Some examples:

```
1-2: c;d;e;f;
< below 1: 1 til 2; 3 til 4.5;
3: 2c;g;
> between 2&3: 1.7 til 2m + 1;
bar
1-2: d;e;2g;
3: 2d;a;
< 2,3: 2 til 3.8;
bar
1-2: e;g;2c;
3: 2.e;4c;
endbar
```

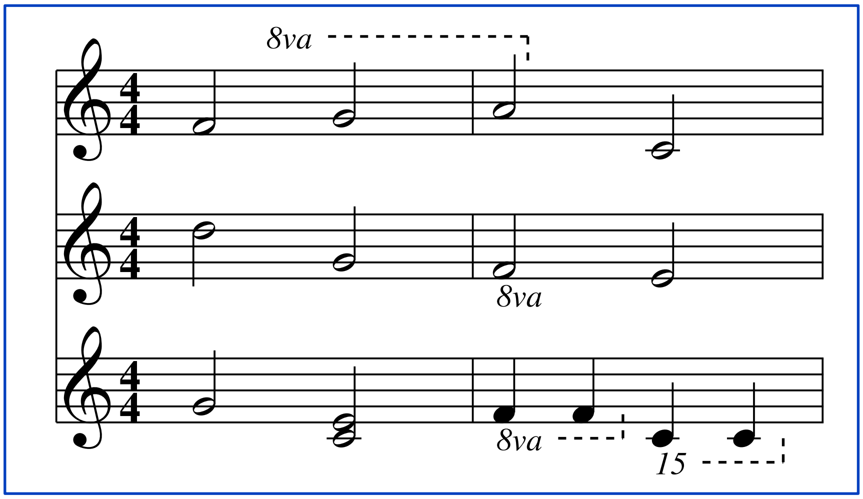


The placement of crescendo and decrescendo marks can be controlled by setting the dyndist parameter.

10.7 Octave marks

Octave statements are used to mark notes that are to be played one or more octaves higher or lower than written. An "octave above" statement is used to specify playing higher than written, or "octave below" for playing lower than written. Each item must include a begintime and a text string. The text string is most typically "8va" although Mup will print whatever you say. For MIDI purposes, up to two digits at the beginning of the string are examined, and a string starting with 8 will be treated as one octave, 15 or 16 will be treated as two octaves (15 is really "correct," but a few publishers may use 16), 22 or 24 as three octaves, etc. It will always default to be printed in 12-point Times ital font, although you can override that using the usual \f and \s conventions. If the octave shifting applies to more than a single chord, there should also be a duration specified, reaching to include the last affected note. In this case, Mup will draw a dashed line to mark the span affected. Note that specifying the exact beat of a chord indicates the horizontal center of the chord, so if you want the dashed line to reach a bit past the note, it will be necessary to specify a duration slightly beyond the point of the last chord to be included. Examples:

```
1: 2f;g;
2: 2d+;g;
3: 2g;ce;
octave above 1: 2 "8va" til 1m + 1.3;
bar
1: 2a;c;
2: 2f;e;
3: 4f;;c;;
octave below 2: 1 "8va";
octave below 3: 1 "8va" til 2.5; 3 "15" til 4.5;
bar
```

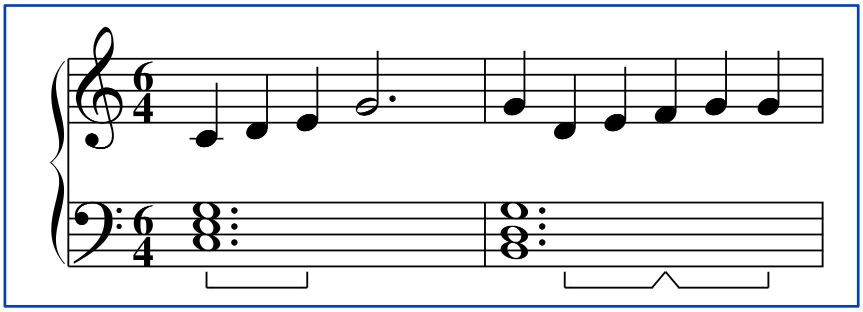


10.8 Piano pedal marks

Piano pedal marks are somewhat different than other statements described in this section. Rather than having begintime, text, and duration, each item is just a time offset value, plus an optional "*". If no pedal mark is currently in progress, the first time offset value indicates where the pedal is depressed. Any subsequent pedal items on that staff will then indicate a "blip"—lifting and then immediately depressing the pedal (which is indicated on the printed music by a "^"), unless there is a "*", in which case it means to lift the pedal and leave it up.

Some examples may help:

```
1: c;d;e;2.g;
2: 1.ceg;
// depress pedal on beat 1, release on 3
pedal 2: 1; 3*;
bar

1: g;d;e;f;g;;
2: 1.gdb-;
// depress pedal on 2, release and depress
// on 4, release on 6
pedal below 2: 2; 4; 6*;
bar
```



See also the "pedstyle" parameter.

10.9 Rolls

Rolls can be specified with the "roll" statement, which has a format:

roll *staff voice* **:** *timeval***;**

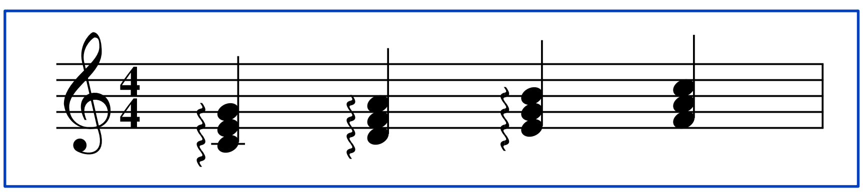
A simple example would be:

```
roll 2 1: 3;
```

which indicates that a roll is to be placed on the chord at count 3 of staff 2 voice 1. As usual, if the *voice* is omitted, voice 1 is assumed.

Multiple rolls in a measure can be listed on a single statement if they are associated with the same voice. For example:

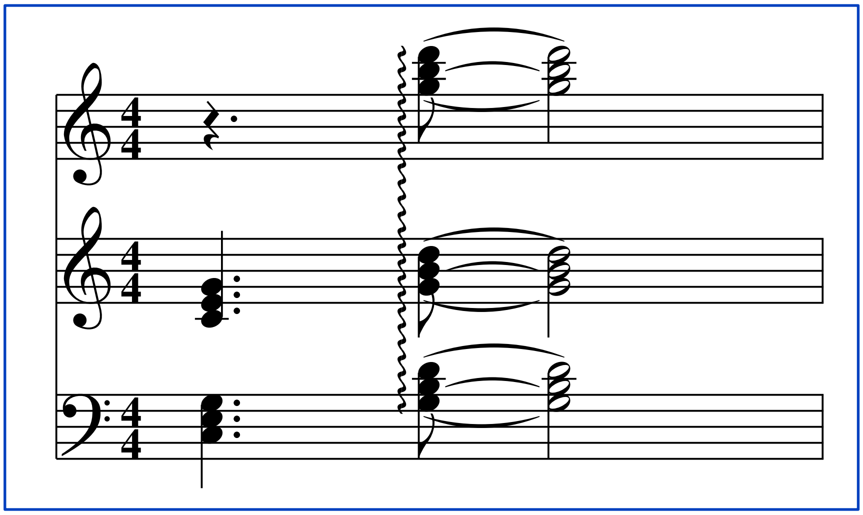
```
1: ceg;dfa;egb;fac+;
// rolls on 3 chords: on the first,
// second, and third beats of the measure
roll 1: 1;2;3;
bar
```



would produce rolls on the chords on counts 1, 2, and 3.

A roll can extend over several chords on different voices, or even different staffs. This is specified by giving the top and bottom staffs and voices, with the keyword "to" between them. For example:

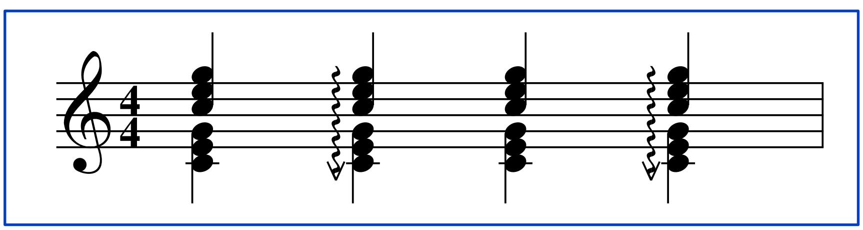
```
1: 4.r;8g+b+d++ tie;2;
2-3: 4.ceg;8gbd+ tie;2;
roll 1 1 to 3 1: 2.5;
bar
```



The roll would extend from voice 1 of staff 1 to voice 1 of staff 3.

If you wish the roll to be downward, the keyword "down" can be placed after "roll." This will result in a downward arrow being drawn on the bottom end of the roll.

```
score
  vscheme=2o
music
1 1: c+e+g+;;;;
1 2: ceg;;;;
// downward roll on staff 1
// on beats 2 and 4
roll down 1 1 to 1 2: 2; 4;
bar
```



You can also explicitly say "up" to cause an upward arrow to be drawn at the top of the roll. If no direction is specified, no arrow is drawn.

11. TAGS, PRINTING, LINES, AND CURVES

11.1 Location tags

The concept of "location tags" has been mentioned several times. We now describe this facility in more detail.

A location tag is associated with a note, chord, lyric syllable or barline. It can then be referenced in order to place a second object, like a comment or dotted line, relative to the first object.

A tag name can be either a single lowercase letter, or an underscore folowed by one or more letters, digits, or underscores. Each location tag is really a collection of six values, namely the north, south, east, west, x, and y values of the tagged object. These are referenced by giving the tag name followed by a dot, followed by the letter n, s, e, w, x, or y. For example:

```
c.n // north of tag c
_xyz.e  // east of tag _xyz
x.x // x coordinate of tag x
```

The n, s, e, and w values describe the smallest rectangle that will completely enclose the object being tagged. The north refers to the top of the object, the south to the bottom, the west to the left edge, and the east the right edge. The x and y values correspond to the "center" coordinate of the object. This is not necessarily the geometric center, but more of a "logical center." In the case of a tag associated with an individual note, it is the geometric center of the notehead. However, on tags associated with a chord, the x is at the center line of noteheads that are on the "normal" side of the stem. (Normally, when a stem is up, notes are put on the left side of the stem, and when the stem is down they are placed on the right side. However, when two notes adjacent on the staff have to be printed in a single chord, one has to be moved to the opposite side.) The y of a chord is always the middle line of the staff containing the chord. The x of a bar line is its geometric center. The y of a bar line is the center line of the top visible staff. For lyrics, the n, s, e, and w give the boundaries of the smallest box that encloses the syllable, including anything inside <^ >, but excluding anything inside < >, while the x and y are the center of that box.

Tag names can be reused. The value of a tag will always be its most recent definition.

There are also several "pre-defined" tags. They are:

_page	the entire page
_win	the available part of the page
_cur	the current location
_score	the current score
_staff.*N*	staff *N* of the current score

The _page tag refers to the entire page. The x and y values of _page are at the geometric center of the page. This tag is seldom useful, but is provided in case you want to force something to a specific place on a page.

The _win tag refers to the area of the page available for printing. The name comes from the idea that it provides a view, as if through a window, of a portion of the page. Its precise meaning depends upon the current context. In header, footer, top, bottom, header2, footer2, top2, bottom2, and block contexts, it refers to the area of the page taken up by the corresponding element. In music context, it refers to the area of the page that remains after excluding the margins and the header, footer, top and bottom for the page. There is no way to access location tags in any other contexts, so _win (and all other location tags) are meaningless in other contexts.

The _cur tag refers to the current location on the page. This is a single point, such that east and west values are the same as x, and north and south values are the same as y. It is only useful immediately after a command that explicitly sets the current location, that is, after a printing command or line or curve command. Since Mup places musical data in an arbitrary order, trying to use _cur at other times is likely to cause output at a seemingly random place.

The _score built-in tag refers to the current or most recent score. _score.x will be at the position of the left edge of the staffs. _score.y will be at the middle line of the top visible staff. _score.w will be the left margin. _score.e will be the right margin. _score.n will be the farthest upward that anything associated with the score protrudes. _score.s will be the farthest downward that anything associated with the score protrudes.

There are also built-in tags for each staff in the current or most recent score. Since there is one per staff, there is a special notation: _staff followed by a dot and the staff number. Only staff numbers that exist (i.e., are between 1 and the value of the staffs parameter) and that are currently visible can be referenced. _staff.2.x will be the place on staff 2 between where the clef and time signature end and the first measure of the score begins. (Actually all staffs have the same x value.) _staff.5.y will be the middle line of the staff 5. _staff.7.w will the left edge of the staff label if it has a staff label; otherwise the left edge of the staff (the same as _score.x). _staff.10.e will be the right margin of staff 10 (or really any staff, since they will all be the same). _staff.4.n will be the farthest upward protrusion of anything associated with staff 4. _staff.15.s will be the farthest downward protrusion of anything associated with staff 15.

Location tags can be referenced by a number of commands, including those for printing text, or drawing lines or curves, or in user-defined PostScript code. In the simplest case, two tag references are given in parentheses. The first tag will refer to a horizontal direction (the "x" direction for mathematicians), namely a tag with a w, e, or x after the dot. The second tag is then a vertical tag, having n, s, or y. For example:

```
(g.x, g.y)   // x and y of tag "g"
(_tag.w, _item.n)   // west of tag "_tag" and
                    // north of tag "_item"
```

The first example refers to the x,y coordinate of tag "g". The second item refers to two different tags. The point referenced has its horizontal location aligned with the west side of the object having the tag "_tag," while its vertical position is in line with the north side of the object having the tag "_item". In other words, if a line were drawn along the west side of the object tagged with "_tag" and another line were drawn along the top edge of the object with tag "_item", the point where those two lines crossed would be the point referenced.

Frequently, you may want to place something relative to a tagged object. You can add offsets in both the x and y dimensions. These offsets are given in stepsizes. As was mentioned earlier, a stepsize is half the distance between two staff lines.

```
(g.x + 4.5, g.y + 4.5)
```

refers to the point 4.5 stepsizes to the right and above the point (g.x, g.y). Adding an offset moves to the right in the horizontal direction and upward in the vertical direction. You can also subtract an offset:

```
(k.w - 1.3, m.n - 2)
```

refers to a point 1.3 stepsizes to the left of the west edge of k and 2 stepsizes below the top of m.

In the horizontal direction, offsets can be given in terms of "time". This is specified by using the keyword "time" followed by a number of beats. As an example, suppose we have the tag reference:

```
(q.x + time 1, q.y + 2)
```

If the time signature is 4/4 and tag "q" happens to be associated with a half note, this tag reference would refer to a point halfway between that note and the following chord, since it includes an offset of a quarter note. If "q" had been associated with a whole note, the point would be only 1/4 of the way between the note and the following chord. Since notes are placed based on various constraints and aesthetic considerations, the actual distance will vary depending on which tag is referenced.

In the case of a tag associated with a bar line, the time to distance mapping is done based on the distance between the bar (at count "0") and the first chord in the measure (at count "1"). So, for example, in 4/4 time, a reference to a bar line tag + time 0.5 would indicate a place halfway between the bar line and the first chord, whereas tag + time 0.25 would be one fourth of the way.

The mapping of time to distance is done based on the note, rest, lyric syllable, or bar line associated with the most recent horizontal tag in the expression. For example, if the most recent horizontal tag is associated with a quarter note, and the distance between that note and the next was 0.5 inches, specifying "+ time 2" (a half note) would mean 1.0 inch to the right, or specifying "- time 2" would mean 1.0 inch to the left, even if the notes to the left or right happened to be spread somewhat differently than 0.5 inches per quarter note. Thus it is usually advisable not to specify a time offset greater than the time value of the note or rest associated with the tag, nor to subtract a time value. Stated another way, when specifying the X portion of a coordinate using a time offset, it is best to use a location tag based on whichever note, rest, or bar is immediately to the left of the X location you are trying to specify.

If the first horizontal tag in a horizontal expression is associated with a bar, and that bar happens to be at the end of a score other than the final score, and the result of evaluating the expression is a location that would be out in the right margin or off the right edge of the page, it will be moved to act as if the bar was at the beginning of the following score. If that adjustment is not desired, make sure the location specified is slightly before the bar line.

The various kinds of offsets can be combined.

```
( _pp + 1.2 - time 3.5, _zz + 2)
```

If no tag is specified, the reference becomes an absolute reference, giving an exact page location. For example:

```
(10, 20)
```

is 10 stepsizes from the left edge and 20 stepsizes above the bottom of the page.

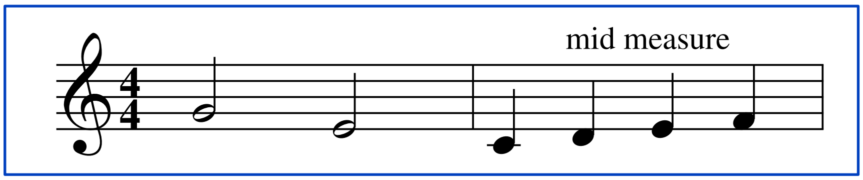
While usually the horizontal and vertical specifications will each be just a tag and direction, possibly plus or minus some offset, they can be more complicated arithmetic expressions. The expressions can involve * for multiplication, / for division, or % for modulo. These operators have higher precedence than addition or subtraction, but you can use parentheses to force different precendence. You can also use + or - as a unary operator, which has the highest precedence of all. The unary + is rarely useful, but unary - is used if you want to have a negative number. There are also several functions available that you can use, namely:

Name	Description	Parameters
sqrt	square root	1
sin	sine	1
cos	cosine	1
tan	tangent	1
asin	arc sine	1
acos	arc cosine	1
atan	arc tangent	1
atan2	arc tangent y/x	2
hypot	hypotenuse	2

A function name must be followed by parentheses containing its parameter(s). For functions with two parameters, the parameters are separated by a comma. Function parameters can themselves be arithmetic expressions. The sin, cos, and tan function parameters are expected to be in degrees. The return values of the asin, acos, atan, and atan2 functions are in degrees. For the most part, expressions are allowed to include operators and functions in any order. This makes it possible to do things like calculate a horizontal location based on some vertical attribute, if you really want to do that. But it also means Mup will not stop you from creating an expression that may not make any logical sense. Mup will do its best to try to do whatever you say, even if that wasn't what you meant.

A common use for an expression more complicated than just a simple tag reference and offset would be to place something in the middle of a measure. You could set location tags on the bars at either end of the measure, and then use the average to get the midpoint of the measure.

```
1: 2g;e;
// Note that be able to set a tag on the left of the very first measure,
// we could make an invisible bar, just to set a tag on it. But here,
// we can use the actual bar line at the left of the measure of interest.
bar =_firstbar

1: c;d;e;f;
bar =_secondbar

// Print a centered string at the average of the X values of
// the bars on either side of the measure,
// 6 stepsizes above the middle of the top staff of the score.
center ((_firstbar.x + _secondbar.x) / 2, _firstbar.y + 6) "mid measure";
```



A much more complicated example is presented later, in the section on macros, demonstrating use of the trigonometric functions.

11.2 Printing commands

There are several commands for printing text. There are four commands that have similar formats, differing only in how they justify text.

> **print** *location* "*text*"
> **left** *location* "*text*"
> **right** *location* "*text*"
> **center** *location* "*text*"

The *location* is optional. If the *location* is omitted, the "print" command will cause the *text* to be printed beginning at the current horizontal and vertical location. The other commands will cause the text to be placed vertically at the current vertical position, but left justified, right justified, or centered within the current margins of the page.

The *location* can be specified using the special keyword of "nl" which means "next line." This moves the current location to the beginning of the following line before placing the text. In other words, the current vertical position is moved downward by the height of the current text point size (or by as much as necessary if the string is taller than that). Then the *text* is printed on that line with the given justification style.

The other way to specify a *location* is by using coordinates. The justification then takes place relative to the referenced location. For example, consider the following "right" command containing a reference to an absolute location:

```
right (25, 4) "something"
```

The y location given is 4. Since this is an absolute location with no location tags being referenced, this means the vertical position will be 4 stepsizes from the bottom of the page. The x location given is 25. Again, this is an absolute location, so the current horizontal position will be 25 stepsizes from the left edge of the page. Since right justification is indicated, the word "something" will be placed such that the right edge of the final "g" will be 25 stepsizes from the left edge of the page. If "center" had been specified, the middle of the word "something" would be at the 25 stepsize point.

Here are some other examples:

```
print "Author unknown"
center nl "subtitle"
left (h.x - 1.5, h.n + 2.3) "Ad lib"
right (_fine.e + time 3, _note.n + 5) "Duet"
```

Another type of print command is "title." The full format of this command is:

mirrored title *fontfamily font size "text1" "text2" "text3"*

However, only the word "title" and one quoted text string are required. The *fontfamily*, if specified, has one of the values valid for the "fontfamily" parameter (avantgarde, bookman, courier, helvetica, newcentury, palatino, or times). The default is the value of the "fontfamily" parameter. The *font*, if specified, has one of the values valid for the "font" parameter (rom, bold, ital, or boldital). If no *font* is specified, the default is the value of the "font" parameter. The optional *size* is a point size within parentheses. If not specified, the default is the value of the "size" parameter.

In all cases, the location will be like that obtained via the "nl" location to a print command. In other words, the title text string(s) will be printed on the line below the location that was current when the title command was encountered. If there is only one string given, it will be centered between the margins. If two strings are given, both will be printed on the same line, but the first will be left justified and the second will be right justified. If three strings are given, they will all be printed on the same line, with the first left justified, the second centered, and the last right justified.

Adding "mirrored" at the beginning only has effect when there are at least two strings provided and the output is going onto a left page, in which case the first and last strings are interchanged.

Some samples:

```
title bold (12) "Sonata 12"
title (18) "Song Without Words"
title ital (12) "Text: John Doe" "Tune: Jane Doe"
title "Suite in C" "Trumpet I" "Waltz"
title    ""    "A. Composer"
mirrored title "at inner margin" "at outer margin"
```

The final command for printing text is the "paragraph" command. This is used when you have a long section of text, and you would like it to automatically wrap around onto as many lines as necessary. You can specify whether you want the right margin to be "justified" or "ragged." If you don't specify, the type of the previous paragraph is used. The default for the very first paragraph is to be justified. By default, the values of the fontfamily, font, and size parameters are used to determine the text style, but any or all of those parameters can be overridden on the paragraph command. The complete syntax is:

justify_type **paragraph** *fontfamily font* **(***size***)** **"***string***"**

Only the keyword "paragraph" and the string are required. Here are some example paragraphs:

```
paragraph "This is an example of a paragraph. Since no justification
type was specified, and this is the very first paragraph, the default
(justified) is used. A paragraph will wrap around
to as many lines as needed.
An explicit newline is given at the end of this paragraph,
to force a blank line between it and the following paragraph.\n"

ragged paragraph avantgarde ital (15) "This paragraph
is ragged rather than flush right. It is in a different font and size.
Ragged paragraphs are split onto multiple lines if necessary,
but they are not spread out to make lines go all the way to
the right margin."

justified paragraph (14) "     Here is another paragraph.
This one is justified.
Only the size was specified for this paragraph; the font was not,
so the current default will be used.
Some spaces are included at the beginning of the paragraph text,
to create an indented first line for the paragraph."

paragraph "Here is the final paragraph.
Since no justification type was given,
that of the previous paragraph (justified in this case) was used.
A paragraph may be used for many things,
such as describing how you want a piece to be performed,
or a biography of the composer."
```

This is an example of a paragraph. Since no justification type was specified, and this is the very first paragraph, the default (justified) is used. A paragraph will wrap around to as many lines as needed. An explicit newline is given at the end of this paragraph, to force a blank line between it and the following paragraph.

This paragraph is ragged rather than flush right. It is in a different font and size. Ragged paragraphs are split onto multiple lines if necessary, but they are not spread out to make lines go all the way to the right margin.

Here is another paragraph. This one is justified. Only the size was specified for this paragraph; the font was not, so the current default will be used. Some spaces are included at the beginning of the paragraph text, to create an indented first line for the paragraph.

Here is the final paragraph. Since no justification type was given, that of the previous paragraph (justified in this case) was used. A paragraph may be used for many things, such as describing how you want a piece to be performed, or a biography of the composer.

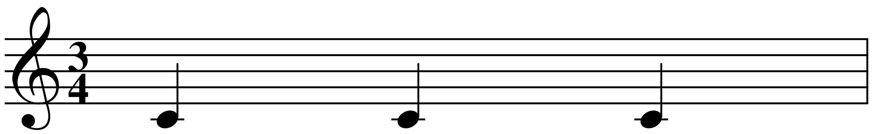
Sometimes you may want to mix blocks of text with music. This can be done by specifying a "block" context. The block context will typically contain one or more "paragraph" commands, although any of the printing commands (paragraph, print, left, center, right, or title) can be used. The block can also contain changes in certain parameters, namely font, size, and fontfamily, which will affect the appearance of the following text. A block can also contain "newscore" or "newpage" commands. Using "newscore" will cause vertical space to be added, as would be used to separate scores. The amount of space is affected by the scoresep and scorepad parameters. Using "newpage" will cause a new page to be started. If a newscore or newpage includes a "leftmargin" specification, that will alter the left margin on the block text that follows. A "rightmargin" specification will alter the right margin of the block text that precedes it.

Here is an example of a block.

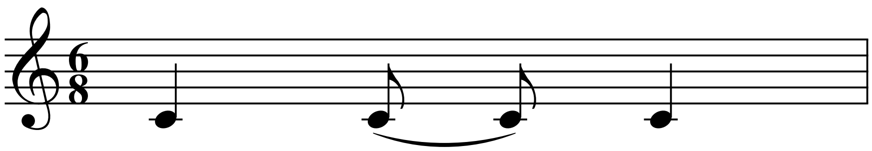
```
block
title bold "Notation in Simple and Compound Meters"
title ""
paragraph "It is common for a person familiar
with mathematics but not with music to assume
that 3/4 and 6/8 time are equivalent,
but that is not the case.
In 3/4 time (which is known as simple triple meter),
a measure containing 3 quarter notes
would be notated like this:"
score time=3/4
music
1: c;;;
bar hidechanges
block
paragraph "whereas in 6/8 time
(which is compound duple meter),
a measure with 3 quarter notes should be notated thus:"
score time=6/8
music
1: 4c;8~;;4;
bar
```

Notation in Simple and Compound Meters

It is common for a person familiar with mathematics but not with music to assume that 3/4 and 6/8 time are equivalent, but that is not the case. In 3/4 time (which is known as simple triple meter), a measure containing 3 quarter notes would be notated like this:



whereas in 6/8 time (which is compound duple meter), a measure with 3 quarter notes should be notated thus:



The printing commands may occur in the header, footer, header2, footer2, top, bottom, top2, bottom2, block, and music contexts. After each printing command, the current location is set to the right edge of the last character printed horizontally and at the baseline of the current line vertically.

11.3 Including raw PostScript

There is another command that looks a lot like the printing commands, but gives you a way to insert raw PostScript into the Mup output. This might be used, for example, to include a picture or logo along with your music. The syntax is

> **postscript** *optional_location optional_exports* **"***raw PostScript***"**

As with the other print commands, if the location is omitted, the current location is used.

Instead of specifying an (x, y) coordinate location, you can declare various PostScript "hooks" that will be called at specific times:

afterprolog — The PostScript code you specify will be placed in the generated Mup output right after the %EndProlog, and before the first page. The current location will be the upper left corner of the page. A typical usage might be to set up some global things that you would then use later in other hooks. Or it could be used to override things defined in the Mup prolog.

beforetrailer — The PostScript code you specify will be placed in the generated Mup output right after the final page, and before the %Trailer. The current location will be the lower right corner of the page. One possible use might be to write out some data gathered to some other file.

atpagebegin — The PostScript code you specify will be placed in the generated Mup output on every page, before anything is written to the page. The current location will be the upper left corner of the page. Some possible uses would be to paint the page some background color or add a fancy border. Or it could be used to add a red "Do not copy" watermark, where the music output would still be completely readable, since it would get drawn on top of the mark.

atpageend The PostScript code you specify will be placed in the generated Mup output on every page, after everything else has been written to the page. The current location will be the lower right corner of the page. An example of a possible use would be to write "SAMPLE" in huge letters across each page. to leave enough music readable that a potential user can get an idea of what the music is like, but overwrite enough that it isn't really usable, to encourage paying for a copy without the mark.

atscorebegin The PostScript code you specify will be placed in the generated Mup output just before the output for each score. The current location will be the upper left corner of the score. A possible usage would be to put a different color background behind each score, so they would really stand out even on a very crowded page. Another would be to put yellow highlighting behind a particular staff on each score.

atscoreend The PostScript code you specify will be placed in the generated Mup output just after the output for each score. The current location will be the lower right corner of the score. One possible use would be to "white-out" something Mup printed that you don't like, but have no other way to prevent. With enough work, you could probably even create an ossia.

It should be noted that since a postscript command can contain arbitrary PostScript code that is merely passed through by Mup, any tools that try to transform Mup input to some other music notation format (e.g., Music XML) will most likely be unable to understand any of that PostScript code, and any information in it will almost certainly be completely lost during the transformation. So while it is provided as an "escape hatch" to allow you to do things Mup does not support natively, its usage is discouraged, unless you really need to do something that Mup does not support directly, and if you are willing to accept the fact that doing so will compromise portability.

The *optional_exports* let you specify a list of Mup location tags whose values you want to pass to your PostScript code. If you list a tag by itself, all six values (x, y, n, s, e, w) are passed to Postscript, or you can list just a specific direction:

```
postscript atscorebegin with _score, _mytag.y, "...PostScript... "
```

As a special case, you can specify _staff, which will pass all six values for all visible staffs. The PostScript name will be the Mup name with Mup prepended to it. So _tag.x in the Mup input will become Mup_tag.x in the PostScript output. Often it is useful to use a more generic name in your Postscript code, and then pass the values from different tags to it at different times. You can do this by specifying an alias tag.

```
postscript atpagebegin with _value = _staff.2 "...PostScript... "
```

In that example, _value will be an alias, and the PostScript name will be Mup_value, but its value will be that of _staff.2. So suppose the value of _staff.2.x is 3.8. If you say:

with _staff.2.x

the output would be:

/Mup_staff.2.x 3.8 def

but if you say:

with _value.x = _staff.2.x

the output would be:

/Mup_value.x 3.8 def

and you can later use the exact same PostScript code (that uses Mup_value.x), but pass it the value from some other tag, like _staff.5.x or your own tag. Note that you don't have to use the same direction for both, so you could say

with _value.x = _another_tag.w

The string containing raw Postscript is copied directly to the Mup output. You can use the usual \" to embed a double quote in the string, but otherwise the string is copied exactly as it is, enclosed inside a PostScript save/restore. After the restore, the current location is reset back to where it was originally.

Alternately, you can use the keyword "file" and then give a file name as the string. The contents of the file are copied to the output verbatim, so it is not necessary to use backslashes before double quotes to preserve them.

Since the PostScript code is copied without any interpretation, Mup does not reserve any space on the page for it, and it is your responsibility to provide valid PostScript. The PostScript language is beyond the scope of this User's Guide; consult a book on PostScript if you need more information.

As a simple example, you might include an Encapsulated PostScript file near the lower left corner of the current page, using:

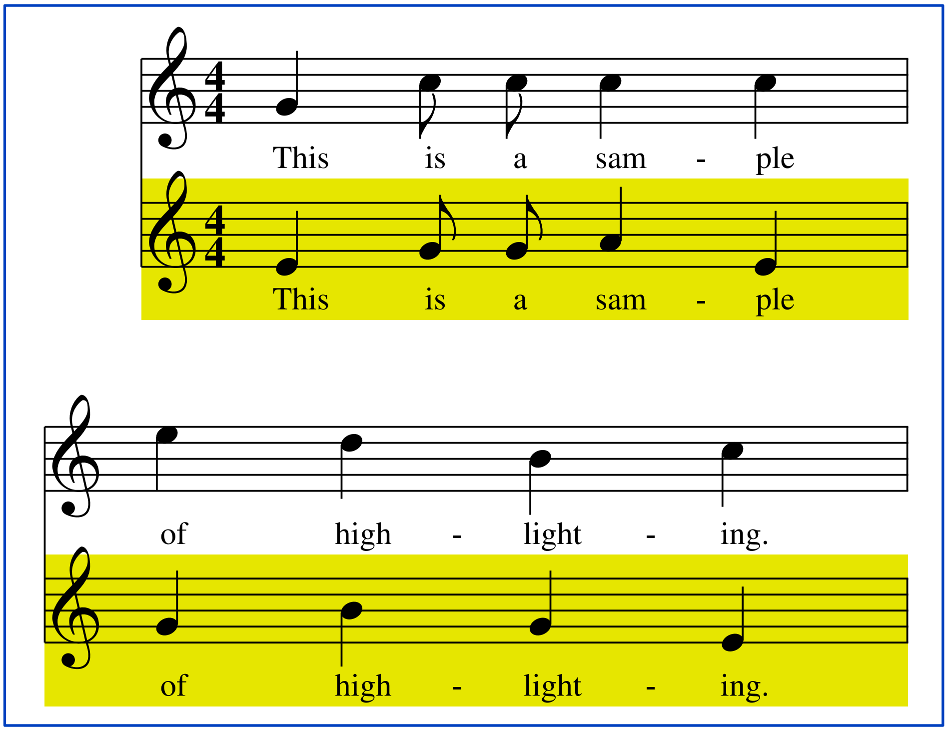
```
postscript "50 50 translate (file.eps) run"
```

Or you could print a message in red italics near the bottom of the page, like this:

```
postscript (70, 20) "
              1 0 0 setrgbcolor
              /NewCenturySchlbk-Italic findfont
              16 scalefont setfont
              (Photocopying prohibited) show
              "
```

Next we show an example of a hook that allows you to do the equivalent of a yellow highlighting pen on a particular staff.

```
score
    staffs=2
music

 define SOPRANO 1 @
 define ALTO 2 @

ifdef HIGHLIGHT
// If HIGHLIGHT is set to a staff number,
// highlight that staff by drawing a yellow box behind it.
postscript atscorebegin with _box = _staff.HIGHLIGHT  "
    newpath
    Mup_box.w Mup_box.n moveto
    Mup_box.e Mup_box.n lineto
    Mup_box.e Mup_box.s lineto
    Mup_box.w Mup_box.s lineto
    closepath 0.9 0.9 0.0 setrgbcolor fill stroke"
endif

SOPRANO: g;8c+;;4;;
ALTO: e;8g;;4a;e;
lyrics below SOPRANO,ALTO: "This is a sam-ple";
bar

newscore

SOPRANO: e+;d+;b;c+;
ALTO: g;b;g;e;
lyrics below SOPRANO,ALTO: "of high-light-ing.";
bar
```



If you invoke Mup with -DHIGHLIGHT=SOPRANO, the top score will be highlighted. If you invoke with -DHIGHLIGHT=ALTO (as shown), the second will be highlighted. Otherwise nothing will be.

11.4 Lines and curves

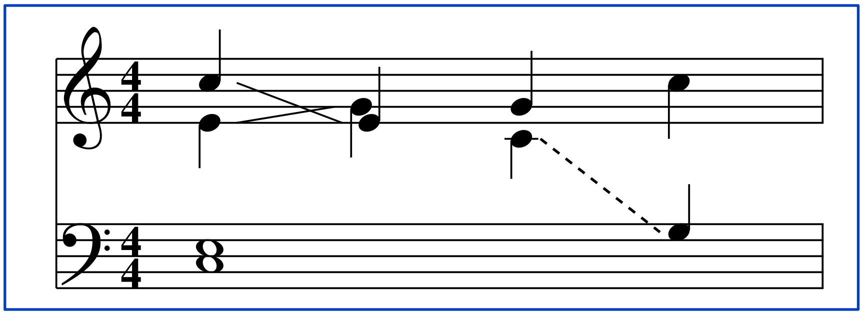
The "line" statement is used for drawing lines. Its format is:

linetype **line** (*X1, Y1*) **to** (*X2, Y2*)

The *linetype* can be "wide," "medium," "wavy," "dotted," "dashed," or it can be omitted, which means narrow. The wavy line could be used for glissandos. It could also be used for manually drawing rolls and trills, although it should rarely, if ever, be necessary to resort to this, since the roll and mussym statements will usually suffice and are much easier to use. Narrow, dashed, or dotted lines would typically be used for voice crossings. The medium line is about twice as wide as a narrow line, and a wide line is about twice as wide as a medium line.

The x and y coordinates are specified as was described in the section on "Location tags."

```
score
  vscheme=2f
  staff2 clef=bass

music
1 1: c+ =c;e =e;g;c+;
1 2: e =_e;g =_g;c =_c;s;
2 1: 2.s;4g =_gg;
2 2: 1ce;
line (c.e + 2, c.y) to (e.w - 2, e.y)
line (_e.e + 2, _e.y) to (_g.w - 2, _g.y)
dashed line (_c.e + 1, _c.y) to (_gg.w - 1, _gg.y)
bar
```

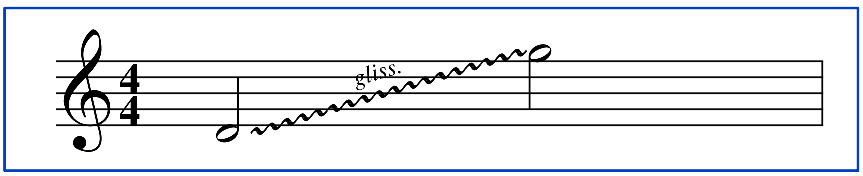


It is possible to print a text string by the line by adding

with *fontfamily font* (*size*) **"***string***"** *side*

at the end of the "line" statement. Only the keyword "with" and the text string itself are required; the font, size, and side information is optional. As usual, the fontfamily, font, and size parameter values are used to get values if the font or size are omitted. The side defaults to "above" but can be set to "below." The most common usage for printing a string with a line is probably for glissandos.

```
1: 2d =n; g+ =m;
wavy line (n.e + 1.5, n.y) to (m.w - 1.5, m.y) with ital (9) "gliss."
bar
```

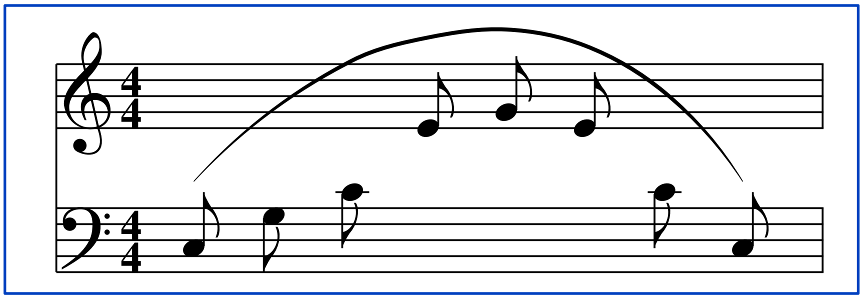


Arbitrary curves can be drawn using the "curve" statement:

linetype **curve** (*X1,Y1*) **to** (*X2,Y2*) **to** (*X3,Y3*) ...

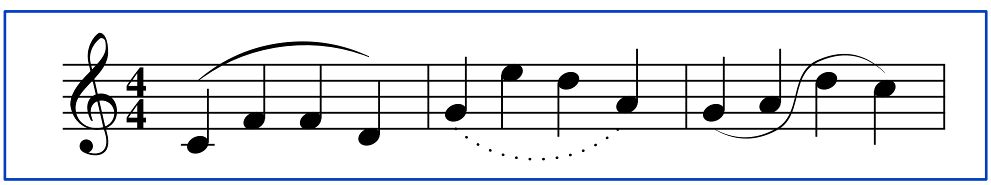
When using this form of the "curve" statement, at least three coordinates must be specified; more are permitted. The *linetype* can be "medium," "wide," "dashed," or "dotted," or omitted. A curve will be drawn through the specified points in the specified order.

```
score
staffs=2

staff 2
  clef=bass

music
1: 4.s;[=c] 8e; [=d] g; [=e] e;4s;
2: [=a] 8c;g; [=b] c+;4.s; [=f] 8c+; [=g] c;
medium curve (a.x, a.n+1) to (b.x, b.n+15) to (c.x, c.n+4) \
        to (d.x, d.n+3) to (e.x, e.n+3) \
        to (f.x, f.n+11) to (g.x, g.n+1)
bar
```



An alternative way to specify a curve is to only specify the two endpoints, along one or more "bulge" values, given in stepsizes. Suppose a single bulge value *n* is given. To find the midpoint of the resulting curve, imagine a line is drawn between the two endpoints, then from the midpoint of that line, move perpendicular to the line for *n* stepsizes. A positive *n* will cause the curve to bulge to the left as you move from the start point to the end point, while a negative *n* will make it bulge to the right. If two bulge values are given, bulges are calculated from the 1/3 and 2/3 point of the line between the endpoints; with three points, at 1/4, 1/2, and 3/4, and so forth. Multiple bulge values are separated by commas.

```
1: c =_c1; f; f; d =_d1;
medium curve (_c1.x, _c1.y + 8) to (_d1.x, _d1.y + 10) bulge 3
bar

1: g =_g1; e+; d+; a =_a1;
dotted curve (_g1.x, _g1.y - 2) to (_a1.x, _a1.y - 2) bulge -4.3
bar

1: g =_g2; a; d+; c+ =_c2;
curve  (_g2.x, _g2.y - 2) to (_c2.x, _c2.y + 2) bulge -2.5, 4
bar
```



12. MISCELLANEOUS FEATURES

12.1 Newscore and newpage, samescore and samepage

Normally, Mup determines how many measures to put on each score and how many scores to put on each page. You can force Mup to move to the next score with the "newscore" statement, or to the next page with a "newpage" statement. Here is an example:

```
1: c;d;e;f;
bar
1: e;f;g;;
bar
newscore   // go to next score
1: 2c;;
bar
1: e;f;g;;
bar
newpage   // go to next page
1: e;g;2c;
bar
```

You can temporarily change the indent for the new score by specifying "leftmargin=*num*," where *num* is a number of inches or centimeters (depending on the current setting of the "units" parameter.) The number may include a decimal fraction part (e.g., 8.5). The number can optionally be preceded by a plus or minus sign, in which case the number is taken as an amount to add or subtract from the normal margin, rather than as the actual margin value. The new score will then be indented by that much, overriding the value of the leftmargin parameter.

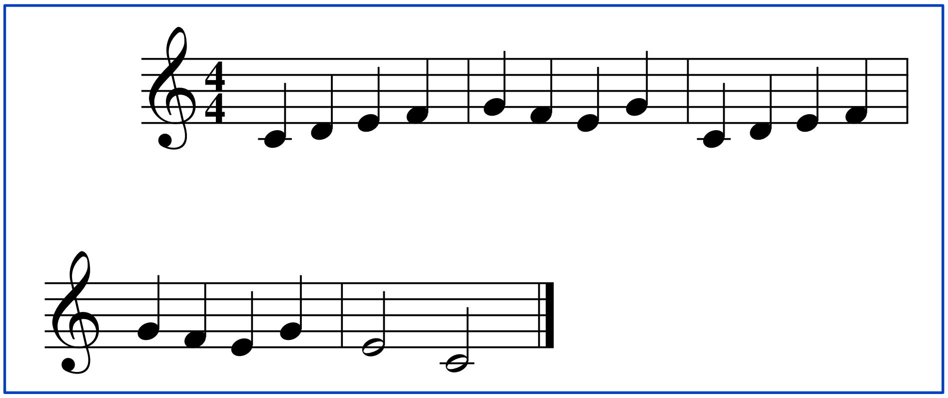
In a similar way, you can specify "rightmargin=*num*," which will affect the right margin on the **previous** score. This might be used, for example, if you want a piece to end with a shorter than normal score.

```
newscore leftmargin=1.2 rightmargin=2.7
```

The equals sign is optional in these margin overrides.

For rightmargin, you can specify rightmargin=auto rather than giving a number. In that case, Mup will calculate the appropriate value to use to make the music on the previous score spaced based on the packexp and packfact parameters, without spreading to fill the line.

```
1: c;d;e;f;
bar

1: g;f;e;g;
bar

1: c;d;e;f;
bar

1: g;f;e;g;
bar

1: 2e;c;
endbar

newscore rightmargin=auto
```



On a "newscore" line you can also specify the separation between the preceding and following scores. The separation is the distance between the bottom line of the bottom visible staff of the preceding score, and the top line of the top visible staff of the following score. It is measured in step sizes. An example is:

```
newscore scoresep = 10.5
```

This distance will be enforced, regardless of the values of the scorepad and scoresep parameters. The distance can even be negative, which will force them to overlap. If there is not room on the page for both scores with the stated distance between them, the second score will be put on the next page instead.

This scoresep option also works if the preceding and/or following item is a block instead of a score. The distance is then measured from the boundaries of the block(s).

Also, a newscore can be put in the middle of a block, which effectively breaks the block into two blocks, and the scoresep option can be used there. This can be especially useful with a big block. Suppose you would normally like it to appear as one big block, but if the page is mostly full of music already, and the whole block doesn't fit, you may prefer to break it in half, so that the first half will still go on the first page in this case, rather than forcing it all to the next page. To do this, at the place you would like to allow the break to occur, put this line:

```
newscore scoresep = 0
```

Sometimes it may be useful to tell Mup where you do *not* want a new score to begin. This is done by:

```
samescorebegin
// ... two or more measures of Mup input
samescoreend
```

Similarly, to specify that certain measures and/or blocks should all be kept on the same page:

```
samepagebegin
// ... two or more measures of Mup input or blocks
samepageend
```

12.2 Headers and footers

Mup provides ways to put headers and footers on pages of output. Often you may want a certain kind of header and footer on the first page, but a different kind on any subsequent pages, so Mup makes it easy to do that. Mup also offers two different "layers" of headers and footers. These layers may be particularly useful if you have a single Mup file that contains multiple songs, or multiple movements of a song. In that case, there may be certain things that you want printed in headers and footers throughout, like the current page number and the name of the entire collection, but other things that you want to have change with each new song or movement.

The "outer" layer is specified by four contexts: header, footer, header2, and footer2. The "inner" layer is specified by four contexts: top, bottom, top2, and bottom2.

All of these contexts can have different versions for left and right pages, by following their name with a modifier of "leftpage" or "rightpage." So, for example, you can have both a "top leftpage" and a "top rightpage." These will override any corresponding version without a page side modifier. So if you also defined a plain "top" in addition to ones for leftpage and rightpage, it wouldn't actually ever get used. If, on the other hand, you defined only a "top leftpage" and a plain "top," then the plain top would get used for right-hand pages, and if you only define "top rightpage," and nothing for the other two, nothing would be used on left-hand pages. Pages will always alternate between left and right, but you can control which to start with, using the firstpage parameter.

All of these contexts are optional. Each of the three variations (leftpage, rightpage, and unmodified) of each outer context (header, footer, header2, footer2) can only be specified once per file. The inner contexts can be specified more than once per file, and each time a set of them occurs, a new page is started. On output pages, all of these contexts are placed horizontally between the left and right margins, and vertically the elements appear in the following order:

(topmargin)
header
top
(one or more scores of music or blocks of text)
bottom
footer
(bottommargin)

Which version is used—the one with or without the "2" suffix—depends on which page is being printed. The items in header and footer appear on only the very first page, while those in header2 and footer2 appear on all subsequent pages. Somewhat similarly, the items in top and bottom will appear on the page that is started when they are encountered in the input, while top2 and bottom2 will then be used on all subsequent pages. However, you can specify a new top and/or bottom later, which will then be used for one page, and you can specify a new top2 and/or bottom2 later, which will replace the previous top2/bottom2. Note, however, that if you change top2 but not top, that new top2 is used immediately on the new page, whereas if you change both, the new top applies to the immediately following new page, and the new top2 isn't used until the following page. Subsequent pages will use top2 in either case.

Some examples may help. First a simple case: suppose you have a single song, and you'd like a title at the top of the first page. This is straightforward:

```
top
    title "Here is the Title"
```

For this simple example, it would work just as well to use "header" instead of "top," so you can use either one, although top is slightly more flexible. Later we'll see some examples where you might use both header and top in the same file, for different kinds of titles.

Now suppose you'd like to make the title bigger and bolder, and would like to add a subtitle and composer information, as well as add a copyright notice to the bottom of the page.

```
top
   title bold (18) "Here is the Title"
   title ital (14) "Here is a subtitle"
   title "Lyrics: Ann Author" "Composer: Me"
bottom
   title "\(copyright) Copyright 2003 by Ann Author and Me"
```

Again, in this simple example, you could use "header" and "footer" rather than "top" and "bottom."

Now suppose the song is long enough to take several pages, and you would like to repeat the title along with the page number on all pages after the first. To accomplish this, you could add:

```
top2
  title "Here is the Title - \%"
```

The \% is a special marker that will get replaced on each page with the current page number. While it can be used in any text string, it is probably only likely to be useful in these header and footer kinds of contexts. Another special marker is \#, which will be replaced by the page number of the final page. This could be useful for doing something like "page \% of \#."

As a variation, perhaps you'd prefer the information at the bottom of the page.

```
bottom2
   title "This is the title"  "Page \%"
```

In this variation, two separate text strings are specified, so the first string will be left justified and the second will be right justified.

Note that if you give a top2 or bottom2, but it turns out there aren't any additional pages, they would never actually get used. But it wouldn't hurt to have set them.

Now let's consider a more complicated example, using both outer and inner contexts. Suppose you are publishing a book of songs, entitled "My Favorite Songs," and you want to put that title at the top of every page throughout the book, and you want a page number at the bottom of each page except the first. You could get that much using:

```
header
   title "My Favorite Songs"
header2
   title "My Favorite Songs"
footer2
   title "\%"
```

But suppose that in addition, you want each song to have its title on its first page in big print, and on subsequent pages in regular size print. To get this, at the beginning of each song, you could use top and top2:

```
top
   title (18) "This is the Song Title"
top2
   title "This is the Song Title"
```

Now suppose you'd prefer to have the page numbers at the left margin of left-hand pages and at the right margin of right-hand pages. Instead of the single unmodified footer2 shown above, you would make two:

```
footer2 leftpage
  title "page \%"  ""
footer2 rightpage
  title ""  "page \%"
```

A composition with multiple movements can be handled similarly, by giving header, footer, header2, and footer2 (or any subset thereof) for the composition as a whole, and then giving new top, bottom, top2, and bottom2 definition (or any subset) at the beginning of each new movement.

These contexts are conceptually in a separate coordinate space, which will be overlaid on each page coordinate space. When in these contexts, the _win special tag applies to the header or footer window rather than the space between the header and footer as it does in the music context.

Since headers or footers will expand as necessary, the value of _win.s and _win.y may be changed by any of the statements in the context. At any given time, they refer to the boundaries as defined by what has been printed so far. Thus, if the first line of a header contains an 18-point title, after that, _win.s would be 18 points below _win.n. If the header then contained a 12-point title, after that it would be 31 points below _win.n (12 points for the title string, plus 1 point of padding that is added between lines printed).

In addition to the "title" commands used in the examples above, you can also use the "print," "left," "right," or "center" commands. In most cases you will want to use "nl" for the *location* on those commands to place things. If you don't specify a *location* for the first of these commands in a header or footer, Mup will start at the left margin, just far enough down from the top to accommodate the text string to be printed. These contexts can also contain settings of the font, fontfamily, and size parameters. These parameter values will be used until the end of the context or until set again to some other value.

12.3 Macros

12.3.1 Simple Macros (without parameters)

Macros can be defined to avoid retyping or to give mnemonic names to things. A macro is defined with the following syntax:

define *macro_name macro_text* @

The *macro_name* consists of one or more uppercase letters, digits, and underscores, with the first character being a letter. The *macro_text* can be any text. It can be any length from empty to many pages. The "@" terminates the macro. A literal "@" can be placed in the *macro_text* by preceding it with a backslash. If you want a literal backslash in the *macro_text*, it also must be preceded by a backslash.

A macro is called by stating the *macro_name* in the input. The *macro_name* is replaced by the *macro_text*. A macro can be defined at any point in the input. It can be used as often as desired any time after it has been defined. A given *macro_name* can be redefined as many times as desired, with each new definition overwriting the previous definition.

As an example, suppose you are printing an orchestral score, and the oboe part happens to be on staff 5. Rather than having to remember which staff it is, you could define a macro:

```
define OBOE 5: @
```

Not only is the name easier to remember than a number, but if you later decide to move the oboe part to a different place in the score, only the macro definition and perhaps a few other things would have to be changed.

Another common use of macros might be if a musical motif occurs several times. You could define a macro for the motive:

```
define SCALE 8c;d;e;f;g;a;b;c+; @
```

then do something like:

```
OBOE SCALE
```

It is possible to remove the definition of a macro using the "undef" statement:

```
undef OBOE
```

It is possible to have parts of the input skipped over depending on whether certain macros are defined or not. This is done using "ifdef," "else," and "endif." The keyword "ifdef" is followed by a macro name. If a macro by that name is currently defined, Mup will continue reading and processing input normally. If it finds a matching "else," it will skip over input until the matching "endif." If the macro is not currently defined, Mup will skip over the input until it finds a matching "else" or "endif." There is also an "ifndef" command that uses the opposite logic: it will read the input up to the "else" or "endif" only if the macro is NOT defined.

The ifdefs can be sprinkled between other items in the input; they need not be on separate lines. They can be nested. Examples:

```
// make last c an octave higher if macro "FRED" is defined
1: c;e;g;c ifdef FRED + endif;

ifdef PIANO
    staff 1 visible=n
else
    ifdef VIOLIN
        staff 2 visible=n
        staff 3 visible=n
    endif
endif
```

Macros can also be set from the command line using the -D option. Only ordinary macros can be defined using the -D option, not macros with parameters.

Macro text cannot begin or end in the middle of a token, like a keyword or a number or a string. Usually it will be clear what things are tokens; generally tokens end where you are allowed to have white space. But there are a few obscure cases. For historical reasons that are now hard to change, "dashed tie" or "dashed slur" or the per-note versions of "dashed ~" and " dashed <>" (or the "dotted" counterparts of any of these) are treated as a single token. That means you can't define one macro for "dashed" and another for "tie" and put them together and have that be recognized.

12.3.2 Macros with parameters

Macros defined within Mup input can be defined to have "parameters." This may be useful when you have something that is repeated with small variations. When defining a macro with parameters, the macro name must be followed immediately by a (with no space between the end of the name and the parenthesis. The opening parenthesis is followed by one or more parameter names, separated by commas, and ending with a close parenthesis. Parameter names have the same rules as macro names: they consist of uppercase letters, numbers, and underscores, starting with an uppercase letter. The parameter names can then appear in the text of the macro definition where you want a value to be substituted.

As an example, suppose you are doing a score with staffs 1 through 4 for vocal parts, and staffs 5 and 6 for a piano accompaniment, and that you frequently want to mark a dynamics change at the same point in time below each of the vocal scores and between the two piano staffs. You could typically do this with something like:

```
boldital below 1-4: 1 "ff";
boldital between 5&6: 1 "ff";
```

but if you needed to do this lots of times, it could get tedious. So let's define a macro with parameters:

```
define DYN( COUNT, VOLUME )
boldital below 1-4: COUNT VOLUME;
boldital between 5&6: COUNT VOLUME;
@
```

This macro has two parameters, which have been given the names COUNT and VOLUME. When you call the macro, you will give them values. For example,

```
DYN(1,"ff")
```

would give a VOLUME of "ff" at COUNT 1, whereas

```
DYN(3.5,"mp")
```

would give a VOLUME of "mp" at COUNT 3.5.

When calling a macro with parameters, the values to give the parameters are given inside parentheses. The values are separated by commas. The values in the parentheses are copied exactly as they are, including any spaces, newlines, macro names, etc. There are only a few exceptions to this: you can include a comma, closing parenthesis, or backslash as part of a parameter value by preceding it with a backslash, and a backslash followed by a newline in a parameter value will be discarded. Thus a macro call of

```
MAC(\\\,\))
```

has one parameter, the text of which is 3 characters long: a backslash, comma, and closing parenthesis. If you backslash other characters, they will be copied without the backslash, but doing this on anything other than a double quote will produce a warning, because it seems unlikely the backslash was really needed.

If in a macro definition a parameter is used inside backticks, as in `NAME`, the value of the parameter will be placed inside double quotes. Thus, another way to do the example above would be:

```
define DYN( COUNT, VOLUME )
boldital below 1-4: COUNT `VOLUME`;
boldital between 5&6: COUNT `VOLUME`;
@

DYN(1,ff)
DYN(3.5,mp)
```

Conceptually, when the macro is expanded, the backticks are replaced by double quote marks, but in addition, any double quote mark found in the value being passed to the parameter will have a backslash inserted before it, and any backslash that occurs within double quotes in the value will also have a backslash inserted before it. Thus, for example:

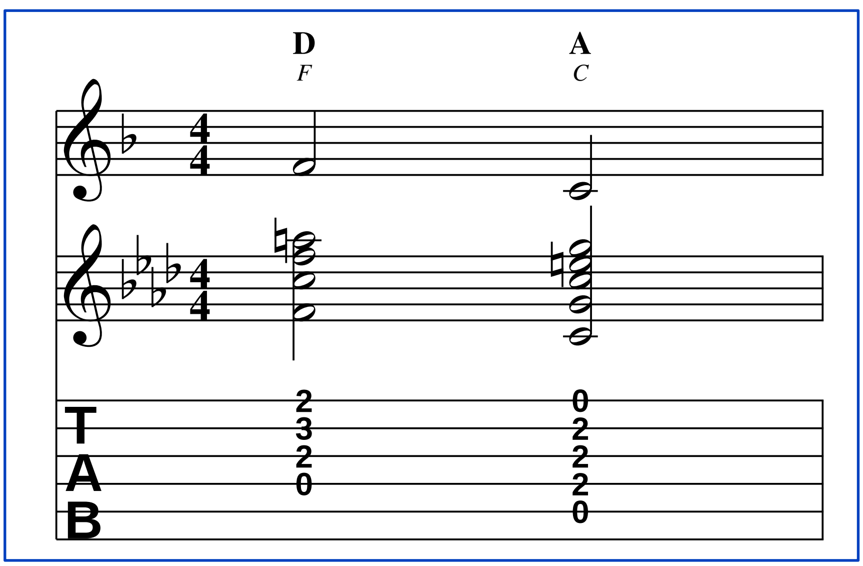
```
// If we define a macro like this:
define QUOTED(X) `X` @

// then for input    value passed is    `X` would be    which would print as

print QUOTED(hello)       hello          "hello"          hello
print QUOTED("hello")     "hello"        "\"hello\""      "hello"
print QUOTED(\\n)         \n             "\n"             a literal newline
print QUOTED("\\n")       "\n"           "\"\\n\""        "\n"
```

Sometimes it can be a little tricky to get the number of backslashes right, or other details like that. The -E Mup command line option shows how macros will expand, which may help you figure out what to do.

12.3.3 Concatenating macro names

Inside the ‘ ‘ it is possible to use ## to concatenate the values of two or more macros to form a macro name, whose value is then converted to a string. This is probably easiest to understand from an example. This example shows a way to print capo and real chords.

```
// Define some guitar chords
define  D              d0 g2 b3 e2      @
define  F        e'1 a3 d3 g2 b1 e1      @
define  A            a0 d2 g2 b2 e0      @
define  C            a3 d2 g0 b1 e0      @

// Define the mapping of F and C transposed by a minor third
define TR_F D@
define TR_C A@

// This is used by the K macro below to derive a pasted-together macro name
define TR TR_@

// Define a macro that will print both real and capo chords,
// when given a count at which to print, and the real chord name.
define K(COUNT, NAME)
   bold chord all align 2: COUNT ‘TR##NAME‘;  // capo chords
   ital(9) chord all align 1: COUNT ‘NAME‘;   // real chords
@

score
  staffs = 3
  key = 1&
staff 2
  transpose = up min 3
staff 3
  stafflines = tab
music
  // Use the K macro to print the real and capo chords
  K(1,F)
  K(3,C)
  1: 2f;c;
  // Enter the chords as you would think about them, but transpose them.
  3: 2 TR_F;TR_C;
  bar
```



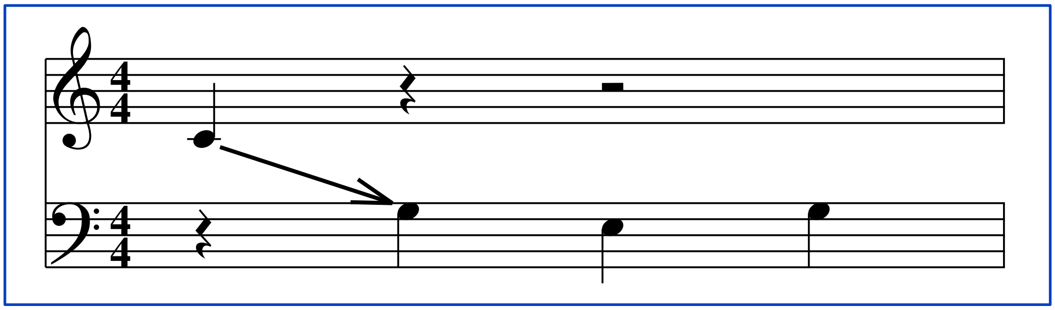
The capo chords line is the one of interest here, which has the ‘TR##NAME‘. The TR macro will be evaluated, and found to have a value of TR_. At the first place the K macro is called, the value being passed to the NAME parameter is F. Those two values (TR_ and F) are concatenated to form TR_F and that macro is then looked up, and found to have the value D. That is then made into a string (because of the ‘ ‘ enclosing the construct) and printed. On the second call to K, a C is passed to NAME, so the pasted-together name will be TR_C, which then yields "A" to be printed. On the second-last line, the TR_F gets replaced by the correct notes for a transposed F, namely a D chord, and the TR_C gets replaced by the correct notes for an A chord. Thus the various TR_F macros, along with the K macro, adjust the music for having capo. In a real example, there would likely be more chords, with an additional TR_ macro to map each one.

It is possible to paste more than two macro names together, as in ‘AA##BB##CC##DD‘ but each component must represent a valid defined name, and the result of pasting them all together must also yield a valid defined macro name.

12.3.4 A complex macro example

Let’s look at an example of much more complicated use of macros. This example demostrates the use of arithmetic functions described in the section on location tags. We will draw a line with an arrow between notes on two different staffs. While it may be possible to write the expressions directly, the result would be very hard to read and understand, so using macros to build up the pieces of the expression can be very helpful.

```
define HEAD_LENGTH             5 @
define HEAD_WIDTH              3 @
define ANGLE(X1,Y1,X2,Y2)      atan2((Y2) - (Y1), (X2) - (X1)) @
define COS(X1,Y1,X2,Y2)        cos(ANGLE(X1,Y1,X2,Y2)) @
define SIN(X1,Y1,X2,Y2)        sin(ANGLE(X1,Y1,X2,Y2)) @
define HEAD_X(X1,Y1,X2,Y2)     (HEAD_LENGTH * COS(X1,Y1,X2,Y2)) @
define HEAD_Y(X1,Y1,X2,Y2)     (HEAD_LENGTH * SIN(X1,Y1,X2,Y2)) @
define HB_X(X1,Y1,X2,Y2)       ((X2) - HEAD_X(X1,Y1,X2,Y2)) @
define HB_Y(X1,Y1,X2,Y2)       ((Y2) - HEAD_Y(X1,Y1,X2,Y2)) @
define THICK_LEN               (HEAD_WIDTH / 2) @
define THICK_X(X1,Y1,X2,Y2)    (THICK_LEN * SIN(X1,Y1,X2,Y2)) @
define THICK_Y(X1,Y1,X2,Y2)    (THICK_LEN * COS(X1,Y1,X2,Y2)) @
define FEATH_UP_X(X1,Y1,X2,Y2) (HB_X(X1,X2,Y1,Y2) - THICK_X(X1,Y1,X2,Y2)) @
define FEATH_UP_Y(X1,Y1,X2,Y2) (HB_Y(X1,X2,Y1,Y2) + THICK_Y(X1,Y1,X2,Y2)) @
define FEATH_DN_X(X1,Y1,X2,Y2) (HB_X(X1,X2,Y1,Y2) + THICK_X(X1,Y1,X2,Y2)) @
define FEATH_DN_Y(X1,Y1,X2,Y2) (HB_Y(X1,X2,Y1,Y2) - THICK_Y(X1,Y1,X2,Y2)) @
define ARROW(X1,Y1,X2,Y2)
  medium line (X1, Y1) to (X2, Y2)
  medium line (X2, Y2) to (FEATH_UP_X(X1,Y1,X2,Y2), FEATH_UP_Y(X1,Y1,X2,Y2))
    medium line (X2, Y2) to (FEATH_DN_X(X1,Y1,X2,Y2), FEATH_DN_Y(X1,Y1,X2,Y2))
@

score
    staffs = 2
staff 2
    clef = bass
music
    1: c =h; r; 2;
    2: r; g =k; e; g;
    bar

ARROW(h.x + 2, h.y - 1, k.x - 2, k.y + 1)
```



12.3.5 Expression macros

There is a special kind of macro definition, where the value is the result of evaluating a numeric expression. It uses "eval" rather than "define" as the keyword, along with an equals sign. At its simplest, the value is just a single number, so

```
eval X=1 @
```

is effectively the same as

```
define X 1 @
```

But the value can be an expression, like

```
eval T=(3+5)*5/20 @
```

which would be evaluated, setting the value of T to 2. Using all literal numbers is not likely to be very useful, but using other macros as variables can be. For example:

```
ifndef SIZE define SIZE 12 @ endif
eval WITHSIZE = SIZE + 2 @
eval LYRSIZE = SIZE-1 @
score
  size=SIZE
  withsize=WITHSIZE
  lyricssize=LYRSIZE
```

By default, that would set size to 12, withsize to 14 and lyricssize to 11, but with a single command line override, like -DSIZE=10, all three values would adjust.

Another common usage would be to increment a value. Suppose you were making a booklet with many songs, and want to automatically number them. At the beginning you could do something like:

```
eval SONGNUM = 0 @
  define NEWSONG(TITLE)
    eval SONGNUM = SONGNUM+1 \@
    block
    title bold (16) ‘SONGNUM‘+ ". " + ‘TITLE‘
    music
  @
```

and then begin each song with:

```
NEWSONG(This is the title of the song)
```

New songs could then be inserted anywhere, and later songs would automatically be renumbered appropriately. See the section on Converting a value to a string for more information on how to turn a number into a letter or Roman numeral string to be printed.

Another use for incrementing might be for staff numbers. You might start with something like:

```
eval VIOLIN = 1 @
eval VIOLA = VIOLIN + 1 @
eval CELLO = VIOLA + 1 @
eval BASS = CELLO + 1 @
```

Then if later you decided to add a second violin part, you could just replace the second line with:

```
eval VIOLIN2 = VIOLIN + 1 @
eval VIOLA = VIOLIN2 +  1 @
```

and the other instruments would move down automatically.

Numbers in expressions can either be whole numbers, like 1 or 42, or they can be numbers with a decimal point, like 0.7 or 3.14. If arithmetic needs to be done involving a whole number and a decimal number, the whole number will be "promoted" to a decimal number, with the result being a decimal number. Decimal arithmetic is done at the precision of your system, but the results (i.e., the ultimate value that the macro takes on) is limited to 6 decimal places.

Supported operators are listed in the table below, with those on a line being higher precedence than those on lines below it:

operators	**operations**	**associativity**
`( )`	grouping	left to right
`! ˜ - +`	not, one’s complement, unary minus, unary plus	right to left
`* / %`	multiply, divide, modulo	left to right
`+ -`	add, subtract	left to right
`<< >>`	left shift, right shift	left to right
`< <= > >=`	less than, less or equal, greater than, greater or equal	left to right
`== !=`	equal, not equal	left to right
&	bitwise AND	left to right
ˆ	bitwise XOR	left to right
|	bitwise OR	left to right
&&	logical AND	left to right
`||`	logical OR	left to right
`? :`	interrogation	right to left

The not, one’s complement, modulo, left and right shift, bitwise and, bitwise or, bitwise exclusive or, logical and, and logical or operators are only allowed on whole numbers. Comparision operators result in either 0 for false, or 1 for true.

There are also some functions supported. Most take a single argument, but a few take two, in which case there is a comma between them.

Name	Argument(s)	Result	Result type
sqrt	number	square root of the number	decimal
sin	angle (degrees)	sin of the angle	decimal
cos	angle (degrees)	cosine of the angle	decimal
tan	angle (degrees)	tangent of the angle	decimal
asin	number	arc sine of the number (degrees)	decimal
acos	number	arc cosine of the number (degrees)	decimal
atan	number	arc tangent of the number (degrees)	decimal
atan2	number,number	arc tangent of the ratio of the numbers (degrees)	decimal
hypot	number,number	square root of sum of the squares of the numbers	decimal
round	number	number rounded to nearest whole	whole
floor	number	number rounded down to whole number	whole
ceiling	number	number rounded up to whole number	whole

Here are some examples of expressions:

```
eval PI=3.1415 @
eval R = (1 + 0.6) / hypot(3.1, 1.896)@
eval CIRC = PI * R * R@
eval S = CIRC > R * 5 ? round(R) : ceiling(CIRC) @
eval ANGLE=asin(0.625) @
```

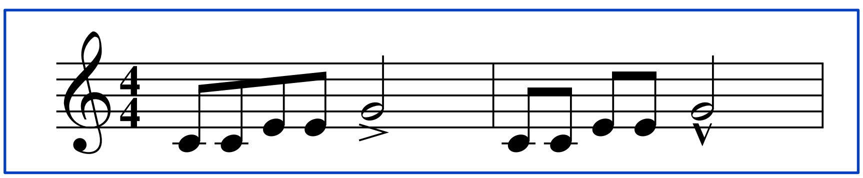
12.3.6 Saving and restoring macros

You can take a snapshot of all your current macro definitions by using the savemacros command, and then restore that set of definitions later using the restoremacros command. These are both followed by a quoted string, which is a name you give to the saved definitions. This might be useful if, for example, you had a multi-movement piece, and wanted to use one set of macros on the first and third movements, but a completely different set on the second movement. You could save the macro state at the very beginning under one name, and after the first movement under a different name. Before the second movement, you could restore to the original state with no macros defined, and define your second set. Before the third movement, you could restore the macros as they had been at the end of the first movement.

Another possible use would be if you have several standalone files, each containing a complete song, and you want to "include" those files in another file. By putting a save/restore around each include, you can prevent any macro definitions in one file from interacting with those from another file.

Here is a very simple example, using single measures rather than a whole movement or song.

```
// Define one "global" macro that will stay around,
// by defining it before doing any savemacros.
define ONE_MEAS
score
    BSTYLE
music
1: 8c;;e;;A 2g;
bar
@

// Save just that one macro as the "default."
savemacros "default"

// Now define a couple other macros and save the macros.
define BSTYLE beamstyle=2,2 @
define A [with >] @
// Note that the name can be any arbitrary text string.
savemacros "2,2 >"

// Restore the default setting and set macros
// with the same names to different values.
restoremacros "default"
define BSTYLE beamstyle=4,4,4,4 @
define A [with ^] @
savemacros "4,4,4,4 ^"

// Now use the "global" macro with each saved macro state.
restoremacros "2,2 >"
ONE_MEAS
restoremacros "4,4,4,4 ^"
ONE_MEAS
```



If you do another savemacros using the same name as you had already used, the old state is forgotten and replaced with the new.

Often, though not always, you may want to use saveparms/restoreparms at the same places as you use savemacros/restoremacros.

12.3.7 Generalized conditionals

Mup also supports more general "if" clauses. If you happen to be familiar with the preprocessors for the C and C++ programming languages, Mup "if" clauses are very similar. If you're not, that's okay, since things are explained below. Also, some of the operations are really very rarely needed, so if you find some of them confusing, you just can skip past this section; you'll likely never have a need for the complicated operations anyway.

The general form is

> **if** *condition* **then** *Mup statements* **else** *Mup statements* **endif**

As with the "ifdef," the "else" and second set of Mup statements is optional.

One form of "if" is really just a variation of ifdef. It uses the keyword "defined" followed by a macro name. So

```
ifdef DUET
```

could also be written

```
if defined DUET then
```

You may put a set of parentheses around the macro name for clarity if you wish:

```
if defined(DUET) then
```

The ! is used to mean "not," so

```
ifndef TRIO
```

could also be written as

```
if ! defined(TRIO) then
```

So far, this just looks longer, so what's the advantage? The difference is that ifdef and ifndef can only be used to check if a single macro is defined or not, whereas the "if" condition is much more general, and therefore much more powerful. Decisions can be based on the values of macros, not just whether they are defined or not, and can also be based on more than one macro at a time, Here is an example of a condition based on several macros at once:

```
if defined(FULL_SCORE) && defined(TRANSPOSE_UP) && ! defined(MIDI) then
```

would be true only if both FULL_SCORE and TRANSPOSE_UP were defined, but MIDI was not defined. The && means "and." There is also || which means "or," so

```
if defined(CELLO) || defined(STRINGBASS)
```

would be true as long as at least one of the macros was defined.

The condition can also include numbers and macros used as numeric values in arithmetic and comparisons. For example,

```
define STAFFS 3 @
define S 5 @
if STAFFS > 5 then
   // ... this would not be executed, since 3 is not greater than 5
endif
if 2 <= STAFFS then
   // ... This would be executed, since 2 is less than or equal to 3
endif
if STAFFS + 1 == S - 1 then
   // ... This would be executed, since 3+1 equals 5-1
endif
```

Note that the symbol to test for "equals" is two equals signs, not just one. This is to be consistent with what is used in the C and C++ languages. The operators for comparisons are:

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal
!=	not equal

Note that the values in the conditions can only be either literal numbers or macros whose values evaluate to a number. They cannot be things like Mup parameters. A macro that is not defined is treated as having a value of zero. Macro values are substituted for macro names just as elsewhere in Mup, so if you use a macro whose resulting value does not evaulate to a number, you may get an error or other unexpected result.

If you are familiar with "octal" and "hexadecimal" numbers, they can be used, following the C language convention of a leading zero for octal or a leading 0x for hexadecimal. (If you’re not familiar with these numbers or conventions, don’t worry about it; it’s never really necessary to use them. Just make sure you don’t accidentally start a number other than a zero with a zero).

Values are limited to 32-bit signed whole numbers. (If you don’t know what that means, what you need to know is that you can only use numbers between -2147483648 and 2147483647, and cannot use fractions.) Results of arithmetic on values will also be whole numbers, so division will result in either rounding or truncation to a whole number, and the exact characteristics may be system dependent.

Before we introduce the remaining operators, it would be good to discuss two concepts, called precedence and associativity. These determine the order in which operations are done. Consider the following expression:

```
5 + 3 * 8
```

What is its value? If we just went left to right, we would add 5 and 3, getting 8, then multiply by 8, for a final value of 64. However, multiplication is generally considered to have higher "precedence" than addition, meaning that multiplications should be done before additions. In other words, the expression should actually be treated as

```
5 + (3 * 8)
```

so we would first multiply 3 by 8, getting 24, and then add 5 and 24, obtaining a final answer of 29.

If you really intended the 64 meaning, that could be shown by parentheses, indicating you want the addition to be done first:

```
(5 + 3) * 8
```

Associativity determines whether operators of equal precedence are done left to right or right to left. Parentheses and all of the operators that have two operands associate left to right, while all the others associate right to left. For example, since addition and subtraction associate left to right, the expression

```
10 - 6 - 1
```

would be evaluated by first subtracting 6 from 10 to get 4, then subtracting 1, yielding 3. If they associated right to left, first 1 would be subtracted from 6 to get 5, which would then be subtracted from 10, yielding 5. So using different associativity can lead to different answers!

Since the "not" operator and unary minus associate right to left, in the expression

```
! - (5)
```

the unary minus would be applied first to get -5, then the "not" would be applied. But what does "not -5" mean? The "not" operator will treat its operand as a Boolean value, with a value of zero meaning false, and any non-zero value being true. Since -5 is not zero, it represents "true," and "not true" would be "false," or zero. By the way, any operator that yields a Boolean result (not, logical and, logical or, less than, greater than, less than or equal, greater than or equal, equal, or not equal) will always yield 1 for true, even though any non-zero value could mean true.

The operators are listed below. Those on the same line have the same precedence, with those on each line having higher precedence than the lines below.

<table>
<tr><th>operators</th><th>operations</th><th>associativity</th></tr>
<tr><td>( )</td><td>grouping</td><td>left to right</td></tr>
<tr><td>! ˜ - +</td><td>not, one’s complement, unary minus, unary plus</td><td>right to left</td></tr>
<tr><td>* / %</td><td>multiply, divide, modulo</td><td>left to right</td></tr>
<tr><td>+ -</td><td>add, subtract</td><td>left to right</td></tr>
<tr><td><< >></td><td>left shift, right shift</td><td>left to right</td></tr>
<tr><td>< <= > >=</td><td>less than, less or equal, greater than, greater or equal</td><td>left to right</td></tr>
<tr><td>== !=</td><td>equal, not equal</td><td>left to right</td></tr>
<tr><td>&</td><td>bitwise AND</td><td>left to right</td></tr>
<tr><td>ˆ</td><td>bitwise XOR</td><td>left to right</td></tr>
<tr><td>|</td><td>bitwise OR</td><td>left to right</td></tr>
<tr><td>&&</td><td>logical AND</td><td>left to right</td></tr>
<tr><td>||</td><td>logical OR</td><td>left to right</td></tr>
<tr><td>? :</td><td>interrogation</td><td>right to left</td></tr>
</table>

12.4 Include

The "include" statement can be used to include the contents of one file inside another.

include "*filename*"

causes input to be read from the specified *filename*. When the end of that file is reached, reading of input resumes from the original file after the include statement.

If the *filename* cannot be found as is, and it is not an absolute path, and if the environment variable MUPPATH is set, Mup will search for the file in each directory listed in MUPPATH. On Unix systems, the directories are separated by colons. On systems with DOS-like file naming conventions, they are separated by semicolons. The MUPPATH may be useful if, for example, you have a number of "boilerplate" files that you want to include in lots of songs. You can put them in some directory and set MUPPATH to list that directory; then any Mup files you have can refer to them. If a file by the name given is not found, and that name does not already have a .mup or .MUP suffix, the MUPPATH is searched again with suffix added. On systems where file names are case sensitive, .mup takes precedence over .MUP. If the file is not found in MUPPATH, then the directory of the "including" file is searched.

12.5 Converting a value to a string

A macro name can be placed inside ‘ ‘ to get a string version of it. That effectively puts double quotes around the macro value, and does any necessary escaping. That may be useful, for example, if you have a numeric value that you want to print, or want to use something both literally and as a string, as in this example:

```
// Choose a clef
ifdef USE_BASS
 define CLEF bass@
else
 define CLEF treble@
endif

score
  // Use CLEF literally
  clef=CLEF
music
// Use CLEF as a string
rom above 1: 1 "This uses the " + ‘CLEF‘ + " clef";
1: c;d;e;c;
bar
```

There is also a string() function available, which maps a number to a string, using a transform. Currently, there are four transforms available. A transform of "LET" maps the number to one or two upper case letters, while "let" maps to lower case letters. Similarly, "ROM" and "rom" will map to upper and lower case Roman numerals, respectively. Some examples:

Input	**Result**
string(1, "LET")	"A"
string(2, "LET")	"B"
string(26, "let")	"z"
string(27, "let")	"aa"
string(3, "ROM")	"III"
string(36, "rom")	"xxxvi"

Those might be useful for things like labeling movements or songs in a collection, if you wanted to use letters or Roman numerals rather than numbers. To demonstrate the functionality, here is how an outline could be created:

```
eval L1 = 1 @
eval L2 = 1 @
eval L3 = 1 @
eval L4 = 1 @

define LEVEL1(TEXT)
 left nl string(L1, "ROM") + ". " + ‘TEXT‘
 eval L1 = L1+1 \@
 eval L2 = 1 \@
 eval L3 = 1 \@
 eval L4 = 1 \@
@

define LEVEL2(TEXT)
 left nl "    " + string(L2, "LET") + ". " + ‘TEXT‘
 eval L2 = L2+1 \@
 eval L3 = 1 \@
 eval L4 = 1 \@
@

define LEVEL3(TEXT)
 left nl "        " + string(L3, "rom") + ". " + ‘TEXT‘
 eval L3 = L3+1 \@
 eval L4 = 1 \@
@

define LEVEL4(TEXT)
 left nl "            " + string(L4, "let") + ". " + ‘TEXT‘
 eval L4 = L4+1 \@
@
block

LEVEL1(This is the first top level item)
LEVEL2(This is a second level item)
LEVEL2(This is another second level item)
LEVEL3(This is a third level item)
LEVEL4(This is a fourth level item)
LEVEL3(This is another third level item)
LEVEL4(A new fourth level item)
LEVEL1(This is another top level item)
LEVEL2(This is a new second level item)
```

I. This is the first top level item
A. This is a second level item
B. This is another second level item
i. This is a third level item
a. This is a fourth level item
ii. This is another third level item
a. A new fourth level item
II. This is another top level item
A. This is a new second level item

The range for LET and let is 1 to 702 (A, B, C, ... Y, Z, AA, AB, AC, ... ZZ), and the range for ROM and rom is 1 to

3999.

12.6 Exit

The exit statement will cause the rest of the input to be ignored. This might be useful if you haven't finished cleaning up the end of a piece, but would like to take a look at (or listen to) everything before that. Anything after the exit statement will be ignored.

12.7 User-defined symbols

Mup provides the most common musical symbols, but there are various other symbols that have been used over the centuries. If Mup doesn't provide some specific symbol that you would like, you can define up to 160 of your own symbols, as well as override any of the built-in music symbols with your own versions. Most people will never need this facility, so feel free to skip past this section unless you feel you do need it. If you just want things like notes and rests changed on certain staffs, see the sections on Shaped Notes and Shape Overrides and Mensural Notation,

To define your own symbols, the description of how to draw those symbols must be provided as PostScript code. On the one hand, that means that unless you can find someone else's definition of just what you want, you will need to understand PostScript at least well enough to define your own. On the other hand, this gives access to the full power of the PostScript language, which lets you define essentially any kind of symbol you want. The PostScript language is beyond the scope of this User's Guide, but there are many books and Internet resources to help you learn, if you are interested. You can also look at the output that Mup generates for examples. If you want something similar to an existing Mup symbol, looking at its definition is probably the best place to start.

A user-defined symbol or override is put in "symbol" context. The general format of this context is as follows:

> **symbol "***name***"**
> > **postscript = "***PostScript code to draw the symbol***"**
> > **bbox =** *llx, lly, urx, ury*
> > **ystemoffset =** *up, down*

The three parameters can be supplied in any order, but only once each per symbol. The ystemoffset parameter is optional, and only used for symbols that will be used as noteheads. This is described in more detail later. The name and the postscript and bbox parameters are mandatory.

The name for a user-defined symbol must include at least one letter, and can include letters, numbers, and underscores. It must be different than existing names for non-ASCII characters.

You can refer to user-defined symbols just like the built-in symbols, using their name in a mussym statement, or in a text string using the name inside \( ). Adding the prefix "sm" to a name will result in a "small" version, just like with the built-in symbols.

In addition to naming and defining your own symbols, you can also override the definitions of existing music symbols. Suppose, for example, you want a different style of C clef. You can then supply a symbol context for "cclef" and provide your own PostScript.

A symbol definition applies to the entire file, no matter where in the file it is defined. If you try to define the same symbol more than once, a warning will be issued, and the last definition will be used. A user-defined symbol must have its "symbol" context before it is referenced.

The symbols you define will become part of a PostScript Type 3 font. Symbols must be defined in a 1000 unit scale coordinate space, where 300 units equals one stepsize, with the symbol's "logical center" at (0, 0). Limitations of Type 3 fonts apply. For example, since PostScript does not allow using setrgbcolor or sethsbcolor after a setcachedevice operation in a BuildChar procedure, it is not possible to set the color of a symbol. The PostScript code will be written to the output exactly as it is, so you will have to provide any escaping needed by PostScript. The normal Mup text string escape rule of needing to put a backslash before a double quote inside a text string also applies. So, for example, suppose you wanted to output a PostScript style string containing a parenthesis inside double quotes. Since PostScript requires a parenthesis inside a string to be preceded by a backslash, the Mup input would require the backslash as well, and since Mup requires double quotes to be backslashed, that would need to be done. So Mup input of:

```
postscript="(\"\(\")"
```

would result in:

```
("\(")
```

being copied to the PostScript output. (Of course, that is not a good symbol definition; it is just intended to show needed escapes.) As with any Mup text string, the postscript string can be as many lines long as you wish. But note that the
postscript = "
part must all be on one line.

There isn't currently any good way to call a PostScript procedure from within a symbol definition. So if you have some some code that is common to several symbols, probably the best way to handle that is to define a Mup macro for the common code, and then use the macro in the postscript string:

```
define COMMONCODE  ....the common code....  @
postscript = "...some specific code  " + ‘COMMONCODE‘ + "  more specific code"
```

The bounding box lists four numbers, giving the (x,y) coordinates of the lower left and upper right corners of the smallest box that can contain the symbol. It is in the same units as the symbol itself: 300 units per stepsize. Normally you should define the bounding box to include a small amount of white space padding around what is actually drawn, typically about 100 units, so that if it is printed right next to another symbol, they won't touch. However, in the case of a symbol you want to use as a notehead, you should not include any padding, so that stems will touch the notehead.

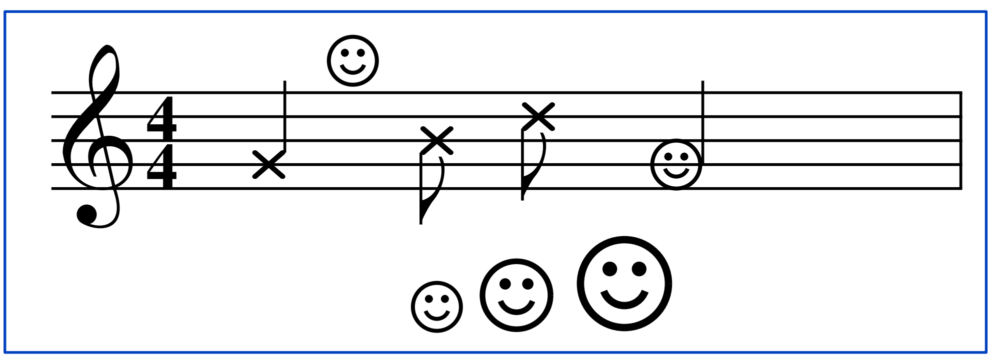
If you want to use a user-defined symbol as a notehead, you need to specify the "ystemoffset" parameter. If the symbol is not to be used as a notehead, you should omit that. The ystemoffset is given as two numbers, the first for when the stem is up, the second for when it is down. They specify, in the same units as the symbol itself (300 units per stepsize), where on the y axis the stem should begin. Any notehead should be defined with its vertical center at y of zero. So if you want the stem to start at the vertical center of the note, the ystemoffset would also be zero. A stepsize below the center of the note would be -300, a half stepsize above would be 150, etc. In the x dimension, the stem is always placed at the edge of the note, as given by the bbox.

If you need an "upside-down" version of a notehead, just name the upside version of the symbol with a "u" prefix on the name of the right side up version. Then you can use the normal u? convention in headshapes context, and Mup will take care of the rest.

Once defined, a user-defined symbol can be used just like the built-in symbols. There are only a few places where Mup uses some special knowledge of the characteristics of music symbols, but in those cases, if you redefine the symbol to have different characteristics, Mup may not place things quite as perfectly as you would like. One place where this might happen is if you redefine the symbol for flat, natural, or dblflat, to make them significantly different, since Mup tries very hard to pack accidentals as tightly as possible, based on some intimate knowledge of the shapes of the built-in versions.

Here is an example that shows both overriding a built-in symbol and defining a completely new symbol, and then using them.

```
// Override the built-in xnote to have thicker lines
symbol "xnote"
    bbox=-435, -365, 435, 375
    postscript="gsave
      % normal xnote has 120 linewidth; we set to 300 here
      1.15 1 scale 300 setlinewidth 1 setlinecap
      -300 -300 moveto 300 300 lineto stroke
      -300 300 moveto 300 -300 lineto stroke
      grestore"
    ystemoffset=300, -300

// Define a smiley face symbol.
symbol "Smiley"
    bbox=-700, -700, 700, 700
    postscript="gsave
      100 setlinewidth
      % the face
      0 0 600 0 360 arc stroke
      % the mouth
      0 0 300 -160 -20 arc stroke
      % the eyes
      -200 200 100 0 360 arc fill
      200 200 100 0 360 arc fill
      grestore"
    ystemoffset = 0, 0

headshapes
    // Set up to use the extra-thick xnote
    // for quarter or shorter, and Smiley for longer.
    "boldxnote" "xnote Smiley Smiley Smiley"
music

1: [hs "boldxnote"]...g;8b;d+;2g;
// You can use the user-defined symbol just like a built-in
mussym above 1: 1.5 "Smiley";
// You can apply size just like for a built-in
rom below 1: 2 "\(Smiley) \s(+5)\(Smiley) \s(+5)\(Smiley)";
bar
```



12.8 Installing other fonts

The "fontfile" statement is used to override a Mup font with some other font. You may place one or more of these anywhere in Mup input. This might be used either because you'd like a different style of printing, or because you need a different alphabet. The "mkmupfnt" program that is distributed with Mup can be used to generate a font description file. You can then use

```
fontfile "file"
```

in your Mup program, where the given "file" is the name of the font description file. The manual page for mkmupfnt describes how to use that program. It also describes the format of the font description file, in case you wish to generate it by some means other than the mkmupfnt program.

Mup searches for fontfiles via the MUPPATH mechanism, just like for included files.

13. CUSTOM ACCIDENTALS AND ALTERNATE TUNINGS

Mup supports custom accidentals: the ability to use any character, including a user-defined symbol, as an accidental. They can be placed on notes, though not in key signatures. Feel free to skip this section; many Mup users will probably never use the features described here.

If you want to generate MIDI for a piece using custom accidentals, a method is provided for defining their frequency offsets. You can also override the default frequency offsets of the standard (built-in) accidentals (#, &, n, x, &&). Also, you can specify a tuning other than the standard equal temperament tuning, and you can define a pitch level different from the standard of the note a4 being 440 Hz.

Note that some MIDI players, including the standard Microsoft Windows Media Player, do not support the special MIDI commands that are needed for these features.

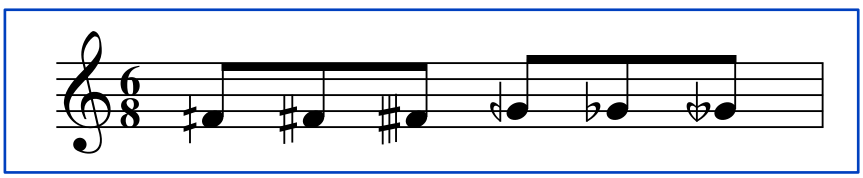
13.1 Printing custom accidentals

Although you can use an existing character as a custom accidental, typically you would want to define a new one, as explained in the section on user-defined symbols. When you put a custom accidental on a note, the x axis of the symbol (the place where y is 0) will be aligned vertically with the note. The position of the y axis (the place where x is 0) does not affect the positioning.

The Mup distribution contains two "include" files of symbol definitions that can be used as custom accidentals. The file "quarterstep_accs" defines accidentals intended for use as quarter steps and three quarter steps. The file "helmholtz_accs" defines many of the accidentals needed for a system called the Extended Helmholtz-Ellis Just Intonation Pitch Notation. Even if you don't intend to use these files in your music, you can use them as examples of how to define symbols.

In a Mup input file, a custom accidental is put after a note letter similarly to a standard accidental. It is specified by the name of the accidental as a string inside braces.

```
include "quarterstep_accs"
music
1: 8f{"qsharp"}; f#; f{"tqsharp"}; g{"qflat"}; g&; g{"tqflat"};
bar
```



To reduce the amount of typing, you can define macros for the accidentals. For example:

```
define QS {"qsharp"} @
```

You can put up to four accidentals on a note. They can be any combination of standard and custom accidentals. You can put parentheses around the entire set, and parentheses will be drawn around the set. You cannot put parentheses around just a subset of them.

13.2 MIDI for custom accidentals and alternate tunings

The parameters and the accidentals context described in this section need not be used if you do not intend to generate MIDI. In that case it is only necessary to define the symbols for the custom accidentals you are using.

13.2.1 Defining custom accidentals for MIDI

To generate MIDI for a piece that uses custom accidentals, their frequency offsets must be defined in an "accidentals" context. A frequency offset tells how much the accidental alters the pitch of the note. It can be specified in any of these ways:

- A positive number by which the note's frequency is to be multiplied. It may include a decimal fraction part (e.g., 0.97 or 1.5). Of course, a number greater than 1 will raise the pitch, and a number less than 1 will lower the pitch.
- A ratio of two positive numbers; that is, two numbers with a / between them. Mup will divide the numbers and multiply the note's frequency by the result. The numbers may include decimal fraction parts, though whole numbers are more often used when stating ratios for this.
- A + or - sign followed by a number and the word "cents". The number may include a decimal fraction. A cent is 1/100 of an equally tempered halfstep, or in other words, 1/1200 of an octave. The note's frequency will be multiplied by 2 to the cents/1200 power. So a positive cents value will raise the pitch, and a negative cents value will lower the pitch.

Each line in an accidentals context contains a symbol name in quotes followed by either the word "all" followed by a single offset, or a list of 1 to 7 note letter / offset pairs. In the single offset case, this same offset will be applied when the accidental is used on any of the 7 notes, a through g. In the other case, each offset follows the note letter it is to be applied to. Not all note letters need to be listed, but a custom accidental cannot be used if the value for its note letter has not been defined.

Any of the standard accidentals can also be listed in an accidentals context: "sharp", "flat", "nat", "dblsharp", "dblflat". Offsets are set for them the same way as for custom accidentals. If any notes are not specified in a list of note letter / offset pairs, the default offset (as described in the section below on tuning) is used for them. A standard accidental for which no line is provided will use the default value for all notes.

By default a natural has a ratio of 1 and it doesn't change the pitch. But you can define a different ratio for it. Note also that when a note has no accidental, it is treated the same as if it had a natural on it.

The first line of an accidentals context consists of the word "accidentals" followed by a name you provide in quotes. To make use of it, in a score context you set the acctable parameter equal its name in quotes. It will be used starting at that point in the file until such time as you change acctable. The accidentals context must occur earlier in the file than the place it is referenced. You can define any number of accidentals contexts. If "acctable" is not set, only standard accidentals can be used, and they will have their default offsets.

The following is an example of an accidentals context:

```
accidentals "mine"
    "acc1"      all 23/22
    "acc2"      all 1.0873
    "acc3"      all +87 cents
    "acc4"      c 17/18  f 0.9103  g -93 cents
```

It is possible to define an invisible accidental. This is useful in cases where you want to alter the pitch of a note for MIDI but you don't want it to print anything. In the symbol definition for an invisible accidental, you print nothing, and you need to set the bounding box to all zeros.

13.2.2 Setting the pitch level

The standard pitch of the note a4, the A above middle C, is 440 Hz. With the a4freq parameter, you can set this to a different number. The frequencies of all other notes are figured relative to a4, so if you set a4freq to frequency X, their frequencies will be multiplied by X/440.

13.2.3 Setting the type of tuning

The tuning parameter lets you set the tuning system to "equal", "pythagorean", or "meantone". Each tuning sets differently the frequencies of the "white" notes (a through g) and the default offsets of the standard accidentals. Naturals, however, have default ratio 1 (or 0 cents) in all of these tunings.

13.2.3.1 Equal temperament

The default value of "tuning" is "equal", which means equal temperament, the modern standard for Western music. In this system, all half steps are equal. Since there are 12 half steps in an octave, each has a ratio of the twelfth root of 2, about 1.05946, which is 100 cents by definition. When this tuning is set, the default value of a sharp is this ratio, and a flat is the inverse, 0.94387, or -100 cents. For a double sharp you square the sharp ratio (1.12246) or double the number of cents (200). For a double flat you square the flat ratio (0.890900) or double the number of cents (-200).

13.2.3.2 Pythagorean tuning

Pythagorean tuning is based on a perfect fifth having a ratio of 3/2 (1.5), which works out to about 702 cents. Starting from the reference note a4, the other notes are determined by going around the circle of fifths, using this ratio. So, if you are using the default 440 for a4, e5 would be 660, b5 would be 990, etc. To get from any natural note to its sharp version in the same octave (like f4 to f#4), you go up 7 fifths and down 4 octaves: (3/2)^7 / 2^4 = 2187/2048, about 1.06787 or 114 cents. So a sharp is defined to be that ratio. A flat is the inverse, 0.93644, or -114 cents. For a double sharp, square the sharp ratio (1.14035) or double the cents (227). For a double flat, square the flat ratio (0.87692) or double the cents (-227).

Pythagorean tuning is usually not used with only its default intervals and accidentals. It is usually used with custom accidentals or redefining the ratios for standard accidentals, as shown in examples below.

13.2.3.3 Meantone tuning

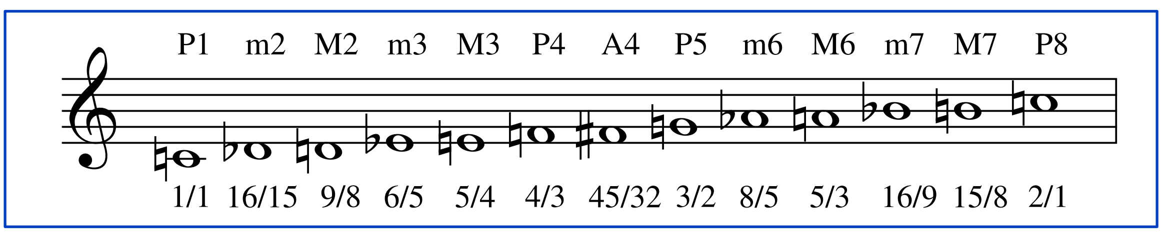
Meantone tuning is based on a major third being 5/4 (1.25), or about 386 cents. Perfect fifths are the fourth root of 5, so that if you go up 4 fifths and down 2 octaves, you get the 5/4 for the major third. Using this value for a fifth (about 1.49535 or 697 cents), the other notes are determined by going around the circle of fifths, using this ratio. To get from any natural note to its sharp version in the same octave (like f4 to f#4), you go up 7 fifths and down 4 octaves: (5^(1/4))^7 / 2^4, about 1.044907 or 76 cents. So a sharp is defined to be that ratio. A flat is the inverse, 0.95702, or -76 cents. For a double sharp, square the sharp ratio (1.09183) or double the cents (152). For a double flat, square the flat ratio (0.91589) or double the cents (-152).

Meantone is a compromise that lets you approximate the sound of just intonation in any key, without having to use custom accidentals or redefining standard accidentals.

13.2.4 Using just intonation

This section illustrates the use of custom accidentals and the related parameters by showing two ways to set up standard just intonation scales, with all the major, minor and perfect intervals, plus the augmented fourth. For an example, we will use a key note of c, and we will have a4 be 440 Hz.

Here are the ratios for the just intervals from c up to each other note:



For both methods, we'll start with pythagorean tuning. Going around the circle of fifths starting from c, we find that this g and d agree with pythagorean. But a would be 27/16. To arrive at 5/3, we need to multiply the 27/16 by 80/81. This lowers it by a "syntonic comma". It turns out that e, b, and f# also need to be lowered by that amount. Going the other direction, we find that f and b& agree, but e&, a&, and d& need to be raised by 81/80, a syntonic comma.

For the just a to be 440 Hz, the pythagorean a has to be 445.5 (multiply by 81/80). So we set a4freq to 445.5.

13.2.4.1 First method: redefine standard accidentals

For the first method, we use normal notation (standard accidentals), but redefine their offsets as necessary to achieve the ratios shown above.

Consider e&. The pythagorean e is 81/64. (Up four fifths from c, then down two octaves.) To get to the 6/5 that we need for e&, we have to multiply 81/64 by 128/135. So when they are used on the note e, flats should be 128/135.

Using similar reasoning, we find that flats on a and d should also be 128/135. But flats on b should be 2048/2187.

The sharp on f should be 135/128.

Naturals also need to be redefined when they differ from their pythagorean values. We have already found those values: naturals on a, e, and b need to be 80/81. The other natural notes agree, and we can let them default to 1/1. Remember, when a note appears without any accidental on it, is is treated the same as if it had a natural.

So, the following will set up everything as required:

```
accidentals "key_of_c_just"
    "flat"      d 128/135  e 128/135  a 128/135  b 2048/2187
    "sharp"     f 135/128
    "nat"       e 80/81  a 80/81  b 80/81
score
    a4freq = 445.5
    tuning = pythagorean
    acctable = "key_of_c_just"
```

Notes not defined in this table will default to their pythagorean values. We have found that those are the correct values for naturals. But they are probably wrong for sharps, flats, double sharps, and double flats. If you wanted to use any of those notes, you would have to decide what its ratio should be, and then figure out what offset should be put in the table. Especially for notes far from the key center (c), it is not obvious what ratio you would want.

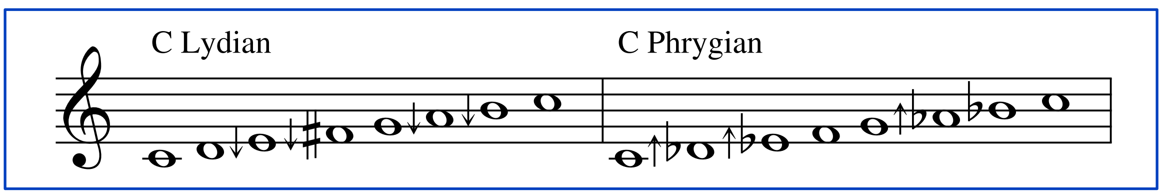
This brings up a problem. You may need multiple versions of some notes, even for ones not far from the key center. For example, the ratio 5/3 is used for a. That works well if you are playing an F major chord, fac+. But if you are playing a D minor chord, dfa, that is not the right ratio. To get a perfect 3/2 fifth from d to a, you would want a to be 27/16. This leads us to the second method.

13.2.4.2 Second method: use custom accidentals

This method also uses pythagorean tuning as a basis, but instead of redefining the standard accidentals to different values for different notes, it defines custom accidentals for offsets that differ from the pythagorean defaults. Each accidental, standard or custom, has the same offset no matter what note it is applied to.

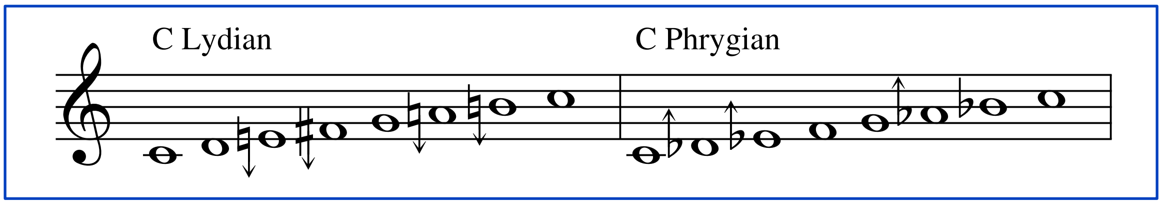
In simple cases such as the scales we are defining in this section, it turns out that each note either agrees with the pythagorean ratio, or else is higher or lower by a syntonic comma, 81/80 or 80/81. You could define accidentals for up and down commas, and use them alone or together with standard accidentals, as follows:

```
symbol "commaup"
    bbox = -350, -700, 350, 700
    postscript = "gsave
        0 600 moveto -250 135 lineto -230 135 lineto
        0 420 lineto 230 135 lineto 250 135 lineto fill
        70 setlinewidth 0 500 moveto 0 -600 lineto stroke
    grestore"
symbol "commadn"
    bbox = -350, -700, 350, 700
    postscript = "gsave
        0 -600 moveto -250 -135 lineto -230 -135 lineto
        0 -420 lineto 230 -135 lineto 250 -135 lineto fill
        70 setlinewidth 0 -500 moveto 0 600 lineto stroke
    grestore"
accidentals "just"
    "commaup"   all 81/80
    "commadn"   all 80/81
score
    a4freq = 445.5
    tuning = pythagorean
    acctable = "just"
    define CU {"commaup"} @
    define CD {"commadn"} @
music
    rom 1: 1 "C Lydian";
    1: 1c; d; eCD; fCD#; g; aCD; bCD; c+;
    bar
    rom 1: 1 "C Phrygian";
    1: 1c; dCU&; eCU&; f; g; aCU&; b&; c+;
    bar
```



Or, you may instead want to define a set of custom accidentals that act as combinations of standard accidentals and commas, and perhaps some accidentals for other ratios. One such set was mentioned earlier: the set defined in the "helmholtz_accs" file that is in the Mup distribution. Using those definitions, the above example looks like this:

```
include "helmholtz_accs"
score
    a4freq = 445.5
    tuning = pythagorean
    acctable = "helmholtz"
    define SD {"sharpdn"} @
    define ND {"natdn"} @
    define FU {"flatup"} @
music
    rom 1: 1 "C Lydian";
    1: 1c; d; eND; fSD; g; aND; bND; c+;
    bar
    rom 1: 1 "C Phrygian";
    1: 1c; dFU; eFU; f; g; aFU; b&; c+;
    bar
```



13.2.5 Debugging tuning settings

To help verify that you have specified the correct input, you can run the "mup" command with the debugging option of "-d 512", along with an option to generate MIDI. In addition to some other debugging output, it will print tables to standard error output, which show the frequency being used for each note in your piece. If you change any of the parameters along the way, it will print a table for each section of your piece as delimited by the places where the parameters change.

14. PARAMETERS

Parameters can be set in various contexts. All parameters have default values, so that you need to explicitly set them only if you want some value other than the default value.

Some parameters can be set only in the score context. Others can be set in either score or staff contexts. A few can be set in score, staff, or voice context. If a given parameter can be set in several contexts, the value is that of the parameter at the most specific context in which it is set. For example, if Mup is working on musical data for voice 2 of staff 5, and it needs to look up the value of a parameter, it will first see if that parameter has been set in context "voice 5 2". If so, it will use that value. If not, it will see if the parameter was set in context "staff 5". If that has not been set either, it will use the value from the score context. The score context initially has all parameters set to their default values. There are a few parameters, mostly related to font and text size, that can also be set in header, footer, and similiar contexts.

Parameters are set by the following syntax:

parameter_name=*value*

Several parameters can be set on a single line by separating them with a semicolon. For example:

```
staffs=2 ; key=2& ; time=2/4
```

Parameters can be unset in staff or voice context using

unset *parameter_name*

Unsetting a parameter in staff context will cause it to revert to its value in score context (unless overridden in voice context). Unsetting a parameter in voice context will cause it to revert to its value in staff context if that is set, otherwise to its value in score context.

There are a few parameter that can be changed mid-measure. That is noted in their descriptions.

It is possible to take a "snapshot" of the current state of all the parameters, and then restore that state later. This is done in the "control" context. When you want to take a snapshot, you do

```
control
  saveparms "some name"
```

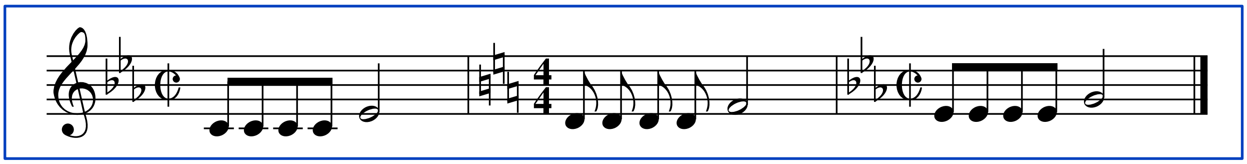
and then to restore, you do

```
control
  restoreparms "some name"
```

using whatever name you used for the save. The name can be anything you like. You can save as often as you like. You can restore from any previous save.

Here is a simple example that saves the default parameter settings, changes three parameters, and saves again under a different name. It then uses the changed parameters for one measure, restores the defaults for one measure, then restores to the changed values for a measure.

```
control
    saveparms "default"
score
    time=cut
    beamstyle=2,2
    key=3&
control
    saveparms "22"
music
1: 8c;;;;2e;
bar
control
    restoreparms "default"
music
1: 8d;;;;2f;
bar
control
    restoreparms "22"
music
1: 8e;;;;2g;
endbar
```



Sometimes when you use saveparms/restoreparms, you may want to use savemacros/restoremacros as well.

The parameters are listed below in alphabetical order. For each, the description includes the parameter's name, legal values, default value, when it takes effect, and contexts in which the parameter can be set, along with an example of its usage. If there are other related parameters, they are referenced as well.

Name a4freq
Description..... This parameter specifies the frequency, in Hertz, of the A in octave 4, which is the A above middle C.
Value............... 100.0 to 1000.0
Default............ 440.00
Context........... score
Takes effect .... at next music context
Example .... a4freq = 420.15
See also........... acctable, tuning

Name .... aboveorder
Description..... This parameter specifies in what order to stack items that are printed above a staff. The value is a comma-separated list of all the types of things that can be printed above a staff. Items are stacked in the order listed, starting from just above the staff and working upward. If you want several types to be handled as a single category, with all types in the category having the same stacking priority, separate them with an ampersand rather than a comma. The ampersand cannot be used with lyrics, ending, or reh. The dyn category applies to crescendo and decrescendo marks (from "<" and ">" statements) as well as text with the dyn modifier. The chord category applies to text with chord, analysis, or figbass modifiers. The othertext category applies to rom, bold, ital, and boldital items that do not have a chord, analysis, figbass, or dyn modifier. If you omit any categories, they will be stacked last, in their default order.
Value.............. mussym, octave, dyn, othertext, chord, lyrics, ending, reh
Default............ mussym, octave, dyn & othertext & chord, lyrics, ending, reh
Context........... score, staff
Takes effect .... at start of next score
Example .... aboveorder = mussym, lyrics, dyn, octave, othertext, chord, ending, reh
See also........... beloworder, betweenorder, chorddist, dist, dyndist

Name .... acctable
Description..... This parameter specifies which table of accidentals, as defined in an accidentals context, to use. It can be set to nothing, to mean to use only the normal standard accidentals (#, &, x, &&, n) with their default meanings as implied by the tuning parameter.
Value.............. a quoted string that matches the name used for an accidental context
Default............ not set
Context........... score
Takes effect .... at next music context
Examples........ acctable = "my_accs"
acctable = // set back to default
See also........... a4freq, tuning

Name .... addtranspose
Description..... This parameter specifies by what additional interval to transpose the music data. There is another parameter called just transpose. Typically you would use the transpose parameter to change the key of individual staffs (for transposing instruments), and then use the addtranspose parameter if you want to change the key of the entire score. But either of these parameters can be used either way. In any case, for each staff, and for the score, the values of transpose and addtranspose are both applied, one after the other, to the current key signature, notes, and chords to determine their resulting values. The interval can be larger than an octave, but must be a valid interval (e.g., there is no such thing as a perfect 6th). It is an error to specify a transposition value that would result in a key signature with more than 7 flats or sharps. It is also an error if transposition would result in a note requiring a triple sharp or triple flat.
Value.............. the word "up" or "down," followed by an interval and a whole number greater than 0. You can optionally add the keyword "notes" or "chords" at the end, to restrict the transposition to just notes or just chord symbols; by default, both are transposed. The interval is one of major, minor, augmented, diminished, or perfect. The intervals can be abbreviated to their first three letters (maj, min, aug, dim, or per). The section on transposition lists transposition intervals and gives further details. Depending on which key signature you are transposing from, some transposition intervals may not work, because they result in more than 7 flats or sharps.
Default........... up perfect 1 (i.e., no transposition)
Context.......... score, staff
Takes effect .... at next music context
Examples........ addtranspose = down major 3
addtranspose = up perfect 5
addtranspose = up minor 2 notes
See also.......... a4freq, key, transpose, useaccs

Name .... alignlabels
Description..... This parmaeter specifies how to align labels for staffs, including those for brace and brackets group. Each nesting level will be aligned independently. This applies to labels relative to each other, not to lines within a single multi-line label.
Value.............. center, left, or right
Default........... right
Context.......... score
Takes effect .... at start of next score
Example .... alignlabels = center
See also.......... brace, bracket, label, label2

Name .... alignped
Description..... If set to n, this parameter allows pedal marks to individually move closer to the staff. If set to y, all pedal marks on a score are aligned. If set to n, you can use align on pedal marks, and in either case, you can use dist, to force them to a particular place.
Value.............. y or n
Default........... y
Context.......... score, staff
Takes effect .... at start of next score
Example .... alignped=n
See also.......... pedstyle

Name .... alignrests
Description..... This parameter controls whether rests stay as close to the middle of the staff as possible, or whether they are adjusted to align with the notes surrounding them. If the value is "n," no alignment to the notes is done. If the value is "y," rests are moved to try to follow the flow of the voice. The alignment of rests with notes is only done when the vscheme parameter is something other than 1, and if there are no non-space chords in voice 3. It is not used on voice 3. This parameter can be changed in mid-measure. Only notes that are inside the time period when this parameter is set to y will affect alignment.
Value.............. y or n
Default............ n
Context........... score, staff, voice
It is possible to change this parameter in the middle of a measure, using << >>.
Takes effect .... at next music context, or immediately if mid-measure
Example .... alignrests=y
See also........... vscheme

Name .... barstyle
Description..... This parameter specifies which staffs are to have their bar lines connected together. When drawing bar lines, a continuous vertical line will be drawn from the top line of the top staff in a range to the bottom line of the bottom staff of the range. Any staff not listed will be barred by itself, with the bar line spanning only the height of the staff.
Value.............. a comma-separated list of staff numbers and/or ranges of staff numbers. Staff numbers can be from 1 to the value of the "staffs" parameter. A range is a pair of numbers separated by a dash. You can also specify all, which means to bars all staffs together regardless of how many there are. You can specify between, which causes the bar lines to be drawn between adjacent staffs and not through them. If you want a mixture of between and not between, all the non-between ranges must be specified first, then the word between, then the between ranges. A given staff number can be specified only once, and there can be no overlapping between ranges.
Default............ each visible staff barred individually
Context........... score
Takes effect .... at next music context
Example .... barstyle = 1-2, 5-8
barstyle = all
barstyle = 1-4,10-12, between 5-9,13-17
barstyle = between all
See also........... staffs, subbarstyle, visible

Name .... beamslope

Description..... This parameter allows you to control the slope of beams. Two values must be given, separated by a comma. Mup calculates an appropriate slope for beams by applying a linear regression algorithm that uses the positions of the noteheads within the beam. The first value supplied for the beamslope parameter is a factor by which to multiply the default slope that Mup calculates. The minimum value of 0.0 would cause all beams to be horizontal, whereas the maximum value of 1.0 will use the slope Mup calculates. Intermediate values will yield beams that are less slanted than the default slope calculation. The second value given to the beamslope parameter is the maximum angle for the beam, in degrees. If the originally calculated value multiplied by the factor yields an angle of greater than this maximum angle, the maximum angle will be used. Cross-staff beams that are between staffs are allowed to have a slope up to 1.4 times the value of the slope of the maximum angle, since they face more constraints. The slope can be overridden on a particular beam by specifying a slope as an interchord attribute on the first chord of the beamed set.

Value.............. 0.0 to 1.0 for the factor, and 0.0 to 45.0 for the maximum angle

Default........... 0.7, 20.0

Context........... score, staff, voice

Takes effect .... at next music context

Example .... beamslope=0.8,20

See also........... tupletslope

Name .... beamstyle

Description..... This parameter specifies how to beam eighth notes or shorter. It is specified as a list of time values. Any number of notes up to each time value will be beamed together. For example, in 4/4 time, with beamstyle=4,4,4,4 each quarter note worth of shorter notes would be beamed together. However, beams would not span across beats. As another example, for an input of 4.; 8; 8; 4.; the two eighth notes would not be beamed together, because they span beats. If beamstyle had been specified as 4,2,4 then the eighth notes would be beamed. Normally, beams also end whenever a rest or space is encountered. However, if an "r" is placed at the end of the list of time values, Mup will beam across rests of less than quarter note duration. Similarly, if an "s" is placed at the end of the list, Mup will beam across spaces of less than quarter note duration. You can specify both r and s in either order. It is possible to specify subbeams, or secondary groupings within a beam, by enclosing a list of time values in parentheses. In this case, the outer beam extends for the sum of the values in the parentheses, while inner beams extend only for the individual values within the parentheses. For example, if you set beamstyle=(4,4),(4,4) and then have a measure that consists of all 16th notes, the first eight notes would be connected by an outer beam, as would the last eight notes, but the second (inner) beams would cover only four notes each. The parentheses cannot be nested. It is possible to override this default beaming style within a specific measure. See the section on Custom Beaming for examples of how to obtain various kinds of beaming. The value of the beamstyle parameter is remembered for any later changes back to the same time signature. In other words, if you set the value of the time parameter and the beamstyle parameter in the same context, then later set only the time, the beamstyle that you had set earlier will be used again.

Value.............. a comma-separated list of time values that add up to a measure. Time values are specified as 4 for a quarter note, 2 for half note, etc, and can be dotted if necessary, or an expression with added and/or subtracted times. The list can optionally be followed by an "r" to indicate beams should span rests. It can optionally be followed by "s" to indicate beams should span spaces. Two or more of the time values may be enclosed in parentheses, to indicate sub-groupings of inner (secondary) beams within outer (primary) beams. If the value is empty, automatic beaming is turned off.

Default........... no beams; each note of eighth or shorter duration is individually flagged.

Context........... score, staff, and voice

Takes effect .... immediately for purposes of checking for interactions with other parameters, at next music context for printing

Examples........ beamstyle = 4,4,4,4
beamstyle = 2.
beamstyle=4+16, 4+16
beamstyle = 2, 2 rs   // beam across rests and spaces
beamstyle = (4., 4., 4.) // one outer beam per measure,
        // with inner beams broken at each dotted quarter duration
beamstyle =   // turn off beaming

Name .... beloworder
Description..... This parameter specifies in what order to stack items that are printed below a staff. The value is a comma-separated list of all the types of things that can be printed below a staff. Items are stacked in the order listed, starting from just below the staff and working downward. If you want several types to be handled as a single category, with all types in the category having the same stacking priority, separate them with an ampersand rather than a comma. The ampersand cannot be used with lyrics or pedal. The dyn category applies to crescendo and decrescendo marks (from "<" and ">" statements) as well as text with the dyn modifier. The chord category applies to text with chord, analysis, or figbass modifiers. The othertext category applies to rom, bold, ital, and boldital items that do not have a chord, analysis, figbass, or dyn modifier. If you omit any categories, they will be stacked last, in their default order.
Value.............. mussym, octave, dyn, othertext, chord, lyrics, pedal
Default............ mussym, octave, dyn & othertext & chord, lyrics, pedal
Context........... score, staff
Takes effect.... at start of next score
Example......... beloworder = mussym, lyrics, dyn, octave, othertext, chord, pedal
See also........... aboveorder, betweenorder, chorddist, dist, dyndist

Name .... betweenorder
Description..... This parameter specifies in what order to stack items that are printed between two staffs. The value is a comma-separated list of all the types of things that can be printed between staffs. Items are stacked in the order listed, starting from a baseline and working upward. If you want several types to be handled as a single category, with all types in the category having the same stacking priority, separate them with an ampersand rather than a comma. The ampersand cannot be used with lyrics. The dyn category applies to crescendo and decrescendo marks (from "<" and ">" statements) as well as text with the dyn modifier. The chord category applies to text with chord, analysis, or figbass modifiers. The othertext category applies to rom, bold, ital, and boldital items that do not have a chord, analysis, figbass, or dyn modifier. If you omit any categories, they will be stacked last, in their default order.
Value.............. mussym, dyn, othertext, chord, lyrics,
Default............ mussym, dyn & othertext & chord, lyrics
Context........... score, staff
Takes effect.... at start of next score
Example......... betweenorder = mussym, lyrics, dyn & othertext, chord
See also........... aboveorder, beloworder, chorddist, dist, dyndist

Name .... bottommargin
Description..... This parameter sets the amount of white space margin to put at the bottom of each page. It is specified in inches if the units parameter is set to inches, or in centimeters if the units parameter is set to cm. This parameter can only be specified before any music or block input. Margins are unaffected by the "scale" parameter. The parameter name can be abbreviated to just "botmargin" if you wish.
Value.............. 0.0 to pageheight minus 0.5 inches
Default............ 0.5 inches
Context........... score
Takes effect.... applies to entire input, and cannot be changed after music input
Example......... bottommargin = 0.8
See also........... leftmargin, rightmargin, topmargin, pageheight, units

Name .... brace
Description..... This parameter specifies which staffs are to be grouped together with a brace to the left of the score. If there is a string given in parentheses, that string will be used as the label to print on the next score, left of the bracket at its vertical center. If there is a second string, that will be used as the label for succeeding scores.
Value............... a comma-separated list of staffs and/or staff ranges, each optionally followed by one or two double-quoted strings enclosed in parentheses. If there are two strings, they are separated by a comma. Staff numbers can range from 1 to the value of the "staffs" parameter. A given staff number can be specified only once, and there can be no overlapping between ranges. Giving no value will result in no braces on any staffs.
Default............ no staffs are grouped by braces.
Context........... score
Takes effect .... immediately for purposes of checking for interactions with other parameters, at start of next score for printing
Examples........ brace = 3-4
brace = 1, 2-3, 4, 5-6
brace = 1-2 ("piano"), 3 ("cello")
brace = 1-2 ("Primo", "I")
brace =   // no braces at all (the default)
See also........... bracket, label, label2, staffs

Name .... bracket
Description..... This parameter specifies which staffs are to be grouped together with a bracket to the left of the score. If there is a string given in parentheses, that string will be used as the label to print on the next score, left of the bracket at its vertical center. If there is a second string, that will be used as the label for succeeding scores.
Value............... a comma-separated list of staffs and/or staff ranges, each optionally followed by one or two double-quoted strings enclosed in parentheses. If there are two strings, they are separated by a comma. Staff numbers can range from 1 to the value of the "staffs" parameter. A bracket range can overlap another bracket range, as long as one range is a proper subset of the other. Giving no value will result in no brackets on any staffs.
Default............ no staffs are grouped by brackets.
Context........... score
Takes effect .... immediately for purposes of checking for interactions with other parameters, at start of next score for printing
Examples........ bracket = 6-7
bracket = 17, 21-23
bracket = 8-9 ("SATB")
bracket = 10-12 ("Strings", "Str")
bracket =   // no brackets at all (the default)
See also........... brace, label, label2, staffs

Name .... bracketrepeats
Description..... When set to "y," brackets are drawn at repeat signs to make it more obvious to the performer where the repeated section begins and ends, which may be helpful when playing in a dimly lit area. Which staffs are bracketed together is controlled by the endingstyle parameter.
Value.............. **y** or **n**
Default........... n
Context.......... score
Takes effect .... at next music context
Example .... bracketrepeats=y
See also.......... endingstyle, repeatdots

Name .... cancelkey
Description..... When set to "y," when a key changes, any sharps or flats in the previous key that are not part of the new key will be canceled by printing natural signs, before printing the new key signature. When set to "n," the naturals will only be printed if the new key has no sharps or flats.
Value.............. y or n
Default........... n
Context.......... score, staff
Takes effect .... at next music context
Example .... cancelkey=y
See also.......... key, useaccs

Name .... carryaccs
Description..... This parameter specifies, for MIDI, whether accidentals follow the normal notation rules of carrying through the remainder of the current measure until explicitly changed. If set to n, each accidental will apply only to the specific chord where it was specified.
Value.............. y or n
Default........... y
Context.......... score, staff
Takes effect .... at next music context
Example .... carryaccs = n
See also.......... useaccs

Name .... chorddist
Description..... This parameter sets the minimum distance from staffs to place chords. When chord marks are printed, they will be placed no closer to the staff than the value of this parameter. This can be used to reduce the ragged effect of having some chord marks much higher than others, because other things were in their way. If a specific chord mark has to be moved farther away than this parameter to avoid running into something, that will still happen, but any others will come out at the level specified by this parameter. This parameter may be overridden on specific items. The section on tempo, dynamic marks, ornaments, etc. gives details on how to do this.
Value.............. a number between 0.0 and 50.0 inclusive, given in stepsizes
Default........... 3.0
Context.......... score, staff
Takes effect .... at next music context
Example .... chorddist = 4
See also.......... dyndist, dist, lyricsdist, scorepad, scoresep

Name .... chordtranslation
Description..... If set to "German" a B in a chord will be printed as H, while a B flat in a chord will be printed as B. If set to a string containing 7 syllables, those syllables will be substituted for pitches in chords, beginning with C. I.e., a C will be replaced by the first syllable, a D by the second syllable, etc.
Value............... "German" or a string like "do re mi fa sol la si" or nothing.
Default............ nothing
Context........... score, staff
Takes effect .... at next music context
Example .... chordtranslation = "DO RE MI FA SOL LA TI"

Name .... clef
Description..... This parameter sets the clef to use. Changing a clef may also change the default octave (see the "defoct" parameter below). However, if clef and defoct are both changed in the same context, the defoct overrides what the clef would have set. If the clef name is followed by a y that means to force printing the clef even if it isn't different from the previous clef.
Value............... treble, treble8, 8treble, frenchviolin, soprano, mezzosoprano, alto, tenor, baritone, bass, 8bass, bass8, or subbass. Can optionally be followed by y, which will force the clef to be printed, even if it is the same as the previous clef. The treble8 clef looks like a treble clef with an 8 below it, and refers to notes that are an octave lower than a normal treble clef. The 8treble clef looks like a treble clef with an 8 above it, and refers to notes that are an octave higher than a normal treble clef. The bass8 clef looks like a bass clef with an 8 below it, and refers to notes that are an octave lower than a normal bass clef. The 8bass clef looks like a bass clef with an 8 above it, and refers to notes that are an octave higher than a normal bass clef. The subbass clef looks like a bass clef, but placed two stepsizes higher on the staff. If the stafflines parameter includes the "drum" keyword, then the value of this clef parameter is only used for determining the placement of notes on the staff, with the drum (or "neutral") clef actually printed. Clef can be changed in the middle of a measure using a construct like <<staff clef=bass>> before a note group.
Default............ treble
Context........... score, staff
It is possible to change this parameter in the middle of a measure, using << >>.
Takes effect .... at next music context, or immediately if mid-measure
Example .... clef = alto
See also........... defoct, stafflines

Name .... cue
Description..... If set to y, all notes are made cue size. This lets you have a long passage of cue notes without having to specific [cue]... over and over.
Value............... y or n
Default............ n
Context........... score, staff, voice
Takes effect .... immediately
Example .... cue=y

Name .... defaultkeymap
Description..... This parameter specifies which keymap to use if none of the other keymap parameters applies. See the section on keymaps in the Text Strings chapter for more details.
Value.............. a string matching the name of a keymap defined earlier, or nothing. An empty string ("") or nothing means don't do any mapping.
Default........... nothing
Context.......... score, staff
Takes effect .... immediately
Examples........ defaultkeymap="Cyrillic"
defaultkeymap=
See also.......... endingkeymap, labelkeymap, lyricskeymap, printkeymap, rehearsalkeymap, textkeymap, withkeymap

Name .... defaultphraseside
Description..... For cases where the side on which to place a phrase mark is not otherwise already determined (and thus Mup would normally choose the side that seemed best), this parameter can be set to force choosing above or below.
Value.............. above, below, or not set
Default........... not set
Context.......... score, staff, voice
Takes effect .... immediately
Examples........ defaultphraseside=above
defaultphraseside= // Let Mup choose

Name .... defoct
Description..... This parameter sets the default octave for any note that does not have an explicit octave specified. An octave goes from C up to the next B, with octave 4 being the octave beginning on middle C. If the clef is changed on a staff, but defoct is not set in that same context, the default octave is changed to match the new clef. defoct can be changed in the middle of a measure using a construct like <<staff defoct=5>> before a note group.
Value.............. a number from 0 to 9 inclusive. Octave 4 is the octave beginning at middle C.
Default........... the octave containing the note represented by the middle line of the staff given the current clef. (Octave 5 for frenchviolin and 8treble; octave 4 for treble, soprano, mezzosoprano, alto and 8bass clefs; octave 3 for treble8, tenor, baritone and bass clefs; octave 2 for bass8 and subbass clefs.)
Context.......... score, staff, voice
Takes effect .... immediately for purposes of checking for interactions with other parameters, at next music context for printing
It is possible to change this parameter in the middle of a measure, using << >>.
Example .... defoct = 3

Name .... dist
Description..... This parameter sets minimum distance from staff to place rom, bold, ital, and boldital items, and rehearsal marks. When these items are printed, they will be placed no closer to the staff than the value of this parameter. This can be used to reduce the ragged effect of having some items much higher than others, because other things were in their way. If a specific item has to be moved farther away than this parameter to avoid running into something, that will still happen, but any others will come out at the level specified by this parameter. If an item is also a chord, the chorddist parameter will be used instead of dist. This parameter may be overridden on specific items. The sections on tempo, dynamic marks, ornaments, etc. and on rehearsal marks give details on how to do this.
Value.............. a number between 0.0 and 50.0 inclusive, given in stepsizes
Default........... 2.0
Context.......... score, staff
Takes effect .... at next music context
Example .... dist = 6
See also........... chorddist, dyndist, lyricsdist, scorepad, scoresep

Name .... division
Description..... This parameter sets the MIDI division (number of clock ticks per quarter note). This typically has a value of 192 or 384. This parameter can only be specified before any music or block input.
Value.............. 1 to 1536
Default........... 192
Context.......... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... division = 384

Name .... dyndist
Description..... This parameter sets minimum distance from staff to place crescendo and decrescendo marks. and text that is marked "dyn." When these items are printed, they will be placed no closer to the staff than the value of this parameter. This can be used to reduce the ragged effect of having some items much higher than others, because other things were in their way. If a specific item has to be moved farther away than this parameter to avoid running into something, that will still happen, but any others will come out at the level specified by this parameter. This parameter may be overridden on specific items. The section on tempo, dynamic marks, ornaments, etc. gives details on how to do this.
Value.............. a number between 0.0 and 50.0 inclusive, given in stepsizes
Default........... 2.0
Context.......... score, staff
Takes effect .... at next music context
Example .... dyndist = 4
See also........... chorddist, dist, lyricsdist, scorepad, scoresep

Name .... emptymeas

Description..... By default, if you don't specify any music input for a given voice, Mup just leaves a measure of space. This parameter lets you specify what Mup should use. The most common value other than space would be a measure of rest, but you can supply any valid music input. So, for example, if your music has a measure-long pattern that repeats frequently in some voice, you could set this parameter to that pattern, and Mup will fill in that music for every measure where you don't override with something else. The value is a string, and is effectively placed in the input as if you had typed it yourself. Note that since errors in the string will generally not be caught until it is actually used, error messages may reference a line a long ways away from the line where the parameter is defined. Note also that since the value is a text string, all the usual rules for text strings apply; for example, any double quotes inside the string must be backslashed. Since the processing of emptymeas happens before the derivation of music on tabnote staffs, you probably do not want to set a tabnote staff to something like emptymeas="mr;" unless the tab staff is also going to be all rests, because that would override the music derivation. Also, when you have set the vscheme to have two or three voices just because there are a few spots in the song that need more than one, you probably only want to set emptymeas="mr;" on voice 1, or you will likely get more rests than you intended.

Value............... a text string (enclosed in double quotes, as usual), containing music input

Default............ ms;

Context........... score, staff, voice

Takes effect .... at next music context

Examples........ emptymeas="mr;"
emptymeas="8.c;16;8.e;16;8.f;16;{8g;f;g;}3;"

Name .... endingkeymap

Description..... This parameter specifies which keymap to use for labels of endings See the section on keymaps in the Text Strings chapter for more details.

Value............... a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.

Default............ nothing

Context........... score, staff

Takes effect .... at next music context

Examples........ endingkeymap="Greek"
endingkeymap=

See also........... defaultkeymap, labelkeymap, lyricskeymap, printkeymap, rehearsalkeymap, textkeymap, withkeymap

Name .... endingstyle
Description..... This parameter controls how first and second endings are placed. This parameter also controls where measure numbers and rehearsal marks are placed, as well as which staffs are bracketed together if the bracketrepeats parameter is set. A value of "top" means that the endings and similar marks will be shown only above the top visible staff. A value of "barred" means these marks will be shown above each set of staffs that is barred together. Each staff that is barred individually will also have the ending shown above it. (See the "barstyle" parameter above.) A value of "grouped" means the marks will be shown above the top visible staff of each range of staffs that are joined by a brace or bracket. In all cases, at least the top visible staff will have endings shown above it.
Value.............. top, barred, or grouped
Default............ top
Context........... score
Takes effect .... at next music context
Example .... endingstyle = grouped
See also........... barstyle, brace, bracket, bracketrepeats, measnum, rehstyle, visible

Name .... extendlyrics
Description..... This parameter specifies whether Mup should automatically add underscore "extender lines" to lyrics. This parameter only has effect when you let Mup derive the time values for lyrics, rather than specifying them explicitly. When the parameter is set to y, and Mup finds a tie or slur in the voice from which lyrics time values are being derived, an underscore will be added to the corresponding syllable, if that syllable doesn't already end with a dash or underscore.
Value.............. y or n
Default............ n
Context........... score, staff, voice
Takes effect .... at next music context
Example .... extendlyrics=y
See also........... sylposition

Name .... firstpage
Description..... This parameter specifies what to number the first page. This value can be overridden by the -p command line option. The number can be optionally followed by "leftpage" or "rightpage" to say whether any header, footer, top, or bottom block on the first page should use the left or right page versions, if those are different. This parameter can only be set before any music or block input.
Value.............. 1 to 5000
Default............ 1 rightpage (unless panelsperpage=2, in which case the page side will be leftpage)
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Examples........ firstpage = 12
firstpage = 1 leftpage
See also........... panelsperpage

Name .... flipmargins
Description..... This parameter specifies if the left and right margins are to be interchanged on every other page. This may be useful if you want extra space for book binding. If set to y, the first physical page will use the values of leftmargin and rightmargin as is, but on the second page, and every other page thereafter, the value for rightmargin will be used for the left margin and the value for leftmargin will be used for the right margin. The settings of the firstpage parameter or the option to print only selected pages have no effect on this parameter.
Value.............. y or n
Default............ n
Context........... score
Takes effect .... at next page
Example .... flipmargins = y
See also........... leftmargin, rightmargin

Name .... font
Description..... This parameter specifies which font to use for print, left, right, center, and title statements.
Value.............. rom, ital, bold, or boldital
Default............ rom
Context........... score, staff, header, footer, header2, footer2, top, bottom, top2, bottom2, block
Takes effect .... immediately
Example .... font = boldital
See also........... fontfamily, lyricsfont, noteleftfont, size, withfont

Name .... fontfamily
Description..... This parameter specifies what font family to use for print, left, right, center, and title statements. It also provides the default for rom, bold, ital, and boldital statements.
Value.............. avantgarde, bookman, courier, helvetica, newcentury, palatino, times
Default............ times
Context........... score, staff, header, footer, header2, footer2, top, bottom, top2, bottom2, block
Takes effect .... immediately
Example .... fontfamily=palatino
See also........... font, lyricsfontfamily, noteleftfontfamily, withfontfamily

Name .... gridfret
Description..... This parameter specifies when to print fret numbers on grids. Normally, the top line of a grid represents the nut. However, if the fingering for a chord is rather far up the neck, it is customary to have the top line of the grid represent some other fret, and print a fret number and "fr" next to the grid, showing the actual fret of the lowest fret mark. This parameter controls when Mup begins using this alternate format. Whenever all the frets of a chord are greater than or equal to the value specified for this parameter, and there are no strings marked "o", the "fr" notation is used. If no value is set for this parameter, the grid will just be made as tall as necessary to accommodate the chord's frets; but in any case, the grid will always be at least as high as the value of the mingridheight parameter.
Value.............. 2 to 99, or not set
Default........... 4
Context.......... score, staff
Takes effect .... at next music context
Examples........ gridfret = 3
gridfret =
See also.......... gridsatend, gridscale, gridswhereused, mingridheight

Name .... gridsatend
Description..... This parameter specifies whether to print guitar grids at the end of the song. If set to "y," grids for all of the chords used in the song will be printed. Grids only associated with particular staffs will only be printed if that staff is visible. Grids associated with "all" will use the score level value of this parameter.
Value.............. y or n
Default........... n
Context.......... score, staff
Takes effect .... During each music context where this has been set to "y," it accumulates chords that are used. If you later set it to n, it stops accumulating, but doesn't forget the ones it accumulated earlier. At the end, it prints out whatever ones it has accumulated, even if the flag is n at that time.
Example .... gridsatend = y
See also.......... gridfret, gridscale, gridswhereused, mingridheight

Name .... gridscale
Description..... This parameter specifies how large to make grids, relative to their default size. For example, a value of 0.5 will make them 1/2 their default size. The default size for grids summarized at the end of the song (the gridsatend parameter) is larger than the default size for those printed with the music (the gridswhereused parameter).
Value.............. 0.1 to 10.0
Default........... 1.0
Context.......... score, staff
Takes effect .... at next music context
Example .... gridscale = 0.5
See also.......... gridsatend, gridfret, gridswhereused, mingridheight, scale, staffscale

Name .... gridswhereused
Description..... This parameter specifies whether to print guitar grids along with chords where they appear in the song. If set to "y," each text item with the chord modifier will have a grid printed below its name.
Value.............. y or n
Default............ n
Context........... score, staff
Takes effect .... at next music context
Example .... gridswhereused = y
See also........... gridfret, gridsatend, gridscale, mingridheight

Name .... indentrestart
Description..... This parameter specifies whether a restart should be indented when it occurs at the beginning of a new score. It does not affect restarts that occur in the middle of a score.
Value.............. y or n
Default............ n
Context........... score
Takes effect .... at next music context
Example .... indentrestart = y

Name .... key
Description..... This parameter sets the key signature. This can be specified either by giving the number of sharps (#) or flats (&), or by giving the name of the key.
Value.............. If using the number of sharps/flats format, the value is a number from 0 to 7, followed by "#" or "&," optionally followed by "major" or "minor." 0& and 0# are equivalent. If using the name of the key, the value is a letter "a" through "g," optionally followed by a "#" or "&," optionally followed by "major" or "minor." The "major" and "minor" can be abbreviated to "maj" or "min." The "major" or "minor" is used for MIDI file purposes, and is optional; if omitted, it defaults to major. If you wish to use a mode other than major or minor, you have to specify the number of sharps or flats.
Default............ c major
Context........... score, staff
Takes effect .... at next music context
Examples........ key = 3&
key = 6#
key = c# minor
key = d major
See also........... addtranspose, cancelkey, carryaccs, stafflines, transpose, useaccs

Name .... label
Description..... This parameter specifies a label to be printed to the left of the staff on the next score. If there is also a brace or bracket label, that label will be to the left of this label. There is a label2 parameter that is used on subsequent scores. In addition to being used for the very first score of a song, this label parameter might be used to clearly mark a change in instrumentation or voices for a particular staff. The label2 would typically be changed at the same time, giving a more abbreviated label. For example, you might set label="Tenor/Bass" and label2="TB"
Value............... a text string enclosed in double quotes
Default............ enough spaces to produce an indent of 1/2 inch
Context........... score, staff
Takes effect .... at start of next score
Example .... label = "oboe"
See also........... alignlabels, brace, bracket, label2

Name label2
Description..... This parameter specifies a label to be printed to the left of the staff on all scores after the first. If there is also a brace or bracket label, that label will be to the left of this label. If both the label and label2 parameters are set at the same time, the label value will be used for the immediately following score, with the label2 value used for subsequent scores. If however, after the first score, only the label2 is changed, then the label2 value will be used on the immediately following score as well as subsequent scores.
Value............... a text string enclosed in double quotes
Default............ no label
Context........... score, staff
Takes effect .... at start of next score
Example .... label2 = "Solo"
See also........... alignlabels, brace, bracket, label

Name .... labelkeymap
Description..... This parameter specifies which keymap to use for labels to the left of staffs. See the section on keymaps in the Text Strings chapter for more details.
Value............... a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default............ nothing
Context........... score, staff
Takes effect .... immediately
Examples........ labelkeymap="Cyrillic"
labelkeymap=
See also........... defaultkeymap, endingkeymap, lyricskeymap, printkeymap, rehearsalkeymap, textkeymap, withkeymap

Name .... leftmargin
Description..... This parameter sets the amount of white space margin to put at the left side of each page. It is specified in inches if the units parameter is set to inches, or in centimeters if the units parameter is set to cm. This parameter can only be specified before any music or block input. Margins are unaffected by the "scale" parameter. You can use the flipmargins parameter to adjust alternating pages to allow room for book binding.
Value.............. 0.0 to pagewidth minus 0.5 inches
Default............ 0.5 inches
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... leftmargin = 0.3
See also........... bottommargin, flipmargins, rightmargin, topmargin, pagewidth, units

Name .... leftspace
Description..... This parameter specifies what portion of the white space around a chord is placed on its left side. Usually (unless packexp is zero), longer notes will have more white space around them than shorter notes, which means that the white space to their left is also larger. The parameter allows you to control how pronounced that effect is, or even specify to put all the space on the right. Two numbers must be specified, separated by a comma. They may include decimal fraction parts. The first specifies what portion of the white space should go on the left. The second specifies a maximum amount of white space, in stepsizes. So if the amount calculated by the first number is greater than the second number, the second number will be used.
Value.............. first number 0.0 to 0.5 and second number 0.0 to 100.0
Default............ 0.15, 5.0
Context........... score
Takes effect .... at next music context
Example .... leftspace = 0.2, 4
See also........... lyricsalign, packexp, packfact

Name .... lyricsalign
Description..... This parameter specifies how to align lyric syllables with chords. Its value is the proportion of each syllable to place to the left of the syllable’s chord. Thus, for example, a value of 0.0 causes the left edge of syllables to be aligned with the chords, whereas a value of 0.5 causes syllables to be centered with the chord, and 1.0 causes the right edge of the syllables to be aligned with the chord.
Value.............. 0.0 to 1.0
Default............ 0.25
Context........... score, staff
Takes effect .... at next music context
Example .... lyricsalign = 0.1
See also........... sylposition

Name .... lyricsdist
Description..... This parameter sets the minimum distance from staffs to place lyrics. When lyrics are printed, they will be placed no closer to the staff than the value of this parameter. This only affects lyrics above and below, not between.
Value.............. a number between 0.0 and 50.0 inclusive, given in stepsizes.
Default........... 2.0
Context.......... score, staff
Takes effect .... at start of next score
Example .... lyricsdist = 4
See also.......... chorddist, dyndist, dist, scorepad, scoresep

Name .... lyricsfont
Description..... This parameter sets which font to use for lyrics.
Value.............. rom, ital, bold, boldital
Default........... rom
Context.......... score, staff
Takes effect .... immediately
Example .... lyricsfont = ital
See also.......... font, lyricssize

Name .... lyricsfontfamily
Description..... This parameter specifies what font family to use for lyrics.
Value.............. avantgarde, bookman, courier, helvetica, newcentury, palatino, times
Default........... times
Context.......... score, staff
Takes effect .... immediately
Example .... lyricsfontfamily=helvetica
See also.......... font, fontfamily, lyricsfont

Name .... lyricskeymap
Description..... This parameter specifies which keymap to use for lyrics. See the section on keymaps in the Text Strings chapter for more details.
Value.............. a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default........... nothing
Context.......... score, staff
Takes effect .... at next music context
Examples........ lyricskeymap="Cyrillic"
lyricskeymap=
See also.......... defaultkeymap, endingkeymap, labelkeymap, printkeymap, rehearsalkeymap, textkeymap, withkeymap

Name .... lyricssize
Description..... This parameter sets point size to use for lyrics.
Value.............. a number from 1 to 100
Default........... 12
Context.......... score, staff
Takes effect .... at next music context
Example .... lyricssize = 10
See also.......... lyricsfont, lyricsfontfamily, size

Name .... maxmeasures
Description..... This parameter specifies the maximum number of measures to put on any score.
Value.............. a number from 1 to 1000
Default........... 1000
Context.......... score
Takes effect .... at start of next score
Example .... maxmeasures = 3
See also.......... leftmargin, maxscores, packexp, packfact, rightmargin, scale, staffscale

Name .... maxscores
Description..... This parameter specifies the maximum number of scores to print per page.
Value.............. 1 to 1000
Default........... 1000
Context.......... score
Takes effect .... at next music context
Example .... maxscores=2
See also.......... scorepad, scoresep

Name .... measnum
Description..... This parameter specifies whether or not to print measure numbers. If set to "y," the current measure number will be printed at the beginning of each score other than the first. The number will be printed above any scores that would receive ending marks (see "endingstyle" parameter). If set to "n," no measure numbers will be printed. If set to "every *N*," measure numbers will be printed on every Nth bar line unless some other rehearsal mark has been specified on that bar.
Value.............. y, n, or every N, where N can be from 1 to 10000
Default........... n
Context.......... score
Takes effect .... at next music context if "every N," else start of next score
Examples........ measnum = y
measnum = every 1 // number all measures
measnum = every 5
See also.......... endingstyle, measnumfont, measnumfontfamily, measnumsize, measnumstyle, rehstyle

Name .... measnumfont
Description..... This parameter specifies which font type to use for the automatic measure numbers, if they are turned on via the measnum parameter.
Value.............. rom, ital, bold, or boldital
Default........... rom
Context.......... score
Takes effect .... at next music context if measnum parameter is "every N," else start of next score
Example .... measnumfont=boldital
See also.......... measnum, measnumfontfamily, measnumsize, measnumstyle

Name .... measnumfontfamily
Description..... This parameter specifies which font family to use for the automatic measure numbers, if they are turned on via the measnum parameter.
Value.............. avantegarde, bookman, courier, helvetica, newcentry, palatino, or times
Default........... times
Context.......... score
Takes effect .... at next music context if measnum parameter is "every N," else start of next score
Example .... measnumfontfamily=helvetica
See also.......... measnum, measnumfont, measnumsize, measnumstyle

Name .... measnumsize
Description..... This parameter specifies what size to use for the automatic measure numbers, in points, if they are turned on via the measnum parameter.
Value.............. 1 to 100
Default........... 11
Context.......... score
Takes effect .... at next music context if measnum parameter is "every N," else start of next score
Example .... measnumsize=15
See also.......... measnum, measnumfont, measnumfontfamily, measnumstyle

Name .... measnumstyle
Description..... This parameter specifies whether to put automatic measure numbers inside boxes or circles, or leave them plain.
Value.............. plain, boxed, or circled
Default........... plain
Context.......... score
Takes effect .... at next music context if measnum parameter is "every N," else start of next score
Example .... measnumstyle=boxed
See also.......... measnum, measnumfont, measnumfontfamily, measnumsize, rehstyle

Name .... midlinestemfloat
Description..... This parameter controls the stem direction of chords whose notes are centered on the middle line of the staff, for cases where it is free to point in either direction. It is not free (and thus midlinestemfloat does not apply) when the direction is forced by the user, or by the vscheme, or another voice, or for chords that are part of a beamed set of chords, or for grace notes or for voice 3. When midlinestemfloat applies, if it is set to n, the stem will always be down. When set to y, the stem will be up in the case where the neighboring chords on either side, if any, have stems up.
Value.............. y or n
Default........... n
Context.......... score, staff, voice
Takes effect .... immediately
Example .... midlinestemfloat=y

Name .... minalignscale
Description..... This parameter specifies how much aligned strings can be compressed, in an effort to prevent them from running in the next item at the same alignment level. Only aligned things are ever compressed, and then only if they would collide with the the next item at the same level, and only as much as needed to make them fit, up the the limit of this parameter. Multiplying this value by the normal width of an item will tell the shortest width Mup will compress to before giving up and printing a warning. A value of 1.0 means nothing will ever be compressed at all.
Value.............. 0.1 to 1.0
Default........... 0.667
Context.......... score, staff
Takes effect .... at start of next score
Example .... minalignscale = 0.85

Name .... mingridheight
Description..... This parameter specifies the minimum number of frets to print on grids. Grids will be made taller than this when necessary, but will never be shorter than this.
Value.............. A number from 2 to 99
Default........... 4
Context.......... score, staff
Takes effect .... at next music context
Example .... mingridheight=6
See also........... gridfret, gridsatend, gridscale, gridswhereused

Name .... musicscale
Description..... This parameter specifies by what factor to scale printed output, but unlike the scale parameter (which applies to the entire page), this parameter applies only to the "music" portion of pages, and does not affect headers, footers, top or bottom blocks. As examples, musicscale=2 prints everything in the music area of the pages twice as large as normal, while musicscale=0.5 prints it at half size. This parameter can only be specified before any music or block input.
Value.............. a number between 0.1 and 10.0
Default........... 1.0
Context.......... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... musicscale=0.95
See also........... packfact, packexp, scale, staffscale

Name .... noteheads
Description..... This parameter describes which notehead shape(s) to use for each pitch in the scale. If you want to use the same shape for all pitches, as is the case with standard notation, the value is a string containing a single shape name (e.g., "norm" for standard notation). If you want to use different shapes for different pitches, the noteheads parameter value is a string containing a list of 7 shape names. They are listed in order starting from the "tonic" of the major key as indicated by the number of sharps or flats in the key signature. There are pre-defined head shapes: norm, x, allx, diam, blank, righttri, isostri, rect, pie, semicirc, slash, and allslash. Additional head shapes can be defined in the headshapes context. See the examples below for the most common settings for this parameter. Head shape can be overridden on an individual chord by using [hs "shapename"] before the chord. It can also be overridden on an individual note by putting hs "shapename" after the note.
Value.............. a string containing either 1 or 7 head shape names
Default........... "norm"
Context........... score, staff, voice
Takes effect .... at next music context
Examples........ // This is the setting for the most common shaped note system using 4 shapes.
noteheads = "righttri norm rect righttri norm rect diam"

// This is the setting for a shaped notes system that uses 7 different shapes.
noteheads = "isostri semicirc diam righttri norm rect pie"
See also........... shapes

Name .... noteinputdir
Description..... If set to "any," then when there are multiple notes in a chord, the notes can be entered in any order, but any notes not in the default octave must have their octave specified, either by octave number, or the appropriate number of plus or minus signs, based on their distance from the default octave. When set to "up," notes in a chord must be entered in pitch order from bottom up. When set to "down," they must be entered in pitch order from top down. When in up or down mode, the first note's octave is specified as with any, but subsequent notes in the chord are relative to the previous note. Note letters less than or equal to an octave away from their preceding note's letter have no octave specified. For notes an octave or more away, in up mode, only plus signs can be used, and in down mode, only minus signs can be used, to specify how many octaves away they are. For example, for up mode, default octave of 4, and input of gbe, the b will be in octave 4, but the e will be in octave 5, because that is the next e that is upward from b4. In down mode with default octave of 4, and the same input of gbe, the b and e would both be in octave 3. This parameter is ignored on tablature staff input and when using chord-at-a-time input mode.
Value.............. up, down, or any
Default........... any
Context........... score, staff, voice
Takes effect .... at next music context
Example .... noteinputdir=up

Name .... noteleftfont
Description..... This controls the font used for strings to be printed to the left of notes, typically to show fingering.
Value.............. rom, bold, ital, or bodlital
Default........... rom
Context........... score, staff, voice
Takes effect .... at next music context
Example .... noteleftfont = ital
See also........... font, noteleftfontfamily, noteleftsize

Name .... noteleftfontfamily
Description..... This controls the font family used for strings to be printed to the left of notes, typically to show fingering.
Value.............. avantgarde, bookman, courier, helvetica, newcentury, palatino, times
Default........... newcentury
Context.......... score, staff, voice
Takes effect .... at next music context
Example .... noteleftfontfamily = helvetica
See also.......... fontfamily, noteleftfont, noteleftsize

Name .... noteleftsize
Description..... This controls the font size used for strings to be printed to the left of notes, typically to show fingering.
Value.............. 1 to 100
Default........... 10
Context.......... score, staff, voice
Takes effect .... at next music context
Example .... noteleftsize = 5
See also.......... fontfamily, noteleftfont, noteleftfontfamily, size

Name .... numbermrpt
Description..... If set to "y" measure repeats are numbered; if set to "n" they aren't.
Value.............. y or n
Default........... y
Context.......... score, staff
Takes effect .... at next music context
Example .... numbermrpt = n

Name .... numbermultrpt
Description..... If set to "y" dbl and quad measure repeats are numbered with "2" or "4" respectively, printed above the middle bar line of the group; if set to "n" they aren't. The number will be printed in the same font as a time signature.
Value.............. y or n
Default........... y
Context.......... score, staff
Takes effect .... at next music context
Example .... numbermulrpt = n

Name .... ontheline
Description..... This parameter specifies whether notes for voices 1 and 2 on a 1-line staff are to be placed on the line. If this is set to n, notes with stem up will be placed above the line and notes with stem down will be placed below the line; otherwise both will be placed on the line. For notes that don't have a stem, the rules are applied using the direction the stem would be if there were a stem. This parameter has no effect on 5-line staffs or tablature staffs. Notes for voice 3 are always placed on the line on 1-line staffs, regardless of the value of this parameter.
Value............... y or n
Default............ y
Context........... score, staff, voice
Takes effect .... at next music context
Example .... ontheline=n
See also........... stafflines

Name .... packexp
Description..... This parameter sets note expansion factor. This factor controls spacing of notes relative to their time values. If set to 1.0, Mup will try to give a half note twice as much space as a quarter note, a whole note twice as much as a half note, etc. If set to 0.0, a chord's time value will have no impact on its placement. Intermediate values will cause relative spacing between the two extremes. Note that individual chords may get more space than they would theoretically "deserve" if they happen to need extra space to accommodate accidentals, dots, etc.
Value............... a number from 0.0 to 1.0 inclusive
Default............ 0.8
Context........... score
Takes effect .... at next music context
Example .... packexp = 0.95
See also........... packfact, pad

Name .... packfact
Description..... This parameter specifies how tightly to pack notes together on output. The smaller the value, the more tightly notes are packed together.
Value............... a number from 0.0 to 10.0
Default............ 1.0
Context........... score
Takes effect .... at start of next score
Example .... packfact = 1.4
See also........... packexp, pad

Name .... pad
Description..... This parameter specifies the amount of padding to be added to notes. This can be used to control how tightly things are packed together. Especially if packexp and packfact are very small, notes can get placed very close together. This parameter can be used to always force a minimum amount of space between horizontally adjacent note groups. A value of zero means notes will be allowed to just touch. More positive values cause more space around notes. A negative value will let things actually overlap, so most people will probably never want to use a negative value, but the option is there if you want to do something unusual. This parameter works somewhat like the "pad" value that can be specified for individual note groups, except that it applies to all groups.
Value............... a number of stepsizes, -5.0 to 50.0
Default............ 0.3333
Context........... score, staff, voice
Takes effect .... at next music context
Example .... pad = 1.76
See also........... packexp, packfact

Name .... pageheight
Description..... This parameter sets the page height. If the units parameter is inches, the value of pageheight is given in inches, or if the units parameter is cm, it is given in centimeters. This parameter can only be specified before any music or block input. If the pagewidth and pageheight parameters are set to values that match a standard paper size in landscape mode, the Mup output will be rotated to print properly in landscape mode. If pageheight and pagesize are both set in the same context, whichever is specified last will override the previous.
Value............... 2.0 to 24.0 inches or 5.0 to 61.0 cm
Default............ 11.0 inches
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... pageheight = 9
See also........... pagewidth, bottommargin topmargin, units

Name .... pagesize
Description..... This parameter sets the page size. This is just an alternate way of specifying pageheight and pagewidth using the common names for paper sizes rather than specifying in inches or cm. An orientation (portrait or landscape) can also be specified; the default is portrait. This parameter can only be specified before any music or block input. If pageheight and/or pagewidth are set in the same context as pagesize, whichever is specified last will override the previous.
Value............... letter, legal, flsa, halfletter, note, a3, a4, a5, a6; optionally followed by portrait or landscape.
Default............ letter
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Examples........ pagesize = a4
pagesize = legal landscape
See also........... pageheight, pagewidth, panelsperpage

Name .... pagewidth
Description..... This parameter sets the page width. If the units parameter is inches, the value of pagewidth is given in inches, or if the units parameter is cm, it is given in centimeters. This parameter can only be specified before any music or block input. If the pagewidth and pageheight parameters are set to values that match a standard paper size in landscape mode, the Mup output will be rotated to print properly in landscape mode. If pagewidth and pagesize are both set in the same context, whichever is specified last will override the previous.
Value.............. 2.0 to 24.0 inches or 5.0 to 61.0 cm
Default............ 8.5 inches
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... pagewidth = 6.5
See also........... pageheight, flipmargins, leftmargin, rightmargin, units

Name .... panelsperpage
Description..... This parameter specifies how many pages of music to print on each physical page. This parameter can only be specified before any music or block input. Note that the pageheight and pagewidth parameters still apply to the physical paper size viewed in portrait mode, even when the panelsperpage value causes the printing to be landscape mode, so you should continue to leave those set as you normally would. The -o command line option may be useful for getting pages printed in desired order. For example, to make a 4-page booklet from a single sheet of paper folded in half, you can use panelsperpage=2, then use -o4,1 to print one side of the paper, and -o2,3 to print the other side. When panelsperpage is 2, the first page side is always left.
Value.............. 1 or 2
Default............ 1
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... panelsperpage=2
See also........... firstpage, pageheight, pagewidth

Name .... pedstyle
Description..... This parameter specifies whether to display piano pedal marks with lines or with the word "Ped" and "*". With the "pedstar" style, a "bounce" of the pedal is shown by a "* Ped," whereas with the "alt pedstar" style, only a "Ped" is printed.
Value.............. line, pedstar, or alt pedstar
Default............ line
Context........... score, staff
Takes effect .... at next music context
Example .... pedstyle = pedstar

Name .... pilescale
Description..... This parameter defines by what factor to scale text when it is piled.
Value.............. 0.1 to 10.0
Default............ 0.65
Context........... score
Takes effect .... at next music context
Example .... pilescale = 0.8

Name .... printedtime
Description..... This parameter defines what to print as a key signature, if you want that to be something other than the actual time signature.
Value............... There are three possible formats for the value. The first is just like for the "time" parameter, except that alternating time is not allowed. An example usage might be if most staffs are in 3/4 time, but one is really in 6/8, so you could set that staff's printedtime parameter to 6/8. Another example would be to set the actual time signature to 7/4, but set individual staffs to 3/4+4/4 or 4/4+3/4 as appropriate. The second format is a single string. One usage might be to set it to "3" and assume the reader can deduce what the time unit is. The third format is two strings, which will be centered one on top of the other. A possible usage would be to put a "3" for the numerator, and an actual note symbol for the "denominator." It is also possible to set to nothing, to revert to using the actual time signature. Default size and font are set to match what normal time signatures would be; you can change those inside the string(s) if you wish.
Default............ nothing
Context........... score, staff
Takes effect .... at next music context
Examples........ printedtime = 3/4
printedtime = 3/4+4/4
printedtime = "3"
printedtime = "4" "\(dn4n)"
printedtime =   // revert to actual time signature
See also........... time

Name .... printkeymap
Description..... This parameter specifies which keymap to use for print, left, right, center, and title commands. See the section on keymaps in the Text Strings chapter for more details.
Value............... a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default............ nothing
Context........... score, block
Takes effect .... immediately
Examples........ printkeymap="Cyrillic"
printkeymap=
See also........... defaultkeymap, endingkeymap, labelkeymap, lyricskeymap, rehearsalkeymap, textkeymap, withkeymap

Name .... printmultnum
Description..... If set to "y," multirests are labeled with the number of measures of rest they represent; if set to "n," they aren't. This would allow you to print some other commentary in place of the number, print it in a different style, etc.
Value............... y or n
Default............ y
Context........... score, staff
Takes effect .... at next music context
Example .... printmultnum = n

Name .... rehearsalkeymap
Description..... This parameter specifies which keymap to use for rehearsal mark strings. This does not apply to lettered or numbered rehersal marks. See the section on keymaps in the Text Strings chapter for more details.
Value............... a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default........... nothing
Context.......... score, staff
Takes effect .... at next music context
Examples........ rehearsalkeymap="Cyrillic"
rehearsalkeymap=
See also.......... defaultkeymap, endingkeymap, labelkeymap, lyricskeymap, printkeymap, textkeymap, withkeymap

Name .... rehstyle
Description..... This parameter specifies whether to enclose rehearsal marks inside box, inside a circle, or just as plain text.
Value............... boxed, circled, or plain
Default........... boxed
Context.......... score, staff
Takes effect .... at next music context
Example .... rehstyle = circled
See also.......... endingstyle

Name .... release
Description..... This parameter specifies how soon (in milliseconds) before the full time value of note to release the note when generating MIDI output. This controls how legato (smooth) the music is. A value of 0 will make it very legato. The larger the value, the more detached notes will be. This parameter specifies a maximum amount to shorten notes; a note will never be shortened to less than 75% of its full value, unless it has a dot or wedge on it, it which case the shortening can be a maximum of half or 2/3 respectively. The release value can be changed in the middle of a measure, using a construct like <<score release=50>> before a note group.
Value............... 0 to 500
Default........... 20
Context.......... score, staff, voice
Takes effect .... at next music context, or immediately if mid-measure
It is possible to change this parameter in the middle of a measure, using << >>.
Example .... release = 40

Name .... repeatdots
Description..... If set to "standard," repeat signs are printed using the standard convention of two dots. If set to "all," repeat signs are printed with dots between all the lines of the staff.
Value............... standard or all
Default........... standard
Context.......... score, staff
Takes effect .... at next music context
Example .... repeatdots = all
See also.......... bracketrepeats

Name .... restcombine
Description..... If the given number of measures of rest occur in a row, they will be replaced by a multirest. This parameter can be overridden by the -c command line option. See the description of the -c option for more complete information on how the combining is done.
Value............... 2 to 1000 or nothing
Default............ not set
Context........... score
Takes effect .... whatever the value is at the end of a series of rest measures is what is used
Examples........ restcombine = 5
restcombine =   // turn off combining

Name .... restsymmult
Description..... This parameter specifies how to print multirests. Multirests are normally drawn as a horizontal line on the middle line of the staff, with two vertical lines at the end. But there is an alternate notation style that uses rest symbols (whole, double whole, and quad whole) when the number of measures is short. If this parameter is set to "y," that alternate style will be used for multirests of eight measures or less.
Value............... y or n
Default............ n
Context........... score, staff
Takes effect .... at next multirest. When used with restcombine, if you set this after the first rest measure, it has no effect
Example .... restsymmult = y
See also........... printmultnum

Name .... rightmargin
Description..... This parameter sets the amount of white space margin to put at the right side of each page. It is specified in inches if the units parameter is set to inches, or in centimeters if the units parameter is set to cm. This parameter can only be specified before any music or block input. Margins are unaffected by the "scale" parameter. You can use the flipmargins parameter to adjust alternating pages to allow room for book binding.
Value............... 0.0 to pagewidth minus 0.5 inches
Default............ 0.5 inches
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... rightmargin = 0.3
See also........... bottommargin, flipmargins, leftmargin, topmargin, pagewidth, units

Name .... scale
Description..... This parameter specifies by what factor to scale the printed output. For example, scale=2 prints everything twice as large as normal, while scale=0.5 prints everything at half size. This parameter can only be specified before any music or block input.
Value............... a number between 0.1 and 10.0
Default............ 1.0
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... scale=0.95
See also........... musicscale, packfact, packexp, staffscale

Name .... scorepad
Description..... This parameter sets the amount of padding (white space) to leave between scores, accounting for all the things that protrude from both scores. Either a single number, giving a minimum amount, or two numbers, giving a minimum and maximum, can be specified. They are specified in stepsizes. If only the minimum is specified, and it is larger than the default maximum of 2.0, the maximum will be adjusted to equal the minimum. Depending on the setting of the scoresep parameter, the maximum may be exceeded; see the description of scoresep for how these parameters interact to determine the placement of the scores. If a negative value is specified for scorepad, some overlap may occur (subject to the interaction with scoresep). Specifying a negative value may be particularly useful when things protrude downward from the top score and upward from the bottom score, but at different places horizontally, such that it is actually safe to put the scores closer together without collision, even though Mup can't tell that it is safe. Note, however, that this overrides Mup's protection against real collisions, so this must be used with care to avoid undesired overlaps.
Value.............. one or two numbers, in the range from negative the height of the page and the height of a page, in stepsizes. If there are two numbers, they are separated by a comma, and the second must be greater than or equal to the first. This parameter also applies to blocks, but is simpler in that case, since nothing can actually protrude from a block.
Default............ 2.0, 2.0
Context........... score
Takes effect .... between the current score and the next score
Examples........ scorepad = 5
scorepad = -1
scorepad = 3.5,10.75
See also........... scoresep, staffpad, staffsep

Name .... scoresep

Description..... This parameter sets how much space to leave between scores; i.e., between the bottom line of the bottom staff of one score and the top line of the top staff of the following score. Either a single number, giving a minimum amount, or two numbers, giving a minimum and maximum, can be specified. They are specified in stepsizes. If only the minimum is specified, and it is larger than the default maximum of 20.0, the maximum will be adjusted to equal the minimum. Depending on the setting of the scorepad parameter, the maximum may be exceeded. The parameters interact as follows in determining the layout of a page: As many scores are allocated to the page as will fit (or until a "newpage" command is encountered). Initially, they are packed together as tightly as they can be without violating the minimum values of scorepad and scoresep between any neighboring scores. Next, if there is extra space available at the bottom of the page, the scores are spread out, increasing the white space between them, but not increasing any beyond the maximum scorepad value. (Some may, however, already be beyond the maximum scorepad value, because the minimum scoresep value required it.) This spreading is done without regard for the maximum scoresep value. If any of the inter-score gaps start narrower than others (because of the minimum scoresep), they are increased first, in an attempt to even out the differences. If the maximum scorepad value is reached or exceeded between all the scores, and there is still extra space available at the bottom of the page, then the scores are spread out some more, increasing the white space between them, this time ignoring the scorepad values, but not increasing any beyond the maximum scoresep value. (Some may however already be beyond the maximum scoresep value, because of the previous steps.) If still not all the space is used up, it remains as extra space at the bottom of the page. This parameter also applies to blocks, but is simpler in that case, since nothing can actually protrude from a block.

Value............... one or two numbers, in the range from 6.0 to the height of the page in stepsizes. If there are two numbers, they are separated by a comma, and the second must be greater than or equal to the first.

Default............ 12.0, 20.0

Context........... score

Takes effect .... between the current score and the next score

Examples........ scoresep = 25
scoresep = 9.6, 15.3

See also........... scorepad, staffpad, staffsep

Name .... shapes

Description..... This parameter specifies the name of a shapes context that defines music symbol overrides. This allows using alternate symbols on only certains staffs, or for some symbols, only a specific voice.

Value............... A string which matches the name given for a shapes context, or nothing, to turn off shapes overrides.

Default............ Not set

Context........... score, staff, voice

Takes effect .... at next music context

Example .... shapes="myshapes"
shapes=   // to stop overriding

See also........... noteheads

Name .... size
Description..... This parameter specifies what point size to use for text in print, title, left, right, and center statements. It also provides the default for rom, bold, ital, and boldital statements.
Value.............. a number from 1 to 100 inclusive
Default........... 12
Context.......... score, staff, header, footer, header2, footer2, top, bottom, top2, bottom2, block
Takes effect .... immediately
Example .... size = 9
See also.......... font, fontfamily, lyricssize, noteleftsize, measnumsize, withsize

Name .... slashesbetween
Description..... This parameter specifies whether to put two thick slanted lines between scores at the left edge of the staffs. These are often used when there are a lot of staffs, or when the number of scores per page varies, to help the musicians see clearly where the next score begins.
Value.............. y or n
Default........... n
Context.......... score
Takes effect .... between the current score and the next score
Example .... slashesbetween=y

Name .... stafflines

Description..... This parameter specifies how many lines to draw for the staff. Normally, there are 5 lines per staff, but a single-line staff is sometimes used for percussion, and tablature staffs for various instruments may have different numbers of lines. Setting this parameter to 1 will produce a single line staff. The number of lines can be followed by "n" to indicate that clef and key signature are not to be printed. The "n" also implies that accidentals are to be ignored and that notes are never to be transposed. If the number of lines is 1, the clef and key signature are never printed, regardless of whether you add the "n," so the "n" is really only meaningful when used with 5. When stafflines=1, you can only have one note per chord, and the pitch of that note is irrelevant, except for MIDI output. Alternately, rather than specifying "n," you can specify "drum," which means to use the drum clef (also sometimes called the "neutral" clef). With the drum clef, no key signature is printed, accidentals are ignored, and notes are never transposed. The value used for the clef parameter is used to determine the pitch for placement of notes in this case, but the drum clef of two vertical lines is printed.

For a tablature staff, rather than specifying a number of staff lines as the value, the keyword "tab" is used, optionally followed by a list of strings in parentheses. The strings are listed in order from the top line of the tablature staff to the bottom. Each item in the list has at least a string pitch, which is a letter from a to g, optionally followed by # or &. If there is more than one string having the same letter/accidental, they are distinguished by adding one or more single quote marks ("ticks"). An octave number can also be specified. If the list of strings is omitted, standard guitar strings are used, which is tab(e5 b4 g4 d4 a3 e'3). Tablature can only be specified in staff context, not score or voice, and when a tablature staff is specified, the staff above it becomes a "tabnote" staff, which is a normal 5-line staff containing music derived from the tablature staff. The list of strings, if any, can optionally be followed by y or n. Using y will cause the word TAB to be printed vertically at the beginning of every score. Using n will cause that to never to be printed. Not specifying either will cause it to be printed only on the very first staff, which is the most common way of printing tablature.

Value............... 1 or 5, optionally followed by "n" or "drum"; or for tablature staff, the keyword "tab" optionally followed by a list of strings, in parentheses, optionally followed by y or n. Setting the stafflines parameter will also reinitialize other parameters: key, transpose, addtranspose, clef, beamstyle, and defoct.

Default............ 5

Context........... score, staff

Takes effect .... immediately for purposes of checking for interactions with other parameters. Forces new score if the number part of it changed. In any case, the "n" part takes effect at the next score.

Examples........ stafflines=1
stafflines=5n
stafflines = tab // standard guitar tablature staff
stafflines = tab (g3 d3 a2 e2) // standard bass guitar
stafflines = tab (d# g b3 g'3)
stafflines = tab y// print TAB "clef" on every score

See also........... clef, key, transpose, addtranspose

Name .... staffpad
Description..... This parameter sets the minimum amount of space to leave between staffs, accounting for all the things that protrude from both staffs. If a negative value is specified, some overlap may occur, although it will still be limited by the value of the staffsep parameter. Specifying a negative value may be particularly useful when things protrude downward from the top staff and upward from the bottom staff, but at different places horizontally, such that it is actually safe to put the staffs closer together without collision, even though Mup can't tell that it is safe. Note, however, that this overrides Mup's protection against real collisions, so this must be used with care to avoid undesired overlaps. If this parameter is set in staff context for staff N, it affects the distance from staff N to staff N+1.
Value.............. a number between negative the height of the page and the height of a page, in stepsizes
Default........... 0
Context.......... score, staff
Takes effect .... at start of next score
Example .... staffpad = -2
staffpad = 1.6
See also.......... scorepad, scoresep, staffsep

Name .... staffs
Description..... This parameter specifies the number of staffs. It is possible that not all of these staffs will be printed (see the "visible" parameter below and the -s command line argument). Changing the number of staffs causes all parameters that had been set in staff and voice context to be set back to their default values. It is usually preferable to only set the staff parameter once at the beginning of a song, and use the "visible" parameter when you want to change which staffs are actually printed, rather than changing the number of staffs.
Value.............. a number between 1 and 40 inclusive.
Default........... 1
Context.......... score
Takes effect .... forces new score if it changed
Example .... staffs = 12
See also.......... visible

Name .... staffscale
Description..... This parameter specifies how to scale the size of a staff relative to the size of other staffs. A value of 1.0 yields the normal size, whereas 0.5 yields a staff that is half as high, and 2.0 one that is twice as high as normal, and so forth. This might be used, for example, for a piece written for two instruments, say piano and violin, where you want the piano part to be written in normal size, but want to show the violin part in smaller size, such that while the pianist will have the violin part available for reference, it won't take up a lot of space. While it is possible to set staffscale in score context, to make all staff-related things a different size, in most cases it will be better to use the musicscale parameter for that.
Value.............. 0.1 to 10.0
Default........... 1.0
Context.......... score, staff
Takes effect .... forces new score if it changed
Example .... staffscale=0.75
See also.......... musicscale, scale

Name .... staffsep
Description..... This parameter specifies the minimum amount of space to leave between any two adjacent staffs within the same score. It is specified in stepsizes, and is measured from the bottom line of the staff above to the top line of the staff below. Staffs will be spread wider than this minimum if necessary to prevent things from colliding. If this parameter is set in staff context for staff N, it affects the distance from staff N to staff N+1.
Value............... a number from 6.0 to the height of the page in stepsizes
Default............ 10
Context........... score, staff
Takes effect .... at start of next score
Example .... staffsep = 14
staffsep = 17.8
See also........... scorepad, scoresep, staffpad

Name .... stemlen
Description..... This parameter specifies how long stems should be, in stepsizes. This is for normal-sized chords; grace or cue size chords will gets stems that are 5/7 of this length. This length can be overridden on specific chords, using the len attribute in brackets before the chord. Stem lengths can also be affected by the stemshorten parameter.
Value............... 0.0 to 100.0
Default............ 7.0
Context........... score, staff, voice
Takes effect .... at next music context
Example .... stemlen = 0 // to make all notes stemless
See also........... stemshorten

Name .... stemshorten
Description..... There are several circumstances in which Mup normally shortens stems slightly in an attempt to improve appearance. This parameter will accept one, two, or four values to control how stems are affected in those cases. On beamed chords, Mup will sometimes shorten stems slightly. This first value of this parameter lets you control the maximum amount of shortening that will ever be done on beamed notes. It is specified in stepsizes. The remaining three (optional) values control how Mup shortens stems that protrude from the staff. Most publishers of music shorten such stems somewhat, but there is some inconsistency in exactly how much. The second value to this parameter specifies the maximum amount to shorten any protruding stem, in stepsizes. The third and fourth specify at what point to begin shortening and at what point to reach the maximum shortening. These are specified in number of stepsizes from the middle line of the staff. Note that Mup will still lengthen stems from this value if necessary to accommodate things like dots or flags. By default, Mup will use full-length stems (normally 7.0 stepsizes) for any stem-up note at or below the middle line of the staff, and for any stem-down note at or above the middle line of the staff. Beyond there, it will gradually shorten stems until they get down to 5.0 stepsizes in length (2.0 stepsizes worth of shortening) for notes 6 stepsizes or more away from the middle line. So using a treble clef staff as an example, stem-up notes b or lower will normally get stems 7.0 stepsizes long (or even longer if necessary), but the c right above there will be a little shorter, and so on until the a on the first ledger line above the staff is the maximum shortening of 2.0 stepsizes. You can always override on individual chords using [len=N] to force a particular length.
Value............... 0.0 to 2.0 for maximum beam shortening, 0.0 to 7.0 for maximum shortening of stems that protrude from the staff, and -4 to 50 for where to begin and end shortening of protruding stems
Default............ 1.0, 2.0, 1, 6
Context........... score, staff, voice
Takes effect .... at next music context
Examples........

```
stemshorten = 0     // never shorten any beamed stems
stemshorten = 0, 0// never shorten any stems
// For beams, allow shortening up to 1.5 stepsizes.
// For protruding, start shortening 4 stepsizes from the
// middle line (i.e., f+ assuming treble clef),
// and reach maximum shortening of 3.5 stepsizes
// at 12 stepsizes from the middle line (g++).
stemshorten = 1.5, 3.5, 4, 12
```

See also........... stemlen

Name .... subbarstyle

Description..... This parameter controls where subdivisions of bars, if any, are drawn and how they look. You might consider using this instead of or in connection with an additive time signature.

Value............... Multiple specifications can be given. Each specification optionally begins with a linestyle of "dashed" or "dotted." If neither is given, then normal solid lines will be used. Next is the bartype, which may be either "bar," for a single line, or "dblbar," for double lines. Next comes the appearance, which is two values in parentheses, and separated by the word "to." The first value in the pair tells where to start drawing relative to the top staff in each range, and the second tells where to stop drawing relative to the bottom staff in each range. Each of the values is the word "top" (meaning relative to the top line of the staff), "middle" (meaning relative to the middle line of the staff), or "bottom" (meaning relative to the bottom line of the staff), optionally followed by a plus or minus sign and a number of stepsizes to add or subtract to get to the endpoint. The appearance can instead be just the word "between," meaning to draw from the bottom line of a staff to the top line of the staff below it, and applies to all subsequent specifications on the line. Or the appearance can be omitted, which will result in something that looks like normal bar lines (in which case you'll likely want to be using dashed or dotted to distinguish from normal bars). The appearance is followed by one or more ranges of staffs, like for the barstyle parameter, so things like 1-3 or 1-4,5-8,9,12 or the word "all." Finally there is the keyword "time" followed by one or more counts at which to draw the subbars. The counts may include decimal parts. Note that subbars are only drawn on a staff when a note or rest actually occurs on that count on that staff.

Default............ not set

Context........... score

Takes effect .... at next music context

Examples........ subbarstyle=bar all time 4
subbarstyle=dotted dblbar (top-2 to bottom-2) 1-5 time 3.5
subbarstyle=bar 3-7 between time 2 dotted dblbar 1-2 time 3, 5.75

See also........... bar, time

Name .... swingunit
Description..... This parameter only affects MIDI output. Some styles of music are often written in "swing time," meaning the players are expected to play pairs of notes with the first twice as long as the second, even though they are written as if they were the same duration, or as if the first was three times as long as the second. The most common example would be where the written notation shows two eighth notes like 8;; or a dotted rhythm like 8.;16; but the musician "knows" that the composer really intended it to be played as if it were a triplet {4;8;}3; This parameter adjusts the Mup MIDI output to follow this performance convention. If this parameter is set, each measure is divided into segments of durations of "swingunit," starting at the beginning. (Usually the time signature divided by swingunit will be a whole number, but if not, the last piece will be shorter.) Then within each segment, the time where one group ends and the next group starts will be altered in either of these two circumstances: (1) The current boundary time is halfway into a swingunit, and each group is at least half a swingunit long, or (2) The current boundary time is 3/4 of the way into a swingunit, and the first group is at least 3/4 of a swingunit long, and the second group is at least 1/4 of a swingunit long. In both of these cases, the durations are altered so that the meeting point is 2/3 of the way into the swingunit.
Value.............. a time value, like 2, 4, or 8, or not set to anything. It can be a dotted value like 2. or 16.. although dotted values are rarely likely to be useful. It can even be a time expression like 2.-32 although that is even less likely to be useful.
Default............ not set
Context........... score, staff, voice
Takes effect .... at next music context
Examples........ swingunit = 4
swingunit =   // turn off swing
See also........... timeunit

Name .... sylposition
Description..... A | can be used in lyrics at the beginning of a syllable (after anything in angle brackets) to indicate syllable alignment. This will override the lyricsalign parameter, and may be useful for aligning verse numbers or to make syllables at the beginning of poetic lines line up. If the | is not preceded by a number, the sylposition parameter specifies the default alignment value to use. It is the number of points (1 point is 1/72 of an inch) from the horizontal "middle" of the chord to place the left edge of the syllable. Negative values are to the left of the middle, positive to the right, so this value is usually negative.
Value............... -100 to 100
Default............ -5
Context........... score, staff
Takes effect .... at next music context
Example .... sylposition = -4
See also........... lyricsalign

Name .... tabwhitebox
Description..... This parameter specifies whether or not to put a small white box behind each fret number on tablature staffs. This may make the music a little easier to read, since the staff lines won't be going through the middle of the fret numbers.
Value.............. y or n
Default........... n
Context.......... score, staff, voice
Takes effect .... at next music context
Examples........ tabwhitebox = y
See also.......... stafflines

Name .... textkeymap
Description..... This parameter specifies which keymap to use for rom, ital, bold, and boldital commands. See the section on keymaps in the Text Strings chapter for more details.
Value.............. a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default........... nothing
Context.......... score, staff
Takes effect .... at next music context
Examples........ textkeymap="Cyrillic"
textkeymap=
See also.......... defaultkeymap, endingkeymap, labelkeymap, lyricskeymap, printkeymap, rehearsalkeymap, withkeymap

Name .... time
Description..... This parameter sets the time signature. Music data for each measure is checked to ensure that the total time in the measure for each voice and verse adds up to exactly the time signature, though supplying too little will be allowed with a warning. Setting the time parameter will also reinitialize the timeunit parameter and the beamstyle parameter to their most recent values for the same time signature (which would be their default values if they had never been explicitly set for this time signature), unless they are also set in the same context. The printedtime parameter value is not reset.
Value.............. either a ratio of the form *N/D* or the word "cut" or "common." If the ratio form is used, *N* must be between 1 and 99 inclusive, and *D* must be 1, 2, 4, 8, 16, 32, or 64. If followed by "y," the time signature will be printed even if it didn't change. The numerator of the time signature can be the sum of several numbers, as in 3+4/4 or 2+3+2/2. You can also have several fractions added together, as in 3/4 + 4/4. It is also possible to provide "alternating" time signatures, where you list two (or more, although two is typical) time signatures separated by white space. Each measure then uses the next time signature in the list. For example, for 3/4 4/4, the first measure would be in 3/4 time, the second measure in 4/4, the third back in 3/4, the fourth in 4/4, and so forth. It is possible to combine all the various complexities, with things like 3+4/8 + 2+3/4 4+3/4, although that would be very uncommon. The time signature can optionally be followed by the letter "n" to specify that the time signature is not to be printed. Or it can be followed by the letter "y," which causes alternating time signatures to be treated differently. By default, the alternating signatures are printed just once, as a list, and the performer has to remember that each subsequent measure has a different time signature. Using y forces Mup to print the appropriate time signature on each measure.
Default............ 4/4
Context........... score
Takes effect .... immediately
Examples........ time = 6/8
time = cut
time = 13/16n
time = 2+3+4 / 8 // additive numerator
time = 3/4 + 4/4 // fractions added together
time = 4/4 3/4 // alternating
time = 3/4 6/8 y // alternating, printing time sig on every measure
See also........... beamstyle, printedtime, subbarstyle, timeunit

Name .... timeunit
Description..... This parameter sets the default time unit. If the first note of a measure has no time value specified, the value of the timeunit parameter will be used. If the time signature is changed in a context where timeunit is not also set, the timeunit parameter reverts back to its previous value for that time signature, which defaults to the value of the denominator (bottom number) of the new time signature.
Value............... 1/8, 1/4, 1/2, 1, 2, 4, 8, 16, 32, 64, 128, or 256 representing octuple whole, quadruple whole, double whole, whole, half, quarter, eighth, sixteenth, thirty-second, sixty-fourth, 128th, or 256th. (Each represents one half the time of the one before it in the list). This can be followed by zero or more dots. Each dot adds 50% of the previous note or dot to the time. It can also be a time expression, like 2+8 or 1-4+16. The time value must be less than or equal to the time signature. The timeunit value can be reinitialized indirectly by setting the time parameter. Setting the time parameter will set the timeunit to the value used most recently for that time signature.
Default............ the denominator (bottom number) of the time signature
Context........... score, staff, voice
Takes effect .... at next music context
Examples........ timeunit = 2
timeunit = 4.
timeunit = 2 + 8
See also........... swingunit, time

Name .... topmargin
Description..... This parameter sets the amount of white space margin to put at the top of each page. It is specified in inches if the units parameter is set to inches, or in centimeters if the units parameter is set to cm. This parameter can only be specified before any music or block input. Margins are unaffected by the "scale" parameter.
Value............... 0.0 to pageheight minus 0.5 inches
Default............ 0.5 inches
Context........... score
Takes effect .... applies to entire input, and cannot be changed after music input
Example .... topmargin = 0.8
See also........... bottommargin, leftmargin, rightmargin, pageheight, units

Name .... transpose

Description..... This parameter specify by what interval to transpose the music data. The interval can be larger than an octave, but must be a valid interval (e.g., there is no such thing as a perfect 6th). It is an error to specify a transposition value that would result in a key signature with more than 7 flats or sharps. It is also an error if transposition would result in a note requiring a triple sharp or triple flat.

Value.............. the word "up" or "down," followed by an interval and a whole number greater than 0. You can optionally add the keyword "notes" or "chords" at the end, to restrict the transposition to just notes or just chord symbols; by default, both are transposed. The interval is one of major, minor, augmented, diminished, or perfect. The intervals can be abbreviated to their first 3 letters (maj, min, aug, dim, or per). The section on transposition lists transposition intervals and gives further details. Depending on which key signature you are transposing from, some transposition intervals may not work because they result in more than 7 flats or sharps. There is also another parameter called addtranspose. Typically you would use the transpose parameter to change the key of individual staffs (for transposing instruments), and then use the addtranspose parameter if you want to change the key of the entire score. But either of these parameters can be used either way. In any case, for each staff, and for the score, the values of transpose and addtranspose are both applied, one after the other, to the current key signature, notes, and chords to determine their resulting values.

Default............ up perfect 1 (i.e., no transposition)

Context........... score, staff

Takes effect .... at next music context

Examples........ transpose = up minor 3
transpose = down perfect 4
transpose = up minor 3 chords

See also........... a4freq, addtranspose, key, useaccs

Name .... tuning

Description..... This parameter specifies what tuning system to use for the "white" notes" (a through g) and the standard accidentals. In equal temperament, each octave is divided into 12 equally spaced half steps, with the frequency of each note being the twelfth root of 2 times that of the note below it. In pythagorean tuning, perfect fifths have a ratio of 3/2. In meantone, major thirds have a ratio of 5/4. See the chapter on Custom Accidentals and Alternate Tunings for more information.

Value.............. equal, pythagorean, or meantone

Default............ equal

Context........... score

Takes effect .... at next music context

Example .... tuning = meantone

See also........... a4freq, acctable

Name .... tupletslope
Description..... This parameter allows you to control the slope of tuplet brackets. Two values must be given, separated by a comma. Mup calculates an appropriate slope for tuplet brackets by applying a linear regression algorithm. The first value supplied for the beamslope parameter is a factor by which to multiply the default slope that Mup calculates. The minimum value of 0.0 would cause all brackets to be horizontal, whereas the maximum value of 1.0 will use the slope Mup calculates. Intermediate values will yield brackets that are less slanted than the default slope calculation. The second value given to the tupletslope parameter is the maximum angle for the bracket, in degrees. If the originally calculated value multiplied by the factor yields an angle of greater than this maximum angle, the maximum angle will be used.
Value.............. 0.0 to 1.0 for the factor, and 0.0 to 45.0 for the maximum angle
Default........... 0.7, 20
Context.......... score, staff, voice
Takes effect .... immediately
Example .... tupletslope=0.5,15
See also.......... beamslope

Name .... units
Description..... This parameter specifies whether margin and page size parameters are specified in inches or in centimeters.
Value.............. inches or cm
Default........... inches
Context.......... score
Takes effect .... immediately
Example .... units = cm
See also.......... topmargin, bottommargin leftmargin, rightmargin, pageheight, pagewidth

Name .... useaccs
Description..... This parameter specifies whether to use accidentals throughout rather than a normal key signature. A value of "n" means to use a key signature, and only use accidentals where specified by the user. A value of "y none" means to not use a key signature, but instead add accidentals everywhere as would be required by the key signature, using the standard practice that an accidental remains in effect for the remainder of the current measure. A value of "y all" causes accidentals to be placed on every single note. A value of "y nonnat" causes accidentals to be placed on every single note except when the accidental would be a natural, with naturals only printed when they would be required when using none. A value of "y noneremuser" is like "y none" but removes unnecessary user accidentals. A value of "y nonnatremuser" is like "y nonnat" but removes unnecessary user accidentals. A "y" by itself without a qualifier is an abbreviation for the "y none" value. When using this parameter, you may sometimes want to use the carryaccs parameter as well.
Value.............. n, y none, y all, y nonnat, y noneremuser, y nonnatremuser
Default........... n
Context.......... score, staff
Takes effect .... at next music context. If you turn it on in the middle of a score, it prints a key signature of naturals to cancel the key signature.
Example .... useaccs = y all
See also.......... addtranspose, carryaccs, key, cancelkey, transpose

Name .... vcombine

Description..... This parameter tells Mup to combine the specified voices onto a single stem whenever possible. One common use would be if you want multiple voices for MIDI purposes, but want them printed on the same stems. Another typical use would be to obtain a printing style common for hymns and certain other styles of music, where the two voices on each staff are printed on one common stem whenever possible, but when a note is shared between two voices, two opposing stems are used to make it clear the note is indeed shared. The value of this parameter is a list of voices plus an optional qualifier. The list format is like elsewhere for voices; common examples would be `1,2` or `1-3` or `2-3`. Order of voices is significant: sometimes Mup may have to choose between two possible combinations, so voices listed first get priority. The list of voices can be followed by a qualifier to specify what happens when voices overlap. If the qualifier is "nooverlap," voices will only be combined if the bottom note of the higher voice is higher than the top note of the lower voice. If the qualifier is "stepsapart," voices will only be combined if the bottom note of the higher voice is at least two steps higher than the top note of the lower voice. (That is the typical value for getting the hymn style described above.) If the qualifier is "shareone," the bottom note of the top voice must be no lower than than top note of the lower voice for combining to occur. If the qualifier is "overlap," combining will occur without regard for how the voices overlap. For the purpose of the qualifier, voice 1 is assumed to be the highest voice, voice 3 the middle voice, and voice 2 the lowest. If the qualifier is "restsonly", notes will never be combined, but rests will be combined whenever possible. If no qualifier is specified, the default is nooverlap. Finally, an independent qualifier of "bymeas" can be specified, in which case combining will only be done in measures where the combining can be done on all chords in the measure. While the vcombine parameter is allowed to be used with any vscheme parameter value, using it with vscheme=1 is pointless, and only vscheme values of 2f and 3f are really appropriate. This parameters can be used with both voice-at-a-time and chord-at-a-time input styles. It has no effect on tablature or 1-line staffs. Note that there are various cases where combining will not be done, such as when time values or beamings are different in different voices, and cases where combining would cause information loss, such as when a shared note is tied in one voice but not another. In such cases, the usual non-combined format will be used.

Value............... comma-separated list of voices or voice ranges, or nothing, optionally followed by nooverlap, stepsapart, shareone, overlap, or restsonly, optionally followed by bymeas

Default............ not set

Context........... score, staff

It is possible to change this parameter in the middle of a measure, using << >>.

Takes effect .... at next music context

Examples........ vcombine=3,1-2 shareone bymeas

vcombine=   // turn off combining

See also........... vscheme

Name .... visible
Description..... This parameter specifies whether a staff or voice is actually to be printed. This can be useful for printing a subset of a full score. The value is either y or n, for yes or no, or whereused. When whereused is specified, if a staff has no notes or lyrics or other associated things on an entire score, that staff is not printed. This might be used, for example, to save paper on an orchestral score by only printing staffs for instruments when they are actually playing. At least one staff must be visible at all times. When an individual voice is made invisible, but the other voice(s) on that staff remains visible, all the tempo, dynamics, and similar marks associated with the staff will still be printed, since Mup cannot know for sure whether you meant them to be associated with a particular voice or with the staff as a whole. When MIDI output is generated, this parameter controls whether the staff or voice is audible, so you can control which voices are played. The -s command line argument can also be used to control which staffs are printed or played.
Value............... y, n, or whereused
Default............ y
Context........... score, staff, voice
Takes effect .... at next music context. If it results in a staff becoming visible or invisible, it forces a new score.
Example .... visible = n
See also........... brace, bracket, endingstyle, staffs

Name .... vscheme
Description..... This parameter sets voice scheme. A value of 1 means there is only a single voice on a staff. The direction of note stems will be determined based on how high or low the notes are on the staff. A value of 2o means there are two voices with "opposing" stems. In other words, the stems of voice 1 will always point upward, and the stems of voice 2 will always point downward, unless they are explicitly forced the other way. A value of 2f means there are two voices with "free" or "floating" stems. That means in places where there are notes or rests in both voices, stem directions will be as if 2o were set. However, if one of the voices has "space" where there are no notes or rests, the stem directions of the other voice will be determined as if there were only a single voice. 2o is useful if you want to force stem directions a certain way. 2f is generally preferable when there are two voices only part of the time. The values 3o and 3f are like 2o and 2f except that a third voice is allowed. The third voice's stem defaults to up, but the direction can be changed at any chord. The stem direction remains in effect on subsequent chords of voice 3 until explicitly changed. While there can be voice crossings, in general voice 1 should be the "top" voice, voice 2 the "bottom" voice, and voice 3 the "middle" or "extra" voice. Mup does not use voice 3 when associating things like phrase marks and lyrics with chords. Setting vscheme to a different number of voices will reinitialize all voice level parameters for the affected staffs.
Value............... 1, 2o, 2f, 3o, or 3f
Default............ 1
Context........... score, staff
Example .... vscheme = 2f
Takes effect .... immediately for purposes of checking for interactions with other parameters, at next music context for printing
See also........... vcombine

Name .... warn
Description..... This parameter specifies whether to print warning messages or not. Normally, Mup will print warnings when it encounters input that it considers somewhat dubious. Sometimes, however, that input will really be what you want, so this parameter allows you to turn off warning messages.
Value.............. y or n
Default........... y
Context.......... score
Takes effect .... immediately
Example .... warn = n

Name .... withfont
Description..... This parameter specifies which font type to use for text strings printed with chords using [with "string"].
Value.............. rom, ital, bold, or boldital
Default........... rom
Context.......... score, staff, voice
Takes effect .... at next music context
Example .... withfont=boldital
See also.......... font, fontfamily, size, withfontfamily, withsize

Name .... withfontfamily
Description..... This parameter specifies which font family to use for text strings printed with chords using [with "string"].
Value.............. avantgarde, bookman, courier, helvetica, newcentury, palatino, or times
Default........... times
Context.......... score, staff, voice
Takes effect .... at next music context
Example .... withfontfamily=helvetica
See also.......... font, fontfamily, size, withfont, withsize

Name .... withkeymap
Description..... This parameter specifies which keymap to use for text strings printed with chords using [with "string"]. See the section on keymaps in the Text Strings chapter for more details.
Value.............. a string matching the name of a keymap defined earlier, or nothing. An empty string ("") means don't do any mapping. Setting to nothing unsets any previous value, which effectively causes the defaultkeymap value to be used.
Default........... nothing
Context.......... score, staff
Takes effect .... at next music context
Examples........ withkeymap="Cyrillic"
withkeymap=
See also.......... defaultkeymap, endingkeymap, labelkeymap, lyricskeymap, printkeymap, rehearsalkeymap, textkeymap

Name .... withsize
Description..... This parameter specifies which size to use for text strings printed with chords using [with "string"], specified in points.
Value............... 1 to 100
Default............ 12
Context........... score, staff, voice
Takes effect .... at next music context
Example .... withsize=15
See also........... font, fontfamily, size, withfont, withfontfamily

15. HINTS

This section contains hints on how you can use Mup to accomplish various things that have not been covered up to this point. It doesn't introduce any new language features, but describes some additional ways to apply what you have already learned, and mentions some other tools you may find useful.

15.1 Debugging

Since Mup requires its input in a fairly strict format, when a song is put in, it may contain "typos." Generally, the error messages that Mup prints will give you an idea of what is wrong. However, sometimes Mup is not able to recognize that something is wrong until some distance beyond the actual error. If you can't find anything wrong with the line that Mup lists as being in error, try looking at the end of the previous line, or even earlier lines. Some of the most common problems are missing semicolons and missing quotes. Missing quote marks tend to be especially confusing to Mup, and may cause many error messages, even though there is only one problem. Another common problem that may cause a very large number of error messages is forgetting to state "music" to enter music context.

Often listening to MIDI output is much more effective at spotting things like wrong notes and missing accidentals than trying to find them by eye.

15.2 Adjusting Output

Mup does its best to lay out the music in an aesthetically pleasing way. Often, however, you may want to make adjustments. Perhaps the last part of a piece spilled over onto a third page and you'd like to squeeze it all on two pages, or a page turn falls at an awkward spot. There are several mechanisms available for making adjustments. They have already been discussed individually in various sections of this document, but this section tries to pull things together.

The "newscore," "newpage," "samescorebegin / samescoreend," and "samepagebegin / samepageend" commands can be used to force where breaks do or do not occur. This may be useful for ensuring a section ends at the end of a score or page.

If you want to get a little more or less on each page, it is usually best to start with changing the scale, musicscale, packfact, and/or packexp parameters. You may want to experiment with changing these individually first, to get a feel for how they work, as trying to change all of them at once may lead to interactions that change things more radically than you might expect. Changing musicscale lets you adjust the size of the music without affecting the size of the text in headers and footers. Adjusting the margins is sometimes helpful as well. Other parameters that might be useful in some situations are: maxscores, scorepad, scoresep, staffpad, and staffsep.

The dist, dyndist, and chorddist parameters are useful if you want items to line up vertically. The "align" and "dist" option can be used on rom, bold, ital, boldital, octave, mussym, crescendo and decrescendo statements, to force something where you want it. (The "dist" option can be used on rehearsal marks as well.)

The appearance of lyrics can be adjusted using the lyricsdist, lyricsfont, lyricsfontfamily, lyricssize, and lyricsalign parameters.

Printers often cannot print all the way to the edges of the paper, and sometimes print at some fixed offset from the actual corner of the page. Ghostscript includes an "align.ps" file that you can print that gives you instructions on how to compensate for that, by adding a special "setpagedevice" line to your PostScript files. That special line can be added to Mup output using the postscript afterprolog hook. Since hooks are wrapped in save/restore blocks, and the setpagedevice has to not be in such a block, restore and save lines have to be used to undo the block that Mup adds:

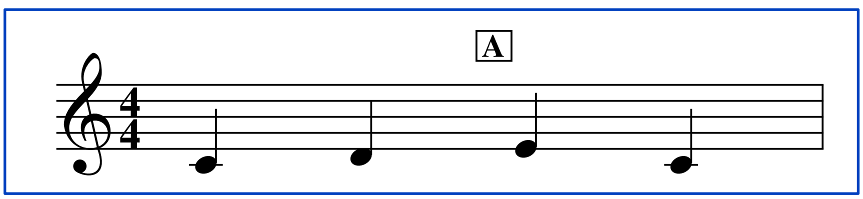
```
postscript afterprolog "
restore
<<  /.HWMargins [18 18 18 12.5] /Margins [-75 0]  >>  setpagedevice
save"
```

Replace the numbers with those calculated by following the align.ps instructions. Note that the spacing may need to be exactly as align.ps says.

15.3 Special uses of invisbar

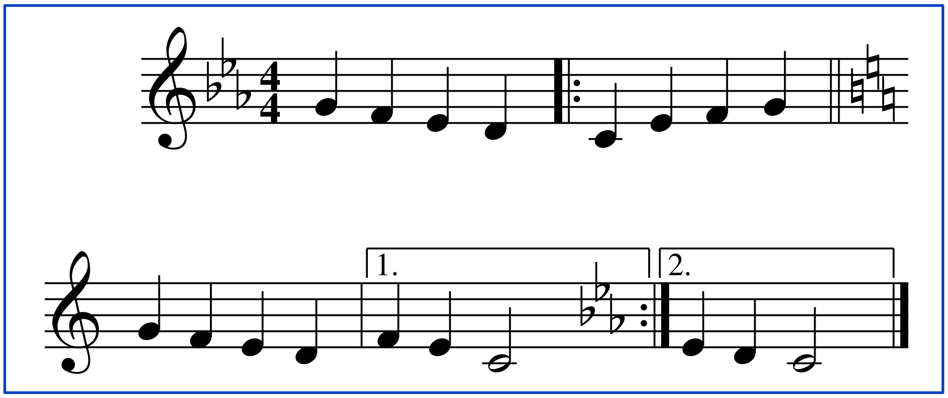
The "invisbar" can be used to force Mup into doing something in the middle of a bar that it normally would allow to happen only at a bar line. Suppose, for example, you wish to place a rehearsal letter in the middle of a measure. This could be accomplished as follows:

```
// assume we are in 4/4 time, but want
// a rehearsal letter by count 3
// of the measure

// do first part of measure and use "space"
// for last part so time values will add up
// properly to a full measure
1: 4c;d;2s;

// put in invisible bar with rehearsal letter
invisbar rehearsal let

// now do the last half of the measure,
// this time with space at the beginning
1: 2s;4e;c;
bar
```



As another example of invisbar use, suppose you want to add a "courtesy" key signature at the end of a repeated section to remind the player that the beginning of the repeated section is in a different key. This can be done by adding an empty measure whose sole purpose is to produce this key signature.

```
score key=3&
music
1: g;f;e;d;
repeatstart
1: c;e;f;g;
dblbar
score key=0&
music
1: g;f;e;d;
bar ending "1."
1: f;e;2c;
// add a courtesy key signature,
// to remind player the beginning
// of the repeated section is in
// a different key
invisbar
score key=3&
music
1: ms;
repeatend ending "2."
1: e;d;2c;
endbar
```



You can use a similar technique to insert time signatures, clefs, etc. at unusual places.

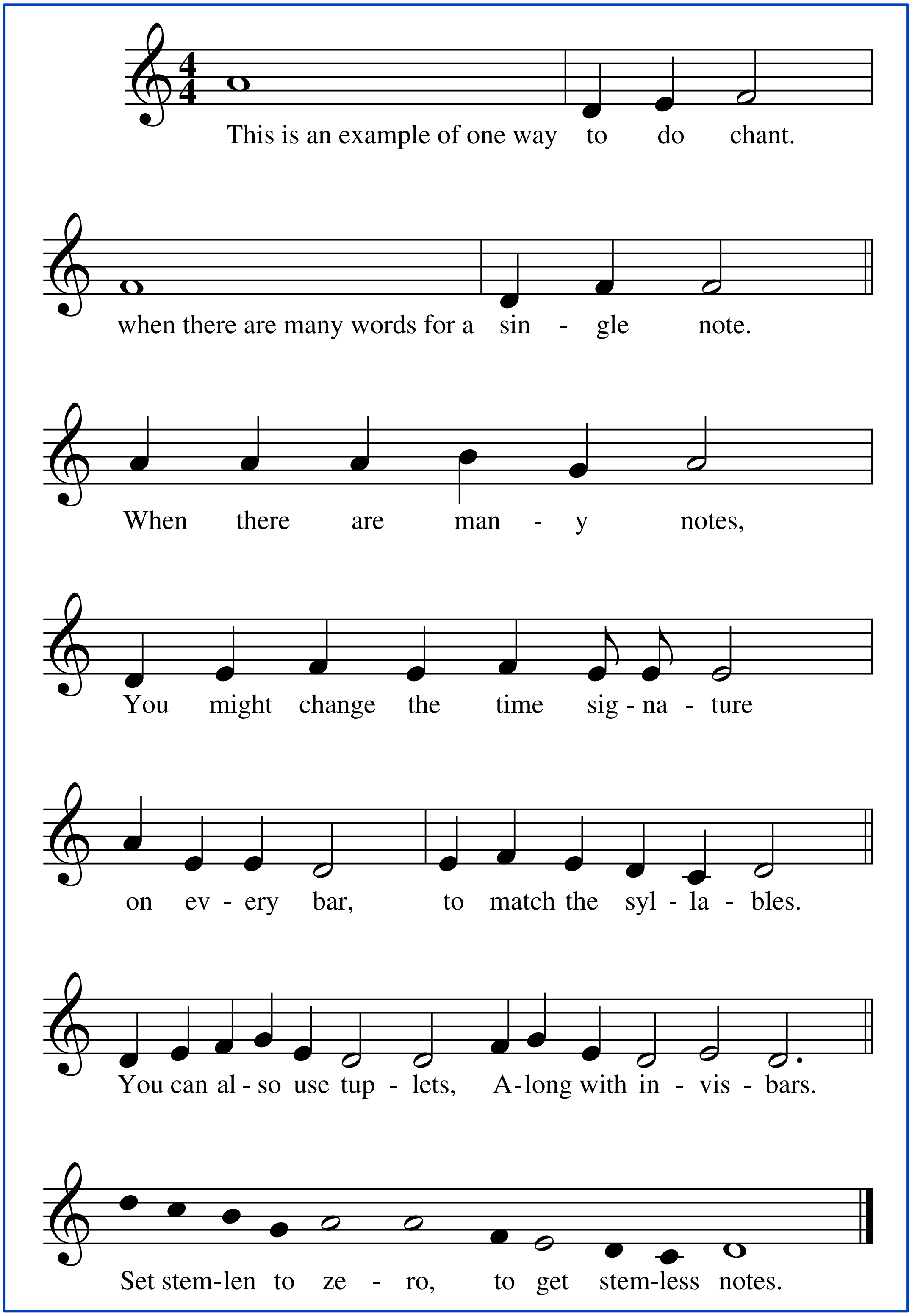
15.4 Chant

Chant typically uses an irregular number of beats per measure, or often no measures at all in the normal sense. Mup checks to make sure you provide enough notes to fill the time signature, but will allow you to use less, filling the remainder with space. So one possible approach to writing chant is to specify a large time signature—at least as big as you will ever have on any given line of music—and then supplying as many notes as you need, letting Mup pad with space as needed. If the warnings are annoying, you can set the warn parameter to turn the warnings off.

Here is an example that shows some other techniques you might use when writing chant.

```
1: 1a;
lyrics 1: "This<^ is an example of one way>";
bar

1: d;e;2f;
lyrics 1: "to do chant.";
bar

1: 1f;
lyrics 1: "when<^ there are many words for a>";
bar

1: d;f;2;
lyrics 1: "sin-gle note.";
dblbar

newscore
// Note use of 'n' to not print the time signature
score time=7/4n
music
1: a;;;b;g;2a;
lyrics 1: "When there are man-y notes,";
bar

score time=8/4n
music
1: d;e;f;e;f;8e;;2;
lyrics 1: "You might change the time sig-na-ture";
bar

score time=5/4n
music
1: a;e;;2d;
lyrics 1: "on ev-ery bar,";
bar

score time=7/4n
music
1: e;f;e;d;c;2d;
lyrics 1: "to match the syl-la-bles.";
dblbar

newscore

score time=8/4n
music
// Note use of 'n' to not print tuplet number/bracket
1: {d;e;f;g;e;2d;;}10n,1/2;
lyrics 1: "You can al-so use tup-lets,";
invisbar

1: {f;g;e;2d;e;2.d;}9n,1/2;
lyrics 1: "A-long with in-vis-bars.";
dblbar
```

```
newscore

score stemlen=0
music

1: d+;c+;b;g;2a;;
lyrics 1: "Set stem-len to ze-ro,";
invisbar

1: {f;2e;4d;c;1d;}9n,1/2;
lyrics 1: "to get stem-less notes.";
endbar
```



15.5 Forcing shared noteheads

Mup will automatically share noteheads when it can figure out it is safe and proper to do so. However, there may be some unusual cases where you would like to force the notes from two voices to share noteheads even when Mup would not do that. That can be done by specifying a horizontal offset of zero, i.e., [ho 0], on one or both of the voices.

15.6 Manually placed tuplet numbers

Generally, Mup will place tuplet numbers for you. However, you do need to print them yourself on cross-staff beams. And there may be cases where you choose to use 'n' to turn off Mup's automatic printing of tuplet numbers, in order to print them manually in a different place than Mup would. Mup normally uses newcentury boldital font for tuplet numbers, in 11-point size for regular notes and 9-point for cue notes. So to make your manually placed tuplet numbers look the same as automatic ones, you might use a macro something like this:

```
define TUPNUM(NUM) "\f(NX)\s(11)" +‘NUM‘ @
1: { 8c; d; e; } 3n; 2.us;
rom above 1 dist0: 1.34 TUPNUM(3);
bar
```

Or another approach would be something like this:

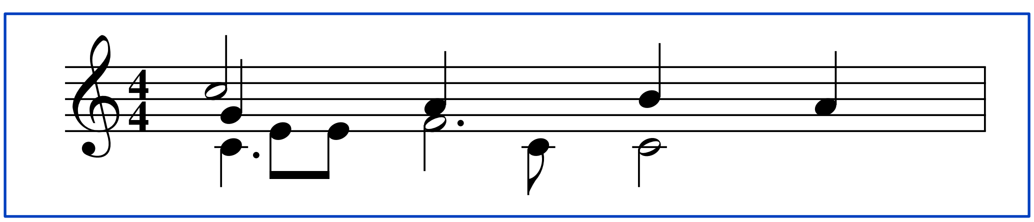
```
define TN(NUM,WHERE,D,N) newcentury boldital WHERE dist D: N ‘NUM‘;@
1: { 8c; d; e; } 3n; 2.us;
TN(3,above 1,0,1.34)
bar
```

15.7 Manual placement of notes

Mup supports up to three voices per staff. If you need more than that, such as when 4 notes of different lengths occur on the same beat, it is possible to position extra notes manually. Manually positioned notes will not be included in MIDI output.

First of all, unless the fourth voice is vertically far away from the other two, you'll probably need to reserve some extra space to the left or right of the other chords. This can be done by adding a bit of padding to the chord. If you want the extra note on the left of the regular voices, add padding to whichever regular voice is leftmost. If you want it on the right, the padding will have to go on the following note, or on the bar line if you are on the last chord of a measure. The amount of padding to add may have to be determined by trial and error; 5 stepsizes is a good first guess. Next, set a location tag on one of the notes in one of the regular groups. The manually positioned note will be placed relative to that location tag. The x will be slightly left or right of the west or east of the existing group; something like 3 stepsizes might be a good first guess. The y can be specified in terms of stepsizes up from the note you used for the location tag. You can use a print statement using one of the special music characters, such as: dblwhole, 1n, up2n, dn2n, up4n, up32n, etc. If the note needs ledger lines, dots, or accidentals, these too have to be manually positioned, which can be a bit tricky. Here is a simple example:

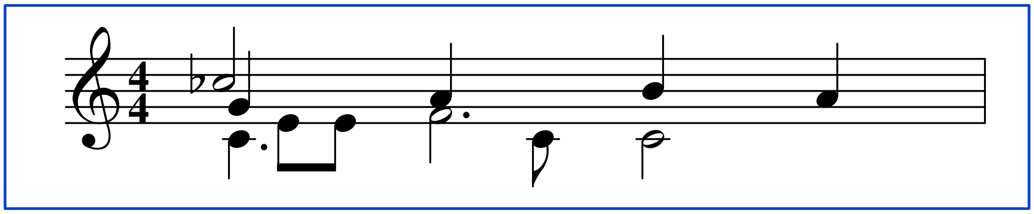
```
score
vscheme=3o
beamstyle=4,4,4,4

music
1 1: [pad 4] 4g =h; a; b; a;
1 2: 4.c;8;2;
1 3: [down] 8e;; [ho 0] 2.f;
print (h.w - 2.3, h.y + 3) "\(up2n)"
bar
```



In this example, 4 stepsizes of padding was added to the quarter note g of voice 1 on staff 1. A location tag "h" was set to this note. After the information about the second voice on staff 1, a half note was manually placed 2.3 stepsizes left of the first voice and at pitch c+ (3 steps up from the g).

If you also need to place an accidental and/or dots, more padding should be requested, and additional print statements used for each item. For example, you could specify perhaps 5 stepsizes of padding, then add:

```
print (h.w - 5, h.y + 3) "\(flat)"
```



to place a flat sign in front of the c.

Ledger lines needed by the notes also have to be placed manually, using the "line" command. If the note is more than three steps above or below the staff, more than one ledger line would be needed, and each would need to be specified separately. The y coordinate of the each ledger line would be the same as the y coordinate of the note, plus or minus some number of step sizes. The length of the line depends on the notehead. About 4.5 stepsizes is a good estimate for most notes, but a double whole needs more like 6 stepsizes.

15.8 Trill with accidental

Sometimes when notating a trill, you may want to place an accidental between the "tr" and the wavy line. The typical way of getting a trill, something like

```
mussym above 1: 1 "tr" til 3;
```

doesn’t allow for such an accidental. However, the desired output can be achieved with a slightly different input:

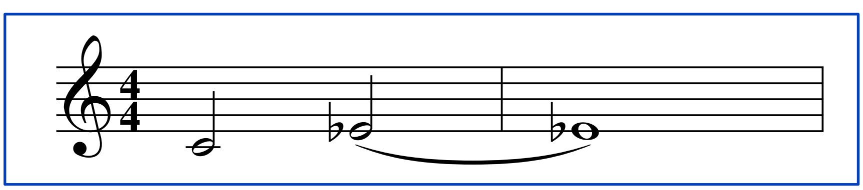
```
rom above 1: 1 "\(tr)\s(-2)\(flat)~" til 3;
```

The \s(-2) makes the flat a little smaller; you can adjust the actual size to your taste. The tilde at the end of the string tells Mup to use a wavy line for the til clause.

15.9 Accidental on tied-to note

When a note with an accidental is tied across a bar line, the standard convention is to not print the accidental on the tied-to note, and Mup uses that convention. If you really want to have the accidental printed, you can specify a slur rather than a tie. If you want the MIDI output to be correct, you can use an ifdef to use slur for printing and tie for MIDI.

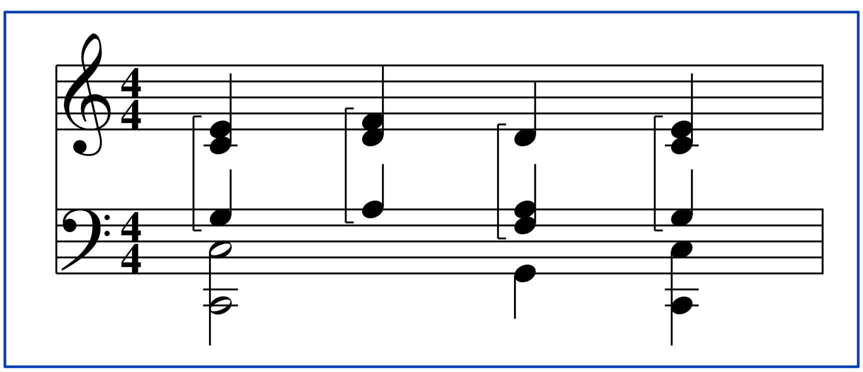
```
1: 2c;e& ifdef MIDI ~ else <> endif ;
bar

1: 1e&;
bar
```



15.10 Bracketing notes across staffs

In keyboard music, sometimes a bracket is drawn to indicate that notes from two staffs are to be played with the same hand. The bracket is really just a vertical line with short horizontal lines at each end. But if you need to make lots of brackets, a macro with parameters can be very helpful.

```
score
  staffs=2

staff 2
  clef=bass
  vscheme=2f

// Define a macro to draw a bracket to show that notes on two
// different staffs are to be played with the same hand.
// The parameters are location tags for the top and bottom notes
// to be included in the bracket.
define BRACK(TOP, BOT)
// Draw a short horizontal line 0.5 stepsizes above the top note
line (TOP.w - 2, TOP.n + 0.5) to (TOP.w - 1, TOP.n + 0.5)
// Draw a vertical line from 0.5 stepsizes above the top note
// to 0.5 stepsizes below the bottom note.
// Do all the 'x' coordinates relative to the same note (in
// this case the top), so that if the top and bottom chord happen
// to be different widths, the line will still be vertical.
line (TOP.w - 2, TOP.n + 0.5) to (TOP.w - 2, BOT.s - 0.5)
// Draw short horizontal line just below and left of the bottom note.
line (TOP.w - 2, BOT.s - 0.5) to (TOP.w - 1, BOT.s - 0.5)
@

music

// For each chord that is to get a bracket, add some padding to
// make sure there is enough room, and set a location tag
// on the top and bottom notes.
1: [pad 2] ce =a;[] df =b; [] d =c; [] ce =d;
2: [pad 2] g =e; [] a =f; [] af =g; [] g =h;
2 2: 2cc-; 4g-; cc-;
// Now draw the brackets, using the tags as parameters
BRACK (a, e)
BRACK (b, f)
BRACK (c, g)
BRACK (d, h)
bar
```



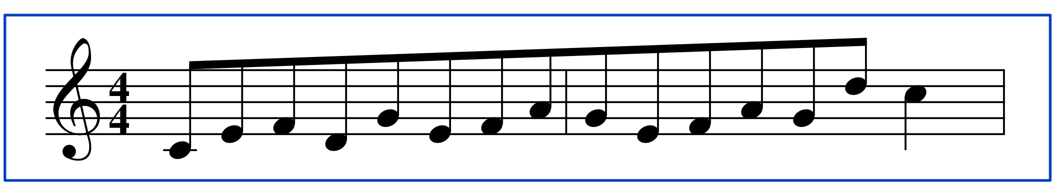
Note that Mup supports cross-staff stems, which is another way to notate a chord that is split across two staffs, and may often be a better choice.

15.11 Cross-bar beaming

Generally, beams do not cross bar lines, and Mup follows that rule. However, if you want beams to cross a bar line, there are at least two possible approaches to getting the desired effect. The first is to explicitly specify stem lengths such that all the beams in the various measures will lie on the same line. You can then use the line command to fill in the gaps in the beams across the bar lines.

The other approach is to draw in the bar line using the "line" construct. To do this you first tell Mup that a measure is twice as long as is really is, so that you can put two actual measures inside what Mup thinks is a single measure. Here is an example.

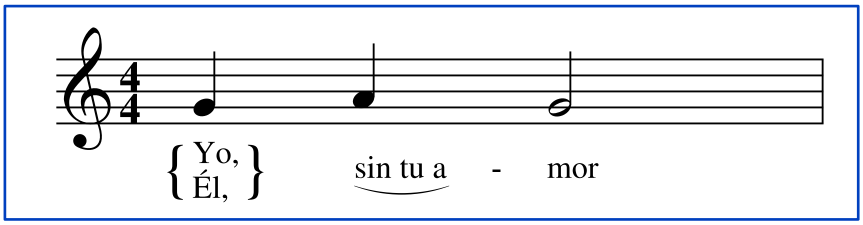
```
// First make an empty measure ending with an
// invisible bar. This is for the sole purpose
// of allowing the real time signature to be
// printed and would not be necessary if we
// wanted to cross a bar line other than
// the very first bar line of the piece.
1: ms;
invisbar

// Now, make the effective time signature twice
// as long as the real time signature, but use "n" so this fake
// time signature is not actually printed.
// Set up for beaming across the entire double-length measure.
// (You could could use other beamstyles if you wish, or custom beaming.)
score time=8/4n
beamstyle=1/2
music

// Now do the double-length measure. On the chord
// just after where we want a bar line, add some extra
// padding and set a location tag so that we can draw
// a bar line relative to the tag.
1: 8c;e;f;d;g;e;f;a; [pad 3; =a]g;e;f;a;g;d+;4c+;
// Also set a location tag on the next bar line,
// so that we can get the vertical endpoints of the
// bar line that we draw from the endpoints of the
// normal Mup-supplied bar line.
bar =b

// Now draw the bar line that goes through the beam.
// Use the horizontal position relative to the "a"
// location tag on the chord in the second actual measure,
// and get the vertical endpoints from the "b" tag
// associated with the next bar line.
line (a.w + 1, b.y + 4) to (a.w + 1, b.y - 4)
```



15.12 Printing relative to lyric syllables

Here is an example that shows how location tags associated with lyric syllables might be useful:

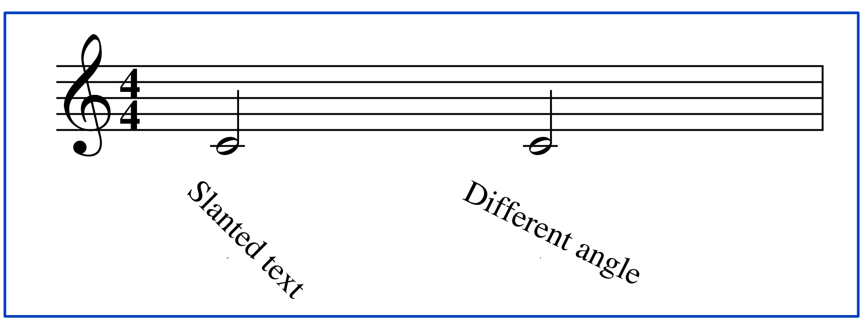
```
1: g;a;2g;
lyrics below 1 :  4;;2; "\v(+40)Yo,\v(-40)\=(_yo) sin<^ tu a>\=(_words)-mor";

// Print an alternate word with brackets.
print (_yo.w, _yo.y - 2.5) "\(E’)l,"
print (_yo.w - 4, _yo.y - 1.3) "\s(+12){"
print (_yo.e - 1, _yo.y - 1.3) "\s(+12)}"

// print curve to join three syllables
curve (_words.w, _words.y - 1) to (_words.e - 1, _words.y - 1) bulge -1
bar
```



15.13 Printing slanted text

While Mup doesn’t have a command for directly printing text at some particular angle, it does allow printing text above a line, and that line can be at an angle. The original intent was to use that for printing "glissando," but it can be used for other things too. So to print arbitrary text at an arbitrary angle, you can make the line so tiny it can hardly be seen. Because of roundoff, it is generally not possible to make it so small that you can’t see it at all, but it can be small enough to look like just a stray ink dot. By adjusting the relative location of the endpoints of the tiny line, you can control the angle.

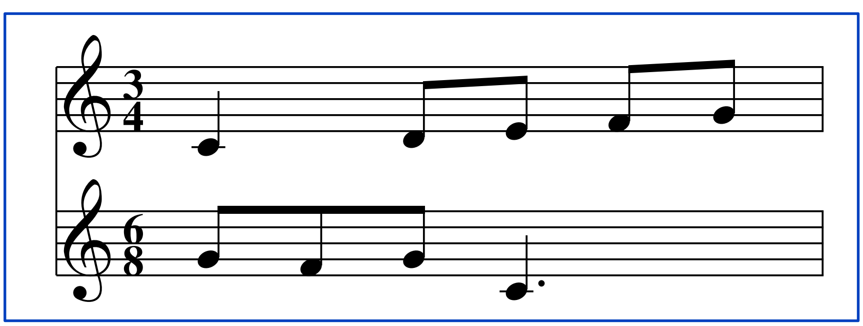
```
1: 2c =c; =q;
line (c.x, c.y - 14) to (c.x + 0.05, c.y -14.05) with "Slanted text";
line (q.x, q.y - 14) to (q.x + 0.04, q.y -14.02) with "Different angle";
bar
```



15.14 Mixed time signatures

Once in a while, music is written with different time signatures on different staffs. Starting with Version 6.8, there is a printedtime parameter to do that, so this section is probably not really necessary anymore, but it provides an example of how to print things manually for the case where the time signatures reduce to the same value, for example, 3/4 and 6/8 time (since 6/8 taken as a fraction and reduced to lowest terms is 3/4). You can make the output use both 3/4 and 6/8 by using a non-printing time signature, then placing the time signatures manually. Here is an example of how to do that:

```
score
    // set time signature to 3/4 but don't print it
    time=3/4n
    staffs=2

staff 1
    // We want this staff to be in 3/4 time,
    // so beam things in groups of
    // quarter note times.
    beamstyle=4,4,4

staff 2
    // We want this staff to effectively be
    // in 6/8 time, so we'll beam things
    // in groups of dotted quarters.
    beamstyle=4., 4.
    // In real 6/8 time, the time unit would
    // be eighth note, so make that the default
    timeunit=8

music

// Add padding to the first chord on at least one
// of the staffs, to make room for the manually placed
// time signatures, and set location tags
1: [pad 5; =t] c; 8d; e; f; g;
2: [=s] g; f; g; 4.c;

// Manually place the time signatures
// They are printed in 16-point newcentury bold font,
// relative to the location tags that were set.
// First print the 3/4
print (t.w - 4, t.y) "\f(newcentury bold)\s(16)3";
print (t.w - 4, t.y - 4) "\f(newcentury bold)\s(16)4";

// Then print the 6/8
print (t.w - 4, s.y) "\f(newcentury bold)\s(16)6";
print (t.w - 4, s.y - 4) "\f(newcentury bold)\s(16)8";
bar
```

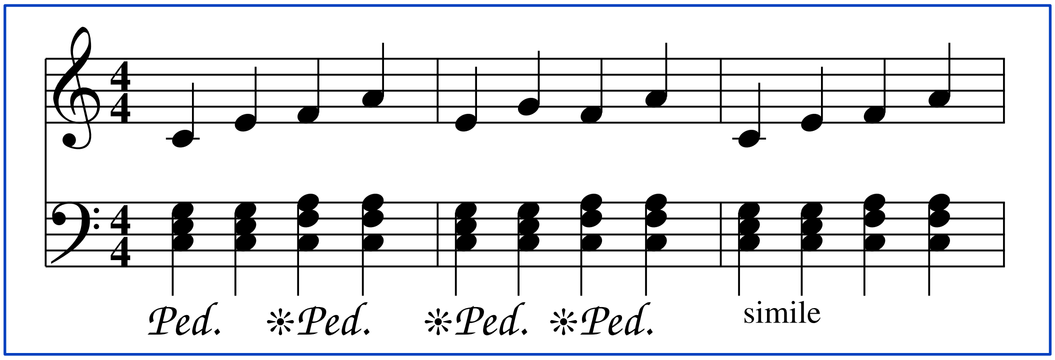


15.15 Defining a macro inside another macro

One possible use for defining one macro inside another is to make it act differently depending on how many times it has been called.

Suppose you want to have pedal on the first and third count of each measure, but instead of having that printed for the whole song, you'd rather just print pedal marks for a couple measures, and then say "simile." However, for MIDI, you would want the pedal to apply to the entire song, as well as both staffs, not just the one it is printed under. One way to accomplish that is given below. A PED macro is defined. Each time it is called, it defines another macro (ONCE, TWICE, and THRICE) to keep track of how many times it has been called. On the third time, it prints the "simile." In any case, if it is not doing MIDI and has not yet reached THRICE, it prints the pedal marks. If it is doing MIDI, it always applies pedal to both staffs.

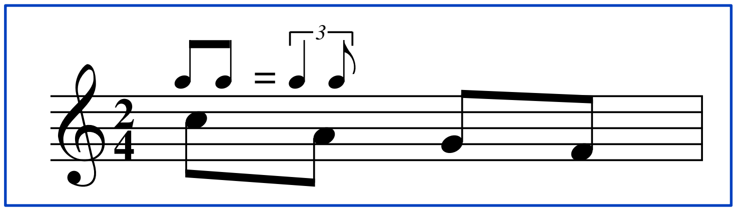
```
score
    staffs=2
    pedstyle=pedstar
    alignped=n
staff 2
    clef=bass

define PED
    ifndef ONCE
        define ONCE @
    else
        ifndef TWICE
            define TWICE @
        else
            ifndef THRICE
                define THRICE @
                rom below 2: 1 "simile";
            endif
        endif
    endif

    ifdef MIDI
        pedal below 1-2: 1;3;
    else
        ifndef THRICE
            pedal below 2: 1;3;
        endif
    endif
@

music

1: c;e;f;a;
2: ceg;;cfa;;
PED
bar

1: e;g;f;a;
2: ceg;;cfa;;
PED
bar

1: c;e;f;a;
2: ceg;;cfa;;
PED
bar
```



15.16 Marking complicated tempo changes

Sometimes you may wish to indicate tempo changes by showing two note values with an equals sign between them. For simple cases, this is fairly straightforward, but if you want to include beamed notes or a triplet, a little more work is required. Another use of this notation is to show that the music is to be played in "swing time," as in this example, which has two ordinary eighth notes on one side of the equals sign, and a triplet made up of a quarter and eighth note on the other:

```
score
    // This example assumes "size" is set to 15
    size=15
    beamstyle=4,4

music

// set a location tag
1: [=c] 8c+;a;g;f;

// set X and Y to where the first note will be placed
define X c.x - 3 @
define Y c.y + 5 @

// print "straight 8ths = triplet quarter plus 8th"

// print the basic notes and equal sign
print (X, Y) "\(smup4n)  \(smup4n)  = \(smup4n)  \(smup8n)"

// print the beam between the straight 8th notes
wide line (X + 2.1, Y + 5.5) to (X + 7.2, Y + 5.5)

// now do the triplet bracket
line (X + 14.7, Y + 7.0)  to (X + 17.6, Y + 7.0) // horz left top segment
line (X + 19.4, Y + 7.0)  to (X + 22.5, Y + 7.0) // horz right top segment
line (X + 14.7, Y + 7.05) to (X + 14.7, Y + 5.3) // vert left side segment
line (X + 22.5, Y + 7.05) to (X + 22.5, Y + 5.3) // vert right side segment

// print the 3 in the middle of the bracket
print (X + 17.9, Y + 6.1) "\s(8)\f(TI)3"  // the 3 for the triplet

bar
```

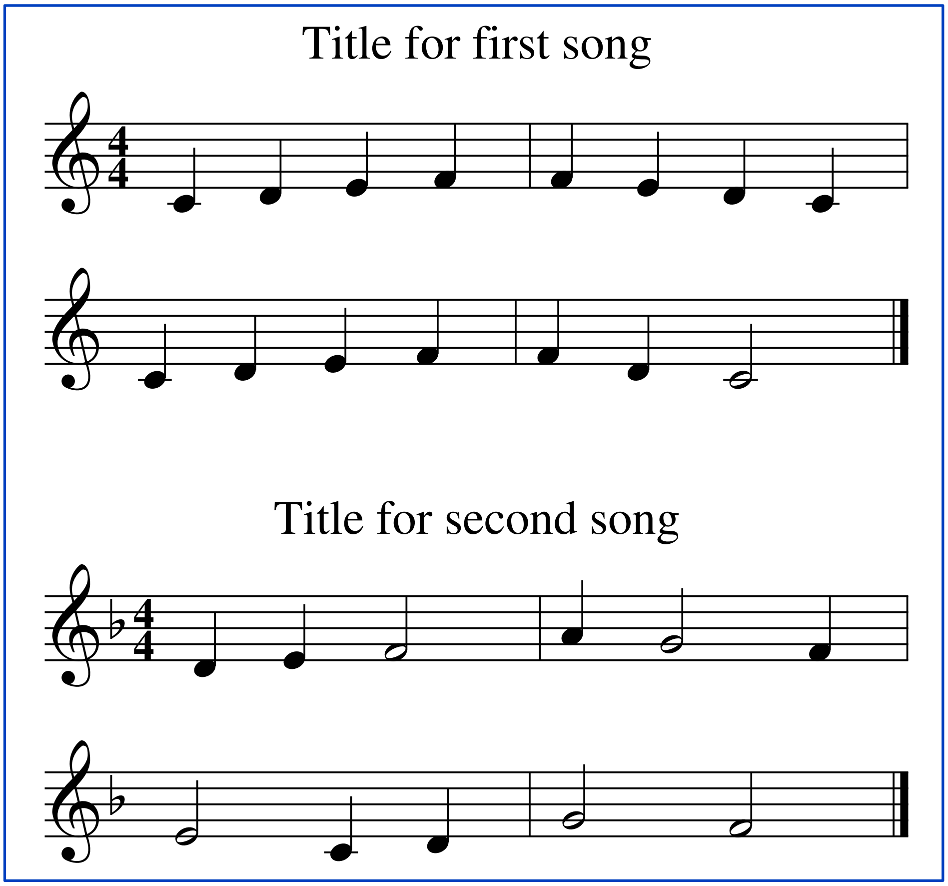


Note that if you want a piece to be played in swing time, you may also want to set the swingunit parameter, which will make the MIDI output use swing time.

15.17 Placing several songs on one page

Sometimes you may wish to print more than one song on the same page. While Mup considers all of its input to be a single song, it is possible to get the effect of separate songs. First of all, on the last bar line of first song, use "hidechanges." That way, if the key or anything is different in the next song, Mup won't print the changes at the end of the first song. Then use a "block" for printing the titles for the second song. Here is an example:

```
score
    scoresep=10,14
    label=""
header
    title (18) "Title for first song"
music
1: c;d;e;f;
bar
1: f;e;d;c;
bar
1: c;d;e;f;
bar
1: f;d;2c;
endbar

// force time signature to be printed on next score
// by changing the time, but only for an invisible measure
score time=5/4n
music
1: ms;
invisbar hidechanges

// Force the block closer to next score,
// so it will better match the spacing of
// the title of the first song.
score scoresep=6,6
// print title for second song
block
title (30) " "   // Allow some extra room above title
title (18) "Title for second song"

score
    // Set up for second song
    time=4/4
    key=1&

music
1: d;e;2f;
bar
// Put score spacing back to original
score scoresep=10,14
music
1: a;2g;4f;
bar
1: 2e;4c;d;
bar
1: 2g;f;
endbar
```

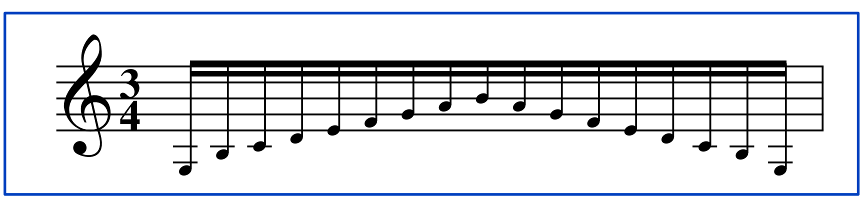


The use of saveparms and restoreparms, and/or savemacs and restoremacs may be useful in some cases, particularly if you want similar settings for several songs that are separated by songs where you want different settings.

15.18 Cadenzas

Cadenzas or cadenza-like passages are often written with an arbitrary number of notes per measure. One way to accomplish this is to use a tuplet with the appropriate number of notes.

```
score
    time = 3/4

music

1: { [cue]... 16g- bm;b-;c;d;e;f;g;a;b;a;g;f;e;d;c;b-;g- ebm;} 17n,2.;
bar
```



15.19 Transposition

The transpose and addtranspose parameters can be used to transpose the pitches for a staff or the whole score. The following table shows what transposition values to use for different intervals.

halfsteps	transposition	alternate transposition
0	perfect 1	diminished 2
1	augmented 1	minor 2
2	major 2	diminished 3
3	augmented 2	minor 3
4	major 3	diminished 4
5	augmented 3	perfect 4
6	augmented 4	diminished 5
7	perfect 5	diminished 6
8	augmented 5	minor 6
9	major 6	diminished 7
10	augmented 6	minor 7
11	major 7	diminished 8

This table continues in a similar way for intervals beyond an octave. For each number of halfsteps you want to transpose, there is more than one way the transpose and addtranspose parameters can be specified, as shown by this table. This allows to you control whether a sharp key or a flat key is to be used in cases where there is a choice. For example, if the key signature has three sharps, and you want to transpose up two halfsteps, you could say either

```
transpose = up maj 2
```

or

```
transpose = up dim 3
```

The first will result in a key of five sharps, and the second will result in seven flats. These are equivalent keys; in major for example, they are B and C flat, which are the same note. It is up to you to choose the way you would like it to be printed. But no key is allowed to have more than seven sharps or flats, so in most cases only one of the ways will result in a valid key, and the other way will result in an error message.

The transpose and addtranspose parameters are allowed in score and staff contexts, so they can be set to different values on different staffs, and on the score as a whole. This is useful for printing scores where some of the instruments are transposing instruments (like B-flat clarinet). You can set the transpose parameter on specific staffs to appropriate values for the transposing instruments. Then if you decide you want to move the entire score to a new key, you can set the addtranspose parameter in score context. You can enter all the music at true pitch, but print the score and/or the individual parts with the correct transposition.

Another use for setting different transposition values is the following trick, useful in guitar music for printing both the real chords and chords for capo. Say, for example, you have a song that is in B flat, but the guitar is to use a capo on the third fret, and play in the key of G. You could define the following macro to print both versions of a chord, the capo version above the real version:

```
define CHORD(COUNT, NAME)

    bold chord all: COUNT[-1.3] `NAME`;     // chord for capo
    ital(8) chord 1: COUNT `NAME`;    // real chord
@
```

Then, set these transpositions:

```
score   transpose = down minor 3 // transpose score for capo
staff 1 transpose = up perfect 1 // override score transposition
```

and similarly for any other staffs. Then, wherever you want to print a chord, say (for example)

```
CHORD(3,Dm)
```

This will print both versions of the chord above count 3 of the measure. For the top chord it will transpose this to Bm, the capo chord. For the bottom chord it will leave it as Dm, the real chord.

For MIDI purposes, if your MIDI player supports altered tunings, the a4freq parameter could be used to get the effect of transposing all the voices. See also the section on Concatenating macro names for another approach.

15.20 Placing verses below the scores

Sometimes, particularly with songs that have a lot of verses, you may want to put some verses of lyrics as blocks of text below the music. Probably the easiest way to do this is using "print" or "paragraph" statements inside a "block."

```
block
print (_win.w + 30, _win.n - 15) "2.   "
print "\
Here is the first line
and the second line
and the third line.
"
print (_win.w + 90, _win.n - 15) "3.   "
print "\
Here is another verse’s first line
and its second line
and its third line.
"
```

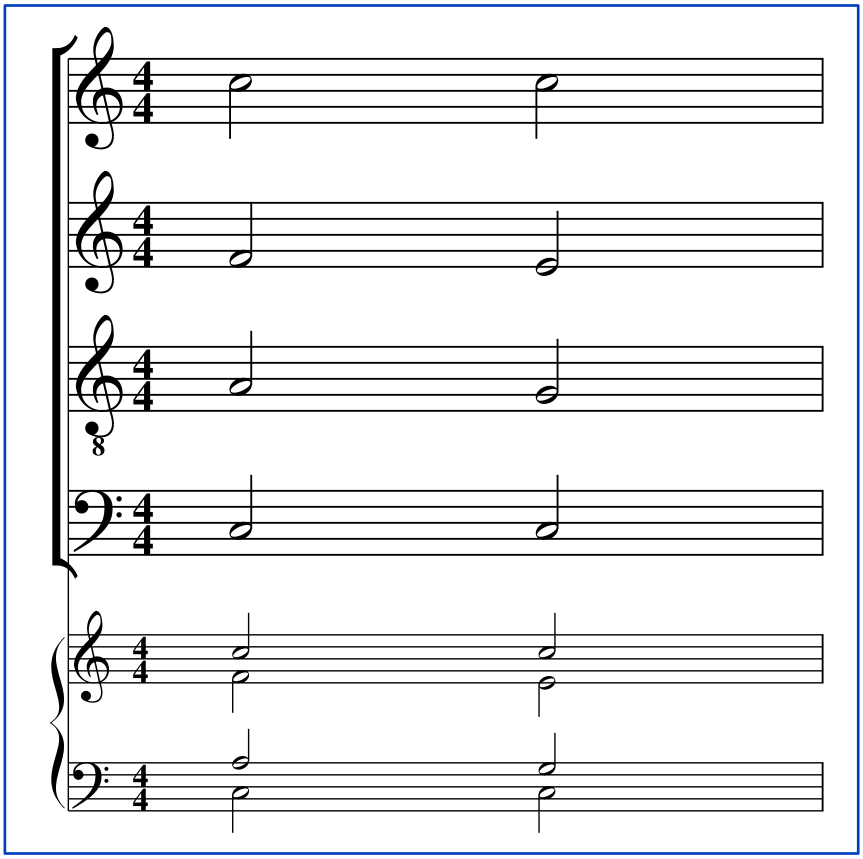
You may need to use a little trial-and-error to determine the coordinates to use on the print statements, but you only have to determine one coordinate per verse, and with a bit of practice it gets easier to find the right values. Since the "print" statement is being used, the font and size will be determined by the size parameter and font parameter, rather than by lyricssize and lyricsfont.

If you also want chord symbols with these verses, you can do that too. It is easiest if you use Courier font, as shown in the example below, since its constant-width characters make it easy to line things up. If you use a proportional-width font, you will have to determine how to place the chords by trial and error.

```
print (_win.w + 50, _win.n - 35) "3.   "
print "\f(CR)\
C            D7    G7
This is the first line
C       F
of the next verse
B\(smflat)
of the song.
"
```

15.21 Automatic piano reduction

Perhaps you’d like a four part vocal piece written on four separate staffs, as well as a piano reduction on two staffs with two voices. This could be done using:

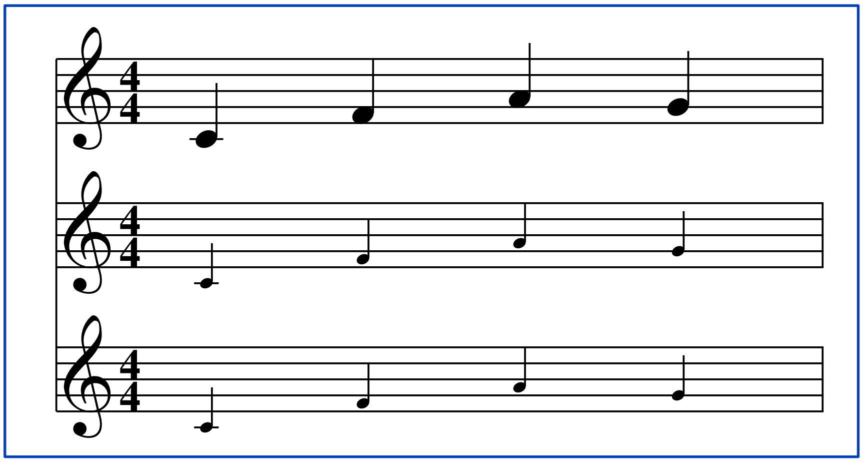
```
score
   staffs=6
   bracket=1-4
   brace=5-6
staff 3
   clef=treble8
staff 4
   clef=bass
staff 5
   vscheme=2o
   staffscale=0.75   // make piano staffs a little smaller
staff 6
   clef=bass
   vscheme=2o
   staffscale=0.75

// Define macros to put each voice on its own staff
// plus the appropriate staff/voice of the piano staffs.
define S 1 1 & 5 1: @  // soprano
define A 2 1 & 5 2: @  // alto
define T 3 1 & 6 1: @  // tenor
define B 4 1 & 6 2: @   // bass

music
S 2c+;;
A 2f;e;
T 2a;g;
B 2c;;
bar
```

This can also be done using chord-at-a-time input style:

```
// Define a macro to put each voice on its own staff
// plus the appropriate staff/voice of the piano staffs.
// This example assumes inputting notes from bottom to top.
define M [ 4 1 & 6 2; \
           3 1 & 6 1; \
           2 1 & 5 2; \
           1 1 & 5 1 ] :   @

music

M 2cafc+;cgec+;
bar
```

15.22 Deriving cue notes from another staff

Sometimes you may want to have the notes on one staff be used as cue notes on another staff, typically to help the players of one instrument know when they should come in, by showing what another instrument will be playing just before their entrance. One way to do this is to define a macro. In this example, the macro is called NAC, for "normal and cue." The first parameter is the staff/voice that should get normal sized notes, the second is the staff/voice that should get cue sized notes, and the third is the Mup input for the notes themselves. Since REG and CUE parameters could each be passed a list, if multiple instruments are playing in unison, they could all in be listed as REG, and if you wanted multiple staffs to get the same cues, they could all be listed as CUE, as shown in this example:

```
define NAC(REG, CUE, INPUT)
    REG: INPUT
    CUE: [cue]... INPUT
@

score
    staffs=4
staff 3
    // it's okay if some of the staffs are invisible
    visible=n
music

NAC(1&3, 2&4, c;f;a;g;)
bar
```



15.23 Diagonal slash marks

Sometimes, instead of a note or notes, a diagonal slash mark is printed on a staff. This is used for various purposes, such as strumming a chord (in a guitar part), or to mean that the previous beat should be repeated or that a note should be improvised. You can get slash marks for all notes by setting the noteheads parameter.

```
noteheads = "allslash"
```

or if you want to use hollow slashes for half notes and longer

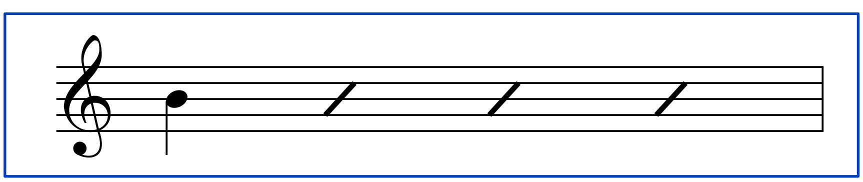
```
noteheads = "slash"
```

Usually such slash marks do not include a stem, so you may wish to also set

```
stemlen=0
```

If you just want specific chords to be slashes, you can use the chord head shape override:

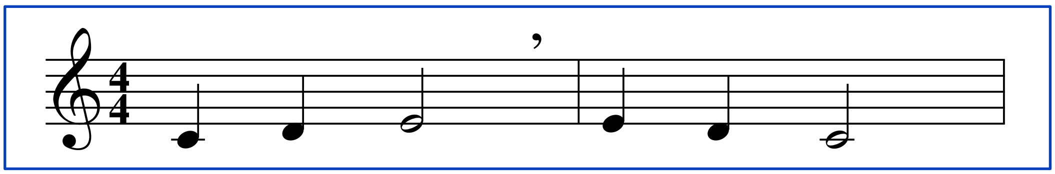
```
1: b;[ hs "allslash"; len 0 ]... ;;;
bar
```



15.24 Breath marks

A comma in a large font can be used as a breath mark. Something in the range of 20 to 30 points usually looks good; you can adjust to your preference.

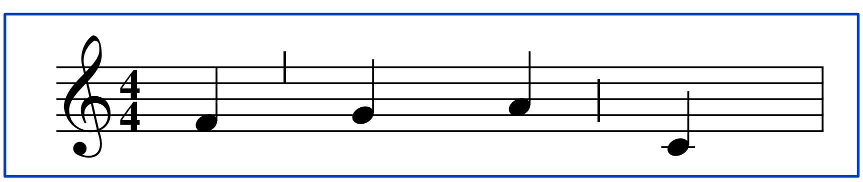
```
1: c;d;2e;
rom (24) above all dist 0: 4.5 ",";
bar

1: e;d;2c;
bar
```



15.25 Breaks

As an alternate to a breath mark, sometimes a short vertical line is drawn through the top line of the staff. Another thing that is sometimes seen is something like a short bar line, just spanning the two middle spaces. That might be used in music that wasn't originally written with true bar lines. It might also be used to subdivide a bar, when there are an odd number of beats, to show the intended grouping of beats. These could be done using macros. Here is one possible implementation. Note that you could apply to multiple staffs by setting the STAFF argument to the list of desired staffs.

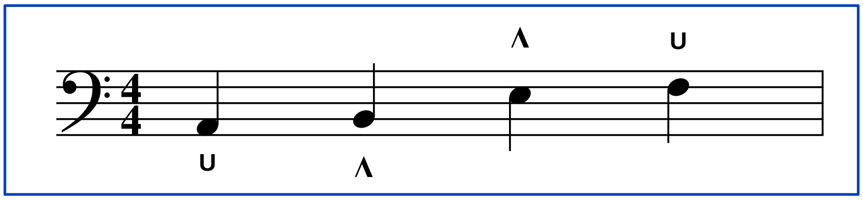
```
// short vertical line through top line of staff
define BREAK(X,STAFF)  rom above STAFF dist -2.3 ! :  X "|"; @
// line spanning two middle spaces of the staff
define SB(X,STAFF)
 rom above STAFF dist -7.2 ! : X "|";
 rom above STAFF dist -5.8 ! : X "|";
@
1: f;g;a;c;
BREAK(1.5,1)
SB(3.5,1)
bar
```



15.26 Organ pedal heel and toe marks

Mup does not include characters specially for the standard organ pedal heel and toe indications, but a U in helvetica font and the acc_hat music symbol can be used. You may want to make them a bit smaller than the default size, so defining macros for them may be useful.

```
score
    clef = bass

// define strings for the organ pedal heel and toe marks
define HEEL "\s(-3)\f(HB)U" @
define TOE "\s(-1)\(acc_hat)" @

music

1: a-;b-;e;f;
rom below 1: 1 HEEL; 2 TOE;
rom above 1: 3 TOE; 4 HEEL;
bar
```



15.27 Printing auxiliary information

Some users may like to print some auxiliary information, such as the name of the Mup file, or the time the file was last modified. That can be accomplished by passing the desired information via a macro definition. For example, on a Linux system, you could create a shell script:

mup -DINFO="$(stat --format 'Generated from file %n, last modified %y' $1)" $1

and then in Mup input files, use something like:

```
if defined(INFO) then
footer
        title `INFO`
endif
```

15.28 Repeated but non-consecutive with lists

Often, the items of a "with list" may apply to multiple chords in a measure, but not consecutive ones, so the shortcuts of [] or []... don't help. Consider, for example, the case where every other chord has an accent. Rather than repeating "[with >]" for each, a macro can be defined, to reduce the amount of typing needed:

```
define A [with >] @
1: A 8d;g-;A d&;g&-;A c;f-;A bn-;en-;
bar
```

15.29 Generating blank staff paper

It is possible to use Mup to generate blank music staff paper. You simply use an input file that contains measure spaces, each ending with an invisbar. You can control whether you want clefs, time signatures, braces, etc., and can control the spacing of staffs using the scoresep parameter. Here is a simple input that will generate a page with 8 completely blank staffs.

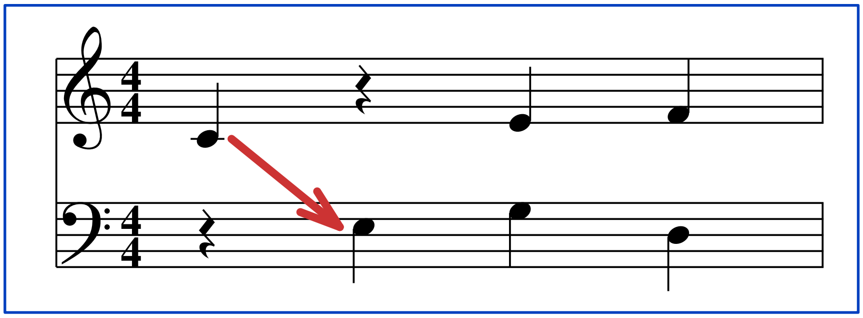
```
score
    scoresep=9,100           // spread staffs out nicely
    stafflines=5n            // don’t print any clefs
    label=""                 // make sure left edges line up both on first
    label2=""                //    and on subsequent lines
    topmargin=1              // allow extra margin to write in header/footer
    bottommargin=1
    time=4/4n                // don’t print any time signature
music

define SCORE
    1: ms;
    invisbar     // no bar line at the end of the staff
@

define SCORE_NEWSCORE
    SCORE
    newscore
@

// print 8 staffs, with newscore between each
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE_NEWSCORE
SCORE
```

15.30 Passing multiple coordinates to PostScript

Now that Mup supports exporting tag information to user PostScript code, this hint is probably mostly obsolete, but it may still be useful for demonstrating how to do some unusual things.

When you use a "postscript" section to add arbitrary PostScript code to the Mup output, you specify a current point, but sometimes it would be useful for the PostScript code to know about the location of more than one thing on the page, for example, to draw a line between two points, or to draw a box or oval around several chords. One way to do that is to have one postscript section to save away the x,y values of each point of interest, and then have a final postscript section that uses the points to draw something. Since Mup puts the contents of postscript sections inside a save/restore block, saving coordinate information for later use is a little tricky, but the following example shows one way it can be done. Earlier, in the section on macros, we showed how to draw a line with an arrow using macros and arithmetic expressions. Here we will show how to accomplish a similar thing using a postscript section.

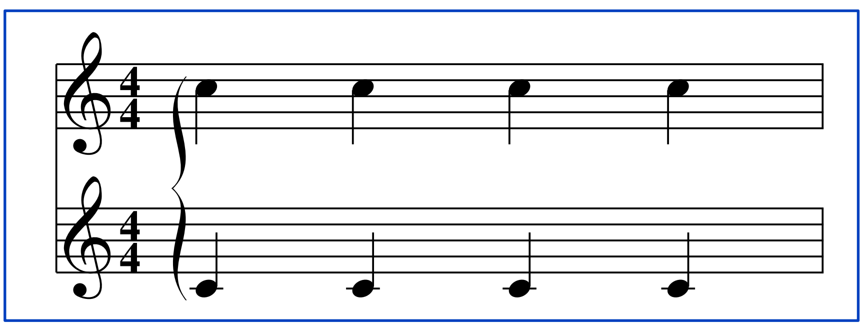
```
score
    staffs=2
staff 2
    clef=bass
music
1: c =c;r;e;f;
2: r;e =e;g;d;
// This PostScript saves the (X,Y) coordinate of a point
// near c in PostScript variables beginX and beginY
postscript (c.x + 3, c.y) "
    % Save current point on the stack
    currentpoint
    % Since Mup did a save operation, move that save object
    % to the top of the stack and do a restore,
    % leaving the currentpoint values on the stack,
    % so we can then save them in beginX and beginY.
    3 -1 roll
    restore
    /beginY exch def
    /beginX exch def
    % Push a ’save’ object for Mup’s restore to use
    save";
// This PostScript retrieves the beginX and beginY that were saved by the
// previous PostScript, along with the given current point coordinate,
// and from that, calculates and prints an arrow.
postscript (e.x - 3, e.y) "
    % Similar to above, save the specified coord in endX and endY
    currentpoint   3 -1 roll   restore
    /endY exch def   /endX exch def
    % Calculate length of the line, sin and cos to get arrowhead angle, etc.
    /fullX endX beginX sub def       /fullY endY beginY sub def
    /fulllen fullX fullX mul fullY fullY mul add sqrt def
    /cosine fullX fulllen div def    /sine fullY fulllen div def
    /headlen 15 def     /headwidth 10 def
    /headX headlen cosine mul def    /headY headlen sine mul def
    /hbX endX headX sub def          /hbY endY headY sub def
    /thicklen headwidth 2.0 div def
    /thickX thicklen sine mul def    /thickY thicklen cosine mul def
    /feathupx hbX thickX sub def     /feathupy hbY thickY add def
    /feathdnx hbX thickX add def     /feathdny hbY thickY sub def
    % Make the arrow wide, a shade of red, and with rounded ends
    gsave 3 setlinewidth 0.8 0.2 0.2 setrgbcolor 1 setlinecap
    % Draw the line and its arrowhead lines
    newpath beginX beginY moveto endX endY lineto stroke
    newpath endX endY moveto feathupx feathupy lineto stroke
    newpath endX endY moveto feathdnx feathdny lineto stroke
    grestore
    % Push a save object to match the one we undid earlier
    save";
bar
```



15.31 Printing braces

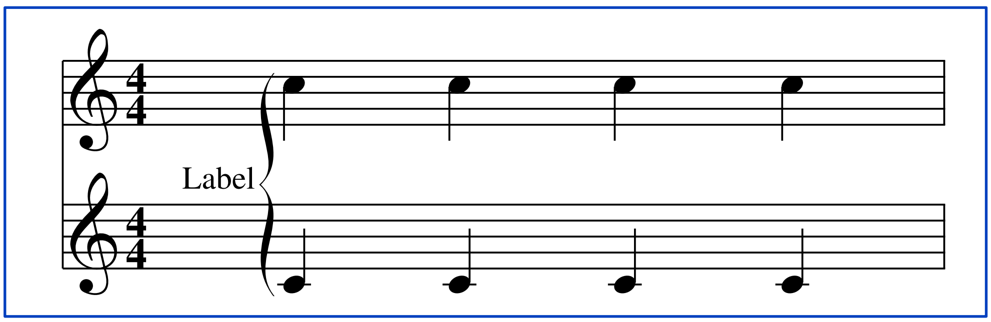
Mup prints braces to the left of scores via the brace parameter, but it may sometimes be desired to print a brace somewhere else, and have it scale appropriately for its height. This can be done using an escape to PostScript. Here is a macro definition that can be used to print a brace, and an example of how to use it:

```
// brace expects an x value and bottom and top y values
define BRACE(TOP, BOTTOM)
       postscript (TOP.w - 1, BOTTOM.s) with _top.y = TOP.n  "
           currentpoint Mup_top.y brace"
@

score
   staffs=2
music

1: c+ =_t;;;;
2: c =_b;;;;
BRACE(_t, _b)
bar
```



A somewhat more complicated version, which would print a string before the brace, might be:

```
define BRACE(TOP, BOTTOM, STRING)
       postscript (TOP.w - 1, BOTTOM.s) with _top.y = TOP.n  "
        currentpoint /yloc exch def /xloc exch def
        currentpoint Mup_top.y brace
        /TimesRoman findfont 12 scalefont setfont
        xloc 7.5 sub (" + STRING + ") stringwidth pop sub
        yloc Mup_top.y add 2 div 1.5 sub moveto (" + STRING +") show"
@

score
  staffs=2
music

1: [pad 10] c+ =_t;;;;
2: c =_b;;;;
BRACE(_t, _b, "Label")
bar
```



That still wouldn't handle things like special characters in the string, or different font sizes, but could handle simple cases, and be a starting point for more complicated ones.

15.32 Special #file and #line comments

There are two "special" kinds of comments. Since Mup takes a text file as input, it may sometimes be convenient to write a program which generates Mup input, based on some higher-level description. But then, if there are any errors found in that generated input, it may be difficult to correlate the error line that Mup outputs with what line in the original input had generated it. To help with that, the generating program could insert special comments to override what Mup uses as the current file name and line number when reporting errors or warnings.

The first special comment format is:

`//#file` *someFileName*

Any warnings or errors encountered after that line will use the given *someFileName* instead of the name of the file actually being read. The "//#file" must be exactly as shown, with no spaces. That must be followed by at least one space or tab. The entire rest of the line after those spaces/tabs will be used as the file name.

The second special comment format is:

`//#line` *N*

where *N* is a number, to be treated as the line number for the following line of input. So:

```
//#file Etude
//#line 42
Any errors on this line will be reported as being from line 42 of file "Etude"
```

Anything on the //#line line after the number will be ignored.

15.33 Using PostScript tools on Mup files

Since Mup generates PostScript, almost any PostScript tool can be used on its output. In particular, the "ps2pdf" tool that comes with Ghostscript can convert Mup output to PDF format, and the "ps2epsi" tool, which also comes with Ghostscript, converts a PostScript file to an Encapsulated PostScript (EPS) file. Many text processing and graphics programs will let you import EPS files, so this can let you insert Mup output into some other document.

There is a "psutils" package, available from most Linux archives, that contains various Postscript tools. The "psnup" program lets you print multiple pages on one sheet of paper with more flexibility than Mup's panelsperpage parameter. The "psselect" command prints a subset of pages. The "psresize" command may be useful for getting something like a3 paper size, which Mup doesn't support directly via the pagesize parameter, or a size that exceeds the range of the pageheight and pagewidth parameters.

You can check the user-donated programs page on Arkkra's website for other programs for processing Mup input or output.

16. MIDI OUTPUT

16.1 Basic information

Mup will optionally produce MIDI output (using the -m or -M command line argument). Mup is first and foremost a music publication program, so its MIDI capabilities have a few limitations. However, the MIDI output is quite useful for "proofreading" (or perhaps we should say "proof-listening"). It is often easier to spot a typo in Mup input by listening to it than to look at the output. Mup provides enough MIDI control to do virtually all of what MIDI supports, and will be adequate for many people. Others, however, may find they want a separate MIDI editor for really serious MIDI work.

The following section assumes a general knowledge of MIDI. If you are not familiar with MIDI, there are many books available on the subject at most music stores or computer book stores. There are also many online resources and tutorials. You could start at http://www.midi.org or use your favorite search engine.

Each voice is put on a separate MIDI track. The first track contains general information such as key and time signature. The next track will be for staff 1, voice 1. If staff 1 has a second voice, that will be the next track, otherwise it will be voice 1 of staff 2, if any, and so forth, one track for each voice, top to bottom.

Output is in MIDI file format 1, with a default of 120 quarter notes per minute, 192 ticks per quarter note. MIDI channel 1 is used by default for all voices. If you want to use different instrument sounds for different voices, you will need to specify a different channel for each voice, then specify the MIDI "program" for that voice. This is demonstrated in some of the examples later in this section.

Mup MIDI output will handle repeats and first and second endings, but it does not know anything about "D.S. al coda" or anything of that sort. It is possible to work around this limitation to some extent using macros. For example, a section between a "sign" and a "jump to coda" symbol could be put inside a macro definition; then the macro can be called. Then later in the piece, where the "D. S." occurs, the macro can be called again if MIDI is defined. For example:

```
// an introductory section, ends with a sign
1: c;d;e;f;
mussym above all: 5 "sign";
bar

// define macro for section between sign and
// symbol to "jump to coda"
define SECTION
1: g;a;g;;
mussym above all: 5 "coda";
bar
@

// print/play the section just defined by
// the macro
SECTION

// now do the music up to the D.S.
1: e;f;2g;
ital above 1: 1 "D. S. al Coda";
dblbar

// human player would now flip back to
// the sign, so do the MIDI equivalent:
// play that section again.
ifdef MIDI
    SECTION
endif

// now do the coda
1: e;d;2c;
rom above 1: 0 "Coda";
endbar
```

Mup mainly just outputs the note information. Mup will recognize octave marks, and move notes up or down appropriately. It recognizes piano pedal marks. It does not attempt to interpret tempo or dynamics marks specified by "rom," "boldital," etc. or ornaments in "mussym" statement. It does not handle ties to a different voice. It does interpret rehearsal marks as cue points. It handles grace notes, rolls, slashes, and alternation groups. You can control how legato the music is by using the Mup "release" parameter.

Some styles of music are often written in "swing time," meaning the players are expected to play pairs of notes with the first twice as long as the second, even though they are written as if they were the same duration, or as if the first were three times as long as the second. The most common example would be where the written notation shows two eighth notes like 8;; or a dotted rhythm like 8.;16; but the musician "knows" that the composer really intended it to be played as if it were a triplet {4;8;}3; The swingunit parameter can be used to get Mup MIDI output to automatically follow that performance convention.

A separate MIDI editing program may be useful for adding really complicated effects, but it is possible to specify MIDI directives to do almost anything you want. They are of the form:

midi *S V*: *begintime "keyword=value";*

The *S* and *V* specify the staff and voice for which the directive is to apply. As elsewhere in Mup, an omitted voice will default to voice 1, and both staff and voice can be given as a list. Certain keywords apply to the entire score. In that case the form

midi all: *begintime "keyword=value";*

is used instead. The items specified using "all" are placed on the first track, the track containing score-wide information. They are not applied to the voices on the other tracks.

The *begintime*, as elsewhere in Mup, gives the beat into the measure where the MIDI output is to be placed. If notes are to be turned on or off at the same instant in time as the *begintime*, first all "note off" commands are generated, then the "midi" command events, then "note on" commands.

The *keyword=value* gives specific information of what MIDI output to generate. The following keywords are currently supported:

keyword	values	meaning	midi S V	midi all
program	0-127	program change (new instrument)	yes	no
parameter	0-127,0-127	parameter	yes	yes
channel	1-16	channel	yes	no
chanpressure	0-127	channel pressure (after touch)	yes	yes
tempo	10-1000	tempo, quarter notes per minute	no	yes
seqnum	0-65535	sequence number	yes	yes
text	text	text meta event	yes	yes
copyright	text	copyright notice	yes	yes
name	text	sequence/track name	yes	yes
instrument	text	instrument name	yes	yes
marker	text	marker meta event	yes	yes
cue	text	cue point	yes	yes
port	0-127	MIDI port	yes	yes
onvelocity	1-127	note on velocity	yes	no
offvelocity	0-127	note off velocity	yes	no
hex	hex data	arbitrary MIDI data	yes	yes

The keywords can be abbreviated to their first three or more letters, except "chanpressure" which requires at least five letters to differentiate it from "channel" ("cha" or "chan" will be interpreted as channel). In most cases, the "=" is followed by either a number or some text. Exceptions to this are discussed in the next few paragraphs.

The "parameter" keyword is followed by two numbers, separated by a comma. The first is the parameter number, the second is the parameter value. Thus to set parameter 7 (which is the volume parameter) to 90 for voice 2 of staff 3, starting at the beginning of the measure, you can use:

```
midi 3 2: 0 "parameter=7,90";
```

The "onvelocity" and "offvelocity" keywords can have one or more values, separated by commas. If there is only one value, it applies to all notes in each chord. If there is more than one value, the first value applies to the top note of the chord, the second value to the second-from-the-top note, and so forth. If there are more notes in a chord than there are values specified, the last value specified applies to all of the remaining notes. So, for example, if you want to emphasize the top note of each chord because it is the melody, you can specify two values, as in

```
midi 1: 1 "onvelocity=76, 60";
```

which would cause the top note to have a velocity of 76 and all other notes to have a velocity of 60.

The "hex" form can be used to insert any arbitrary MIDI data into the MIDI file. The value consists of any even number of hexadecimal digits. Spaces and tabs can be included in the value field for readability.

Note that Mup uses the MIDI standard, which numbers instruments from 0 through 127, but some MIDI playback programs follow a convention of numbering them from 1 through 128.

Here are some examples:

```
midi all: 0 "tempo=72";
midi 1-2 1-2: 0 "channel=2"; 0 "program=14"; 3.5 "program=76";
midi all: 3 "hex= ff 00 02 00 01";   // sequence number 1
midi 3,6: 0 "channel=5"; 0 "prog=15"; 0 "instr=dulcimer";
// set parameter 7 (usually volume) to 100
midi 2: "par = 7, 100";
```

Here is a more extensive example of how midi commands might be used in a song:

```
score
    staffs=2
    vscheme=2o

staff 2
    clef=bass

voice 2 2
    // Make the bottom voice more staccato
    release=50

music

// Set the tempo.
// Start out at 108 quarter notes per minute,
// but on count 4, slow down to 96 per minute.
midi all: 0 "tempo=108"; 4 "tempo=96";

// Put each voice on a different channel
// using a different instrument sound.
// The program numbers correspond to the
// General MIDI sounds as noted.
midi 1 1: 0 "channel=1"; 0 "program=68";  //oboe
midi 1 2: 0 "channel=2"; 0 "program=11";  //vibraphone
midi 2 1: 0 "channel=6"; 0 "program=60";  //french horn
midi 2 2: 0 "channel=4"; 0 "program=35";  //fretless bass

// Make the top voice louder, and put an
// accent on the third beat
midi 1 1: 0 "onvelocity=86"; 3 "onvelocity=100"; 4 "onvel=86";

// Set maximum reverb on french horn part,
// starting at the second beat.
// (Reverb is parameter 91)
midi 2 1: 2 "parameter=91, 127";

// Set chorus on oboe to 75, from the beginning.
// (Chorus is parameter 93)
midi 1 1: 0 "param=93, 75";

// Pan the bass part to middle of left side.
// (Pan is parameter 10, with a value of 0 being hard left,
// 64 in the center, and 127 being hard right, so 32 is
// half way to the left.)
midi 2 2: 0 "parameter=10, 32";

// Now the music to be played...
1 1: e;d;2c;
1 2: c;b-;2g-;
2 1: g;f;2e;
2 2: c;g-;2c;
bar
```

16.2 Gradual MIDI changes

Often you may like to have gradual changes in things like tempo or volume or onvelocity. You can, of course, tell Mup exactly what you want for each individual note, but you can also just specify beginning and ending values and optionally some intermediate values, and Mup will do interpolation, creating as many midi commands as necessary to produce smooth gradual changes.

As a common example, suppose you want to do a ritard. You could do something like:

```
midi all: 3 "tempo=120 to 96" til 1m+4;
```

This would start the tempo at 120 quarter notes per minutes at beat 3 of the current measure and ritard to 96 per minutes at beat 4 of the following measure. If only two values are given, as in that example, the change is done linearly, but you can also specify multiple points that make up a curve. The curve is divided into equal segments: if there are three points, it will be treated as two equal segments; if four points, as three segments, etc. As an example, you could slow down and then speed back up:

```
midi all: 2 "tempo = 112 to 92 to 112" til 2m+4;
```

or increase onvelocity slowly at first and then more so:

```
midi 1: 1 "onvelocity= 40 to 50 to 70 to 105" til 4.5;
```

Changes in time signature are not allowed during a single gradual change, because it may not be entirely clear what is wanted in that case. So you have to tell Mup what you want by starting a new gradual change at each time signature change.

Note that all midi commands must either include both a "to" and a "til" or neither. The to/til are only allowed on midi commands where the values are numbers, namely channel, chanpressure, offvelocity, onvelocity, parameter, port, program, and tempo. Note that a few of those may be of rather dubious usefulness, particularly channel and port, and perhaps program. For a parameter, the parameter number is specified just once, as in:

```
midi 1: 1 "parameter=7, 40 to 60 to 70 to 90 to 65" til 4.5;
```

In the case on onvelocity or offvelocity, the items between "to" can be lists that work like those without "to," with the first applying to the top note, and so forth. The number of comma-separated items does not need to be the same in all lists, since the last value in each list will apply to any remaining notes. So to play a C major chord, accenting each note in turn from bottom to top, you could do:

```
midi 1: 1 "onvelocity=60,60,60,120 to 60,60,120,60 to 60,120,60 to 120,60" til 4;
1: cegc+;;;;
bar
```

17. CONCLUSION

The Mup program provides a convenient way to produce high-quality musical scores. We hope you enjoy using it. Appendix A gives a sample input file, demonstrating many of the features of Mup. You can download additional sample songs from the Arkkra Enterprises website listed below. The website also includes several programs written by Mup users that may make Mup even more useful for you. Once you've used Mup for a while, you may find the Quick Reference helpful for remembering details of the Mup language.

If you have comments or questions, please contact:

Arkkra Enterprises

support@arkkra.com

http://www.arkkra.com

Appendix A

Example of a Mup input file

```
//!Mup-Arkkra

// sample of a Mup input file

header
    title (18) "The Star Spangled Banner"
    title ital (12) "(The United States National Anthem)"
    title (9) "Text: Francis Scott Key" "Tune: J. S. Smith"
    title (9) ""  "arr: William J. Krauss"

score
    topmargin = 0.5
    bottommargin = 0.5
    leftmargin = 0.65
    rightmargin = 0.65
    scale = 0.65
    packfact = 1.3
    key = 3#
    time = 3/4
    staffs = 2
    brace = 1-2
    barstyle = 1-2
    measnum = y
    vscheme = 2o
    beamstyle = 4,4,4
    endingstyle = top

staff 2
    clef = bass

define LYRICS lyrics between 1&2: @

music
    // Starts on a pickup, so use space
    // at beginning of measure.
    // Voices are in unison for the first
    // few notes, so specify two voices
    // at once.
    1 1-2: 2s; 8.e<>; 16c;
    2 1-2: 2s; 8.e<>; 16c;
    LYRICS 2s; 4;    [1] "Oh_";
    // Since the first two sections are
    // almost the same, use a repeat
    repeatstart

    1 1: a-; c; e;
    1 2: a-; c; b-;
    2 1: a-; a; b;
    2 2: a-; a; g;
    LYRICS      [1] "say. can you";  \
            [2] "stripes and bright";
    bar

    1 1: 2a; 8.c+; 16b;
```

```
1 2: 2c; 8.; 16;
2 1: 2a; 8.g; 16;
2 2: 2f; 8.e#; 16;
LYRICS      [1] "see, by the";  \
        [2] "stars, through the";
bar

1 1: a; c; d#;
1 2: c; ; b-;
2 1: a; ; ;
2 2: f; ; b-;
LYRICS      [1] "dawn's ear-ly";  \
        [2] "per-il-ous";
bar

1 1: e; r; 8e; ;
1 2: b-; r; 8b-; ;
2 1: g; r; 8g; ;
2 2: e; r; 8e; ;
LYRICS      [1] "light what so";  \
        [2] "fight, o'er the";
bar

1 1: 4.c+; 8b; 4a;
1 2: 4.e; 8; 4;
2 1: 4.a; 8g; 4a;
2 2: 4.a-; 8b-; 4c;
LYRICS      [1] "proud-ly we";  \
        [2] "ram-parts we";
bar

1 1: 2g; 8.f; 16g;
1 2: 2e; 8.; 16;
2 1: 2b; 8.; 16;
2 2: 2e; 8.d; 16;
LYRICS      [1] "hailed, At the";  \
        [2] "watched, were so";
bar

1 1: a; ; e;
1 2: e; ; ;
2 1: a; ; e;
2 2: c; ; e;
LYRICS      [1] "twi-light's last";  \
        [2] "gal-lant-ly";
bar ending "1."

1-2 1-2: c; 8a-; r; 8.e; 16c;
LYRICS      [1] "gleam-ing, whose broad";
repeatend ending "2."

1 1: c; 8a-; r; 8.c+; 16;
1 2: c; 8a-; r; 8.e; 16;
2 1: c; 8a-; r; 8.a; 16;
```

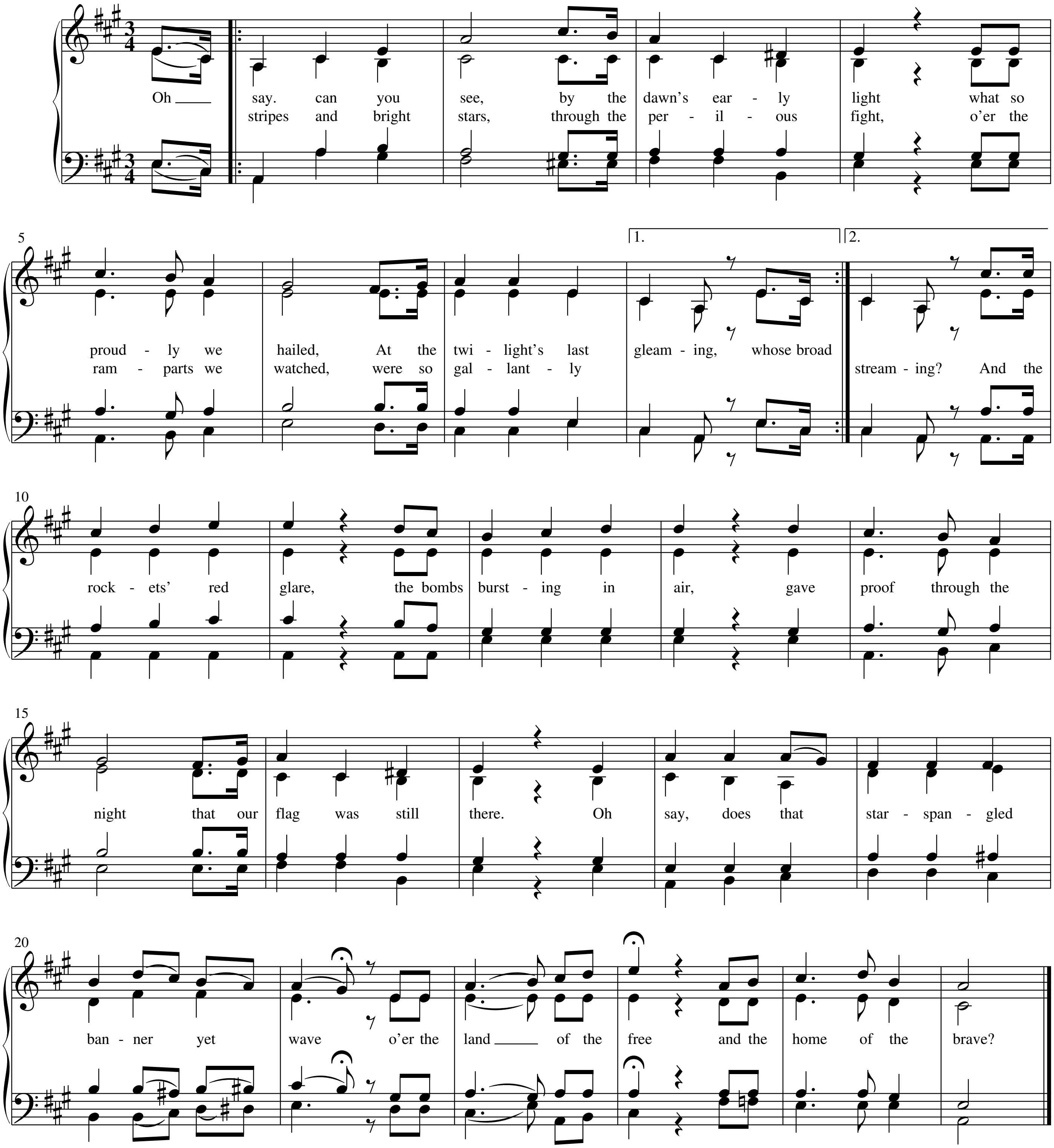
```
    2 2: c; 8a-; r; 8.a-; 16;
    LYRICS      [2] "stream-ing? And the";
    bar endending

    1 1: c+; d+; e+;
    1 2: e; ; ;
    2 1: a; b; c+;
    2 2: a-; ; ;
    // continue to mark as verse 2 from now on,
    // so that the lyrics will line up properly
    // with the lyrics of the second ending.
    LYRICS      [2] "rock-ets' red";
    bar

// Define a mapping, and use the chord-at-a-time input method for the
// next several measures.  Specify the voices in bottom to top order.
define M  [ 2 2; 2 1; 1 2; 1 1 ]  @

    M: a-c+ee+; rrrr; 8a-bed+; a-aec+;
    LYRICS      [2] "glare, the bombs";
    bar

    M: egeb; egec+; eged+;
    LYRICS      [2] "burst-ing in";
    bar

    M: eged+; rrrr; eged+;
    LYRICS      [2] "air, gave";
    bar

    M: 4.a-aec+; 8b-geb; 4caea;
    LYRICS      [2] "proof through the";
    bar

    M: 2ebeg; 8.ebdf; 16ebdg;
    LYRICS      [2] "night that our";
    bar

    M: faca; facc; b-ab-d#;
    LYRICS      [2] "flag was still";
    bar

    M: egb-e; rrrr; egb-e;
    LYRICS      [2] "there. Oh";
    bar

    1 1: 4a; ; 8<>; g;
    1 2: 4c; b-; 4a-;
    2 1: e; ; ;
    2 2: a-; b-; c;
    LYRICS      [2] "say, does that";
    bar

    1 1: f; ; ;
```

```
1 2: d; ; e;
2 1: a; ; a#;
2 2: d; ; c;
LYRICS      [2] "star-span-gled";
bar

1 1: 4b; 8d+<>; c+; b<>; a;
1 2: 4d; 4f; ;
2 1: b; 8<>; a#; b<>; b#;
2 2: b-; 8<>; c; d<>; d#;
LYRICS      [2] "ban-ner yet";
bar

mussym(12) 1-2: 2 "ferm";
1 1: a<>; 8g; r; e; ;
1 2: 4.e; 8r; e; ;
2 1: c+<>; 8b; r; g; ;
2 2: 4.e; 8r; d; ;
LYRICS      [2] "wave_ o'er the";
bar

1 1: 4.a<>; 8b; c+; d+;
1 2: 4.e<>; 8; ; ;
2 1: 4.a<>; 8g; a; ;
2 2: 4.c<>; 8e; a-; b-;
LYRICS      [2] "land_ of the";
bar

mussym(12) 1-2: 1 "ferm";
1 1: e+; r; 8a; b;
1 2: e; r; 8d; ;
2 1: a; r; 8a; ;
2 2: c; r; 8f; fn;
LYRICS      [2] "free and the";
bar

1 1: 4.c+; 8d+; 4b;
1 2: 4.e; 8; 4d;
2 1: 4.a; 8; 4g;
2 2: 4.e; 8; 4;
LYRICS      [2] "home of the";
bar

1 1: 2a; 4s;
1 2: 2c; 4s;
2 1: 2e; 4s;
2 2: 2a-; 4s;
LYRICS      [2] "brave?";
endbar
```

The Star Spangled Banner

(The United States National Anthem)

Text: Francis Scott Key

Tune: J. S. Smith
arr: William J. Krauss

INDEX