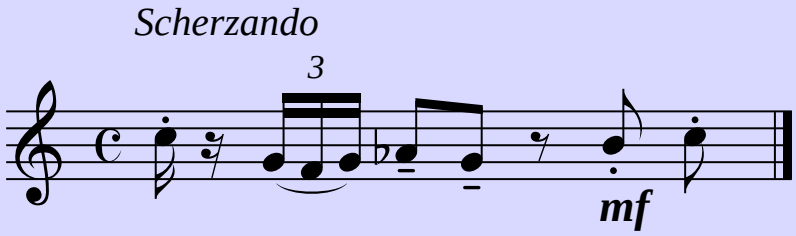
Guido Gonzato

Making Music with ABC Plus

v. 1.1.0



A guide to the notation and its applications

In loving memory of two people who were so dear to me:

Annarosa, who introduced me to music,
and my dad Bruno, for his criticism and encouragement;

and to my son Lorenzo, who will become a better musician than I am.

Contents

List of Tables

List of Figures

About This Book

This manual explains how to make beautiful sheet music and MIDI files using a computer, some free software, and the ABC PLUS music notation.

It's aimed at musicians with some computer expertise who don't want to spend a lot of money on commercial music software. Both folk and classical musicians may benefit from this guide, not to mention music teachers!

This manual comes in printed and electronic versions; the latter is accompanied by a few MIDI files. Just like the software that is used to make the music, this manual is free and can be freely copied and shared.

I hope you will find my work useful and enjoyable.

Cheers,

Guido Gonzato `=8-)`

Part I
Computer Music with ABC PLUS

1 Introduction

If you are a musician and can use a computer, you are lucky. First of all, because you are a musician; secondly, because the computer is a precious tool for writing music. Lots of programs are available.

Most music notation programs have a *graphical* approach: one or more staves are displayed on the screen, and the user edits notes on them with the mouse. An alternative approach is writing music using a *textual notation*. This is a non-graphical mode that represents notes and other symbols using characters. A specialised program then translates the notation into 'normal' sheet music in some electronic format (e.g. in PDF) and/or into a MIDI file.

Graphical programs are easier for beginners and more intuitive, but textual notations make for faster transcription and have other advantages.

Many textual notations have been invented. ABC, introduced by Chris Walshaw in 1991, is one of the best: since it is easy to learn and very powerful, it has gained widespread popularity. Thousands of tunes written in ABC are available on the Internet: in fact, this notation is the *de facto* standard among folk musicians.

ABC was later expanded to provide page layout details, MIDI commands, and multiple voices (polyphony). In this guide, this extended notation will be informally called ABC PLUS. The purpose of this guide is to introduce the reader to ABC PLUS and the most important features of its related programs. Ideally, people who could benefit from this manual are:

- folk musicians who would like to learn as little ABC as necessary to understand the files they find on the net. These people can skip the part about harmony, and probably do not need to study this guide thoroughly;
- classical musicians who would like to use ABC PLUS for typesetting their scores.

In both cases, if you wish to print sheet music for your choir or band, or make a song book, or perhaps just teach music, you have found the right tool!

A note to readers who already know ABC: in this guide I will shamelessly break the standard. 'Pure' ABC cannot do classical music, not even a couple of measures for piano! This guide encourages the adoption of existing extensions to the standard. It should be emphasised that ABC PLUS is fully upwards compatible with ABC.

1.1 Requirements

I assume that you have a PC with Windows, Mac OS/X, GNU/Linux or other Unix variants, and that you are reasonably familiar with computers. Knowledge of the Windows or Unix command line commands is not required. It is required, however, that you can read music: the treble clef and two octaves starting at middle C might suffice.

1.2 Software

The official ABC PLUS web site is http://abcplus.sourceforge.net, where programs and this manual can be freely obtained. The web site for the original ABC notation is http://www.walshaw.plus.com/abc/, where you will find links to tune collections, software, and documentation.

The most complete program for converting ABC PLUS into sheet music is currently `abcm2ps`, a free piece of software available under the GNU GPL license. `abcm2ps` is very powerful, and it creates beautiful, fully customisable sheet music.

`abcm2ps` reads ABC PLUS files and converts them to POSTSCRIPT. This is a format related to PDF, and it can be viewed and printed with another free program: Ghostscript. This application converts POSTSCRIPT files into several formats, among which the most important is Acrobat PDF.

The author of `abcm2ps`, an organ player and programmer called Jean-François Moine, releases 'stable' and 'development' versions of his program. The latter is updated almost weekly, and has many enhancements. As of this writing, the latest stable release is 4.12.30, the development release is 5.7.1.

Another very useful program is `abc2midi`, which converts ABC PLUS files to MIDI files. It is part of the `abc2midi` package, and its usage will be described in Part V of this guide.

Since ABC PLUS files are plain text files, an essential tool for writing them is a good text editor. Countless free editors exist, some of which have facilities for editing ABC and/or ABC PLUS files.

There also exist graphical applications that save music in ABC format, some of which are very nice. These programs make it unnecessary to learn the ABC PLUS notation using this manual—at least the basics. Therefore, I will not describe them here.

`abcm2ps` and `abc2midi` are *command-line driven* programs: in short, you cannot start them double-clicking on their icons. To use these programs, you have to open a command shell (Windows Command Prompt or Unix terminal) and type commands.

Scared off? Well, the command line can be avoided. There exist programs that gather all relevant pieces of software in a single integrated environment:

- ABCexplorer and ABCedit, only available for Windows;
- JEDABC, primarily designed for Unix but also available on Windows;
- `runabc.tcl`, available on all platforms.

I feel that none of this programs is 'perfect'; however, I would suggest that Unix users adopt JEDABC or `runabc.tcl`, while Windows users will probably prefer ABCexplorer or ABCedit. There is also a handy online converter: `http://www.folkinfo.org/songs/abcconvert.php`. It works as a convenient interface to `abcm2ps` and `abc2midi`.

1.3 Motivation

I know from experience that graphical programs (nearly all are commercial software) are considered easier to use than non-graphical ones.[1] So, why should one learn ABC PLUS?

Well, compared to graphical programs ABC PLUS has many *advantages:*

ABC compatibility: ABC PLUS inherits all ABC features: small and readable files, easy searching and indexing of tunebooks, easy creation of music archives, etc;

text only: the importance of this feature can't be overemphasised. Being simple text, ABC PLUS music can be read and written by any computer system on Earth; can be used by visually impaired people, be sent by phone as SMS, scribbled down on beermats,...

power: with ABC PLUS you can create scores from very simple to highly sophisticated;

ease of use: ABC PLUS is easy to learn, and after a little practice it becomes very intuitive;

[1] erroneously, but only expert users realize it.

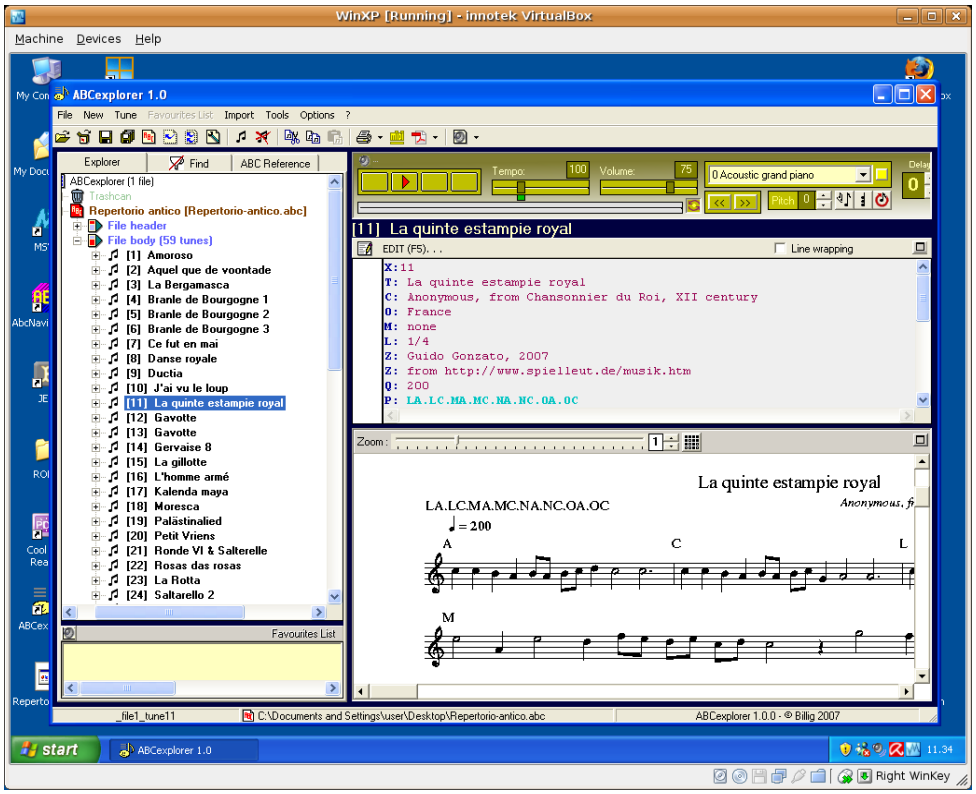


Figure 1: Managing a tune collection with ABCexplorer.

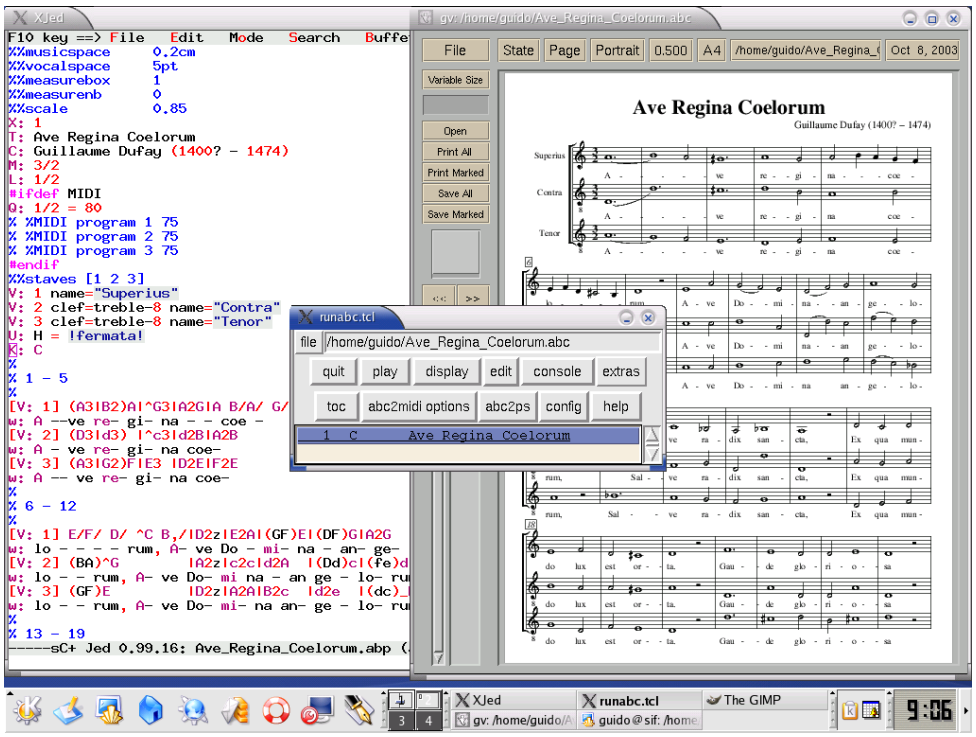


Figure 2: Writing a tune with JEDABC and `runabc.tcl`.

quality: the quality of sheet music you obtain using ABC PLUS tools is excellent;

price: while commercial software is often expensive, most programs for making ABC PLUS music are free, and can be freely copied and shared with your friends or students;

low resources: ABC PLUS programs are very compact and can run on old computers, or even handhelds;

portability: music is created as POSTSCRIPT or PDF and MIDI files instead of proprietary file formats. This way, you can share your music with everybody, not only people who have the software to produce it;

convenience: inserting scores made in ABC PLUS in web pages or word processor files is easily done. Moreover, ABC PLUS is often more flexible and easier to use;

speed: writing music in ABC PLUS is *much* faster than using any graphical program;

learning value: if you teach music, ABC PLUS in an invaluable tool that facilitates the learning of music theory;

fun: in my humble opinion, writing music in ABC PLUS is more fun!

And now, to be objective, let us look at possible *disadvantages:*

learning curve: while a graphical program may allow you to get started right away (at least in theory), ABC PLUS requires that you study a little before you can get started;

file conversion: if your work environment forces you to use a specific commercial program, you may find it difficult to convert existing music files to ABC PLUS and vice versa;

limitations: ABC PLUS is currently not capable of dealing with some types of music. Examples include Gregorian chant, percussions, and non European music.

To overcome the first hurdle, this guide is hopefully a good start; but I also recommend that you look at some scores to see real-life examples of ABC/ABC PLUS in action. The ABC home page has many links to ABC collections.

Also, there is little reason to be forced to use other music software (often an illegal copy), given the many advantages of switching to ABC PLUS. Read this manual and see what ABC PLUS can do for you. You will not regret it!

1.4 How You Do It

In order to write music with ABC PLUS, you follow these steps:

1. using an editor, write the tune in ABC PLUS;
2. convert the tune using `abcm2ps`, creating a POSTSCRIPT file;
3. view the POSTSCRIPT file with Ghostscript;
4. convert the POSTSCRIPT file into PDF format;
5. create a MIDI file with `abc2midi`;
6. finally, if the music you wrote is free from copyright, publish it for others to enjoy!

1.5 Installing the Programs

These are the home page of the programs:

- the abcm2ps typesetter:
  http://moinejf.free.fr
  http://abcplus.sourceforge.net/#abcm2ps
- abc2midi:
  http://ifdo.pugmarks.com/~seymour/runabc/top.html
  http://abcplus.sourceforge.net/#abcMIDI
- Ghostscript and ghostview:
  http://www.cs.wisc.edu/~ghost
  A rather old but convenient version is Ghostscript 5.50 along with Ghostview 2.7:
  ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/aladdin/gs550/gsv27550.exe
  However, if you wish to convert the scores to PDF files you will need the latest release of Ghostscript from this directory:
  http://mirror.cs.wisc.edu/pub/mirrors/ghost/AFPL/
  and the latest GhostView from:
  http://mirror.cs.wisc.edu/pub/mirrors/ghost/ghostgum/
- ABCexplorer:
  http://abc.stalikez.info/abcex.php
- abcedit:
  http://www.abcedit.tk/
- JEDABC:
  http://abcplus.sourceforge.net/#JedABC
- runabc:
  http://ifdo.pugmarks.com/~seymour/runabc/runabc.html

Linux users will find .rpm packages on the ABC PLUS web site. Besides, all Linux distributions include Ghostscript and the gv viewer.

I assume that Unix and MAC OS users will be able to install the programs. On Windows, download abcm2ps and abc2midi, extract the .exe files from the archives, then copy them to C:\WINDOWS (Windows 9x/XP) or C:\WINNT (Windows 2000). Ghostscript and GhostView install like all other Windows applications.

This guide will only describe the most important features of abcm2ps and abc2midi.

1.6 ABC PLUS in a Nutshell

ABC PLUS music is written in text files with extension .abc or .abp The extension is not obligatory, but I recommended that you employ it.

ABC PLUS uses the characters you find on standard computer keyboards to represent notes and symbols:

- characters like A B C a b c z represent notes and rests;
- accidentals, ties, slurs etc. are written with characters like = _ - ( ) and so on;
- expression symbols are notated with words like !fff!, !fermata!, !tenuto!, ...

- the metre, clef, title, and other tune properties are written in words called *fields*, like `M:`;
- low-level details (that is, formatting parameters or MIDI commands) are written using *commands* like `%%titlefont` or `%%MIDI program 19`.

In short: all musical features are written with sequences of characters.

A tune consists of two parts: a *header* and a *body*, which are written in an ABC PLUS *file*. The header contains information about the tune such as the title, author, key, etc. ; these pieces of information are written in fields. The body of the tune contains the music.

An ABC PLUS file may contain several tunes, separated by one or more blank lines. Each tune has its own header and body. Some fields can also appear in the body. The file containing ABC PLUS music is also called the *source* of the tunes.

Strictly speaking, commands for low-level details are not part of the notation. In fact, ABC was designed for a high-level description of tunes, with no special instructions for typesetting or sound output. That said, ABC PLUS provides commands for specifying such details; these will be examined in Section 5.

If you don't have a US keyboard, some important characters may be missing. To obtain these characters under Windows, turn the numeric keypad on, keep the `Alt` key pressed and type some digits:

- `Alt-126` to get the *tilde* `~`;
- `Alt-009` to get a *tab character*;
- `Alt-096` to get an *inverted accent* `` ` ``.
- `Alt-123` to get an *open curly brace* `{`;
- `Alt-125` to get a *closed curly brace* `}`;
- `Alt-091` to get an *open square brace* `[`;
- `Alt-092` to get a *closed square brace* `]`.

1.7 Our First Score

When the programs are installed, you are ready to write your first score.

As our first example, we shall write the C major scale and study the source carefully. In the following, I'll use the Windows command prompt; Unix details will be highlighted.

From Start/Run..., insert `cmd`, then click on 'Ok'. A black window will open showing a line that reads something like:

```
C:\Documents and Settings\user>_
```

The first time, and only the first time you use the command prompt, write this command:

```
C:\Documents and Settings\user>md \abcmusic
```

which creates a folder, called `abcmusic`, where we will store our ABC PLUS tunes. Type:

```
C:\Documents and Settings\user>cd \abcmusic
C:\abcmusic>_
```

which moves to the folder `abcmusic` you've just created. Note that the command prompt changed to confirm you've moved to that folder. Whenever you're going to write tunes, you'll have to move to the `abcmusic` folder beforehand.

Launch the `notepad` editor:

```
C:\abcmusic>notepad scale1.abc
```

Clic on 'Yes' to create a new file, then copy this source *verbatim*:

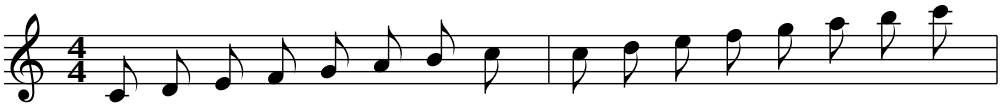
```
X: 1 % start of header
K: C % scale: C major
C D E F G A B c | c d e f g a b c' |
```

Save it, then switch back to the command prompt. We are ready to make the POSTSCRIPT output using `abcm2ps`:

```
C:\abcmusic>abcm2ps -c -O= scale1.abc
abcm2ps-4.12.30 (May 28, 2007)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 17467 bytes)

C:\abcmusic>_
```

To display the POSTSCRIPT file, launch GhostView and open the `scale1.ps` file in `C:\abcmusic`. It will look like this:



To convert the score into PDF, select File/Convert... in GhostView, set `pdfwrite` as output type, and choose a dpi resolution; I suggest 600 dpi.

Unix and MAC OS X users will use the command `ps2pdf scale1.ps`.
Probably, an environment variable needs to be set in order to get A4 paper format from `ps2pdf`. Insert this line in `/etc/profile`:

```
export GS_OPTIONS="-sPAPERSIZE=a4"
```

If you use the `gv` viewer, remember to set the correct paper format, otherwise the page margins will be displayed incorrectly.

Now, let us make a mistake on purpose: insert the `#` character instead of the first bar `|`. Save and try to convert; you will get this error message:

```
C:\abcmusic>abcm2ps -c -O= scale1.abc
abcm2ps-4.12.30 (May 28, 2007)
File scale1.abc
Error in line 3.16: bad character
   3 C D E F G A B c # c d e f g a b c' |
                     ^
```

```
Output written on scale1.ps (1 page, 1 title, 17444 bytes)

C:\abcmusic>_
```

Fix the error in the source and convert it again.

Let us now examine what we wrote in the source. It starts with two fields in the header: `X:`, index, and `K:`, key; note the upper-case letters. These are the only obligatory fields. `X:` is always followed by a number, which is used to identify the tunes written in a file. The `%` character begins a comment; everything that follows `%` till the end of the line is ignored.

Optionally, fields may be contain spaces. `X:1` and `X: 1` are equivalent.

The `K:` field specifies the tune key; `C` stands for 'C major'. In some countries, notes are written as 'do re mi fa sol la si'; if you live in such a country, you may want to consult Table 1 that compares notes written in English and in Italian notation.

Italian note	English note
Do	C
Re	D
Mi	E
Fa	F
Sol	G
La	A
Si	B

Table 1: Comparison between note names in different notations.

The `X:` field must be *the first* in the header, while the `K:` field must be *the last*. Other fields may be inserted in any order between `X:` and `K:`.

The next line in the source contains the notes. Upper-case letters correspond to the central octave, while lower-case letters one octave higher. The `|` character inserts a measure bar, which can be entered at any position. That is, you may write measures of variable length, more than or less than the value specified by the metre.

The last `c` is followed by an apostrophe, which denotes an octave higher. Note that `abcm2ps`, in absence of other indications, automatically set the metre as four quarters, and the note length as eighths.

It wasn't difficult, was it? We are now ready to study all the details that will enable us to typeset beautiful scores.

If you use 'do re mi…', the main hurdle is getting used to 'C D E…' This is not obligatory, because JEDABC lets you insert Italian notes converting them to English notes on the fly. Nevertheless, you will be better off if you learn to use English notes. A trick I used was memorizing the notes as 'doC', 'reD', 'miE', 'faF', 'solG', 'laA', 'siB' and the reverse.

Try and write some ABC PLUS files as an exercise. Type random notes if you wish, but get used to writing, saving, converting, and viewing tunes. I suggest that you do your exercises at the end of each of the following sections.

♫ ♫ ♫ ♫ ♫ ♫ ♫

Part II
Melody

2 Notes

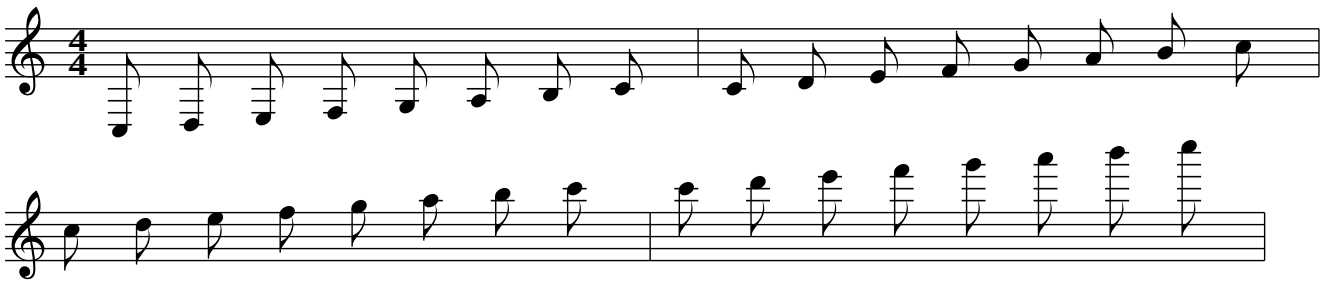
This part of the manual deals with basic characteristics of notes: pitch, length, accidentals, dots, ties, slurs, tuplets, chords, grace notes, and expression symbols.

In some of the following examples, formatting parameters were removed from the sources for clarity.

2.1 Pitch: `A-G a-g ,'`

The following source (`scale2.abc`) shows how to obtain notes under and above the staff; the scale is C major. Instead of just writing `K:C` as in the previous example, we'll add a small bit of code which will be further explained in Section 3.1. `K:C treble`, besides specifying C major, forces the treble clef. In this example it must be specified because there are several notes much below the staff, and `abcm2ps` would set the bass clef by default. Convert the source *without* `-c`:

```
X: 1
K: C treble
% C major, four octaves:
C, D, E, F, G, A, B, C | C D E F G A B c |
c d e f g a b c' | c' d' e' f' g' a' b' c'' |
```



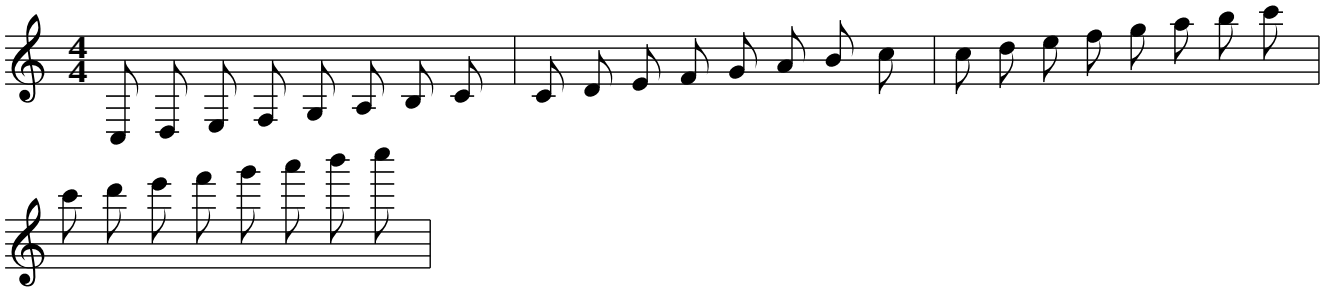
The rule is: if a note is followed by one or more commas, it goes down one or more octaves; if it is followed by one or more apostrophes, it goes up one or more octaves.

Note another important detail: we wrote two lines of ABC PLUS notes, which produced two lines of music. This is one of the basic rules of the ABC notation: *a new line in the source starts a new line in the score.* Exceptions will be examined in Section 2.15.

In the above example, note spacing is too wide; a single line would probably look better. We can instruct `abcm2ps` to ignore line breaks and try and format the measures optimally. This is done with `-c` in the command line.

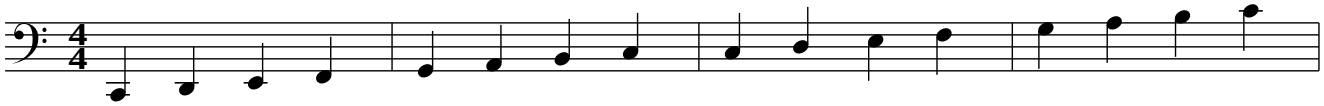
We'll now reformat the file using `-c`:

```
Convert with: abcm2ps -O= -c
```



The bass clef is automatically selected when we write low notes:

```
X: 1
L: 1/4
K: C
C,,D,,E,,F,,|G,,A,,B,,C,|C,D,E,F,|G,A,B,C|
```



2.2 Note Length: *L:*

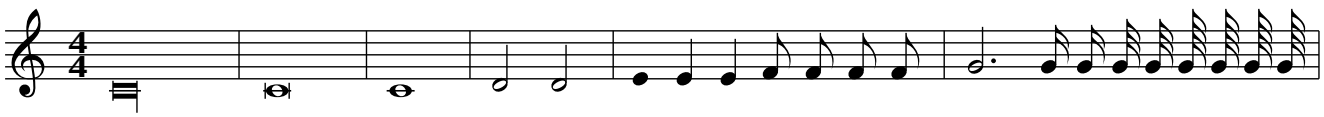
Unless otherwise indicated, the *note length* is automatically set in accordance to the tune metre.

The rule is: if the value of the metre is greater than or equal to 0.75, the default note length will be one eighth; if it is less than 0.75, one sixteenth. For example, when the metre is 4/4 the value is 1 (4 divided by 4 is 1), so the note length will be one eighth; if the metre is 3/4 = 0.75, again one eighth; if the metre is 2/4 = 0.5, the note length will be one sixteenth. By default, the metre is 4/4 and the note length is one eighth.

The `L:` field is used to modify the default note length, specifying a value as in `L:1/4`. (To change the metre, the `M:` field is used; see Section 3.2.)

To double, triple etc. the length of a note, you write the number 2, 3, etc. immediately after. To divide the value of a note by 2, 4 etc. , you write `/2`, `/4`, `/8`... or, equivalently, `/`, `//`, `///`... Spaces between the note and the digit or the slash are not allowed. Let us see an example:

```
X: 1
L: 1/4
K: C
C16|C8|C4|D2 D2|E0 E E F/ F/ F/ F/|
G3 G// G/4 G/8 G/8 G/16 G/16 G/16 G/16|
```



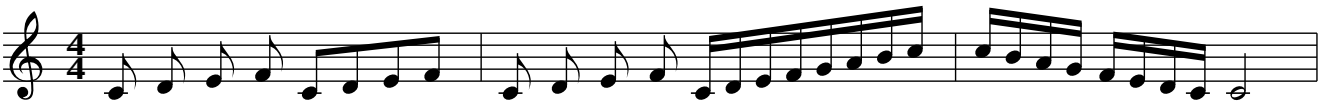
Note that `abcm2ps` supports notes that are longer than a whole note! The first note in the example is called a *longa*, and its length is four whole notes. The second note is a *brevis*, two whole notes. The

spacing between notes is proportional to their length. (We shall see in Section 5 how to make the spacing of notes constant instead of proportional.) Another weird note is the first `E` in measure 5: the trailing `0` denotes a stemless quarter note.[2]

1/128 notes and rests are also available.

Spaces between notes and measure bars can be freely inserted when the notes are longer than one eighth, and are used to improve the readability of the source. But *spaces between notes whose length is equal or lesser than one eighth are not optional*. If these notes are not separated by spaces, they will be grouped under a beam:

```
X: 1
K: C
C D E F CDEF | C D E F C/D/E/F/G/A/B/c/ |
c/B/A/G/ F/E/D/C/ C4 |
```

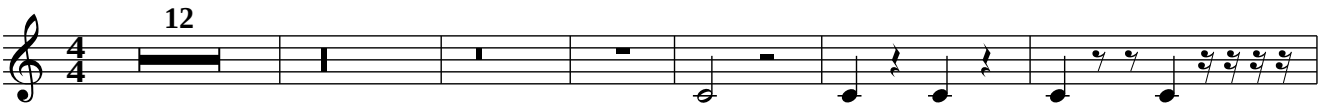


2.3 Rests and Spacing: `z Z x y`

Rests are indicated by the character `z` (lowercase zed). The same rules for note length apply to rests too, which can also be longer than a whole.

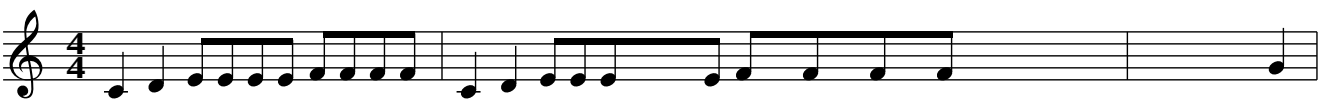
To notate multi-measure rests, you use `Z` (uppercase zed) followed by the number of measures you want to skip:

```
X: 1
L: 1/4
K: C
Z12|z16|z8|z4|C2 z2|C z C z|C z/ z/ C z// z// z// z//|
```



The characters `x` and `y` denote, respectively, *invisible rests* and additional spacing:

```
X: 1
L: 1/4
K: C
C D E/E/E/E/ F/F/F/F/|C D E/E/E/yyE/ F/yF/yF/yF/ yyyy|xxxG|
```



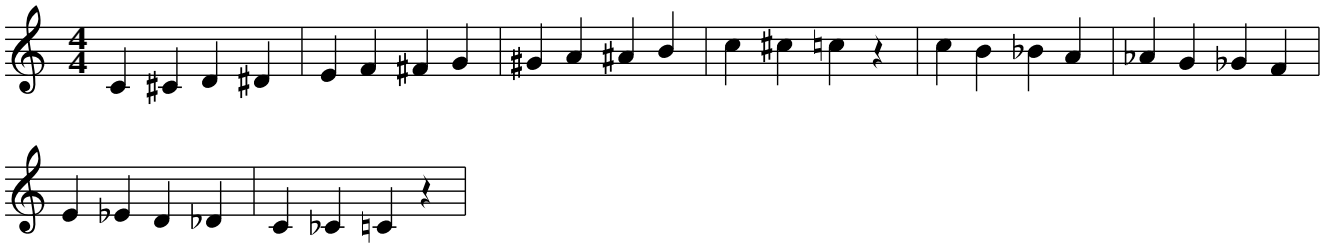
[2] these peculiar note lengths are sometimes found in early music.

`y` can be followed by a width in points; default is 20 points.
Invisible rests are often used for transcribing piano music; examples will be shown in Section 4.2.

2.4 Accidentals: ^_ =

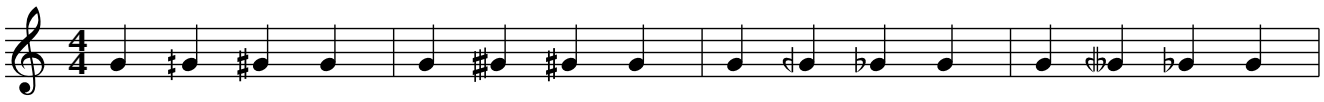
Sharp ♯ is denoted by a `^` before the note, flat ♭ by `_` (underscore), and natural ♮ by `=`. Spaces between the accidental and the note are not allowed. This is the chromatic scale:

```
X: 1
L: 1/4
K: C
C ^C D ^D | E F ^F G | ^G A ^A B | c^c=cz |
c B _B A | _A G _G F | E _E D _D | C_C=Cz |
```



Double accidentals are indicated doubling the special character: double sharp ♯♯ by `^^` and double flat ♭♭ by `__`.
Microtonal accidentals are available for 1/4 and 3/4 sharps and flats. 1/4 accidentals are obtained adding `/` after `^` or `_`, while 3/4 are obtained adding `3/2`:

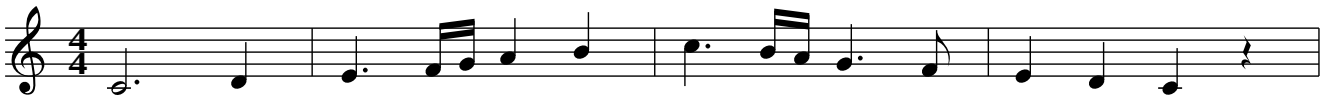
```
X: 1
L: 1/4
K: C
G ^/G ^G G | G ^3/2G ^G G | G _/G _G G | G _3/2G _G G |
```



2.5 Dotted Notes: < >

Notes followed by a dot are printed specifying the right length:

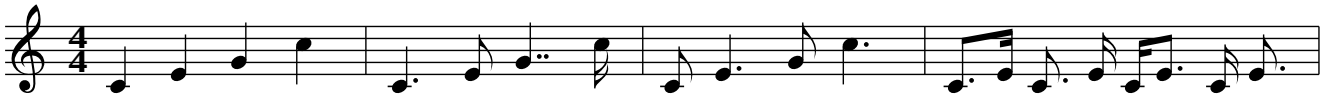
```
X: 1
L: 1/4
K: C
C3D|E3/2 F//G// A B| c3/2 B//A// G>F|E D C z|
```



When a note is dotted and the following is halved or vice versa, we are talking of *broken rhythm*. It is obtained using the characters `>` or `<` between two notes.

When you use `>`, the first note is dotted (that is, its duration increases by half) and the following note is halved. The opposite with `<`. To indicate a note followed by two or three dots, use `>>` or `>>>`.

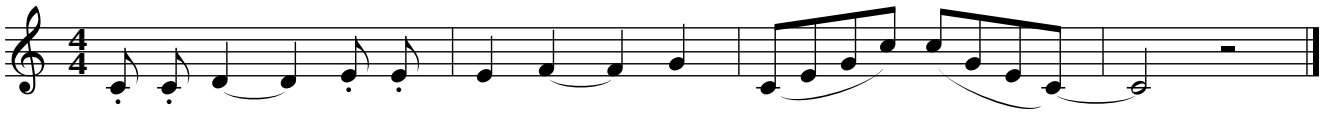
```
X: 1
L: 1/4
K: C
CEGc|C > E G >> c|C < E G < c|C/>E/ C/ > E/ C/<E/ C/ < E/|
```



2.6 Ties, Slurs, Staccato: - () .

A tie is obtained with the character `-` (hyphen) between two notes *of equal pitch*. Slurs are notated enclosing the notes in parentheses. Finally, the staccato mark is obtained by putting a dot before the note. Spaces between these symbols and the notes are not allowed.

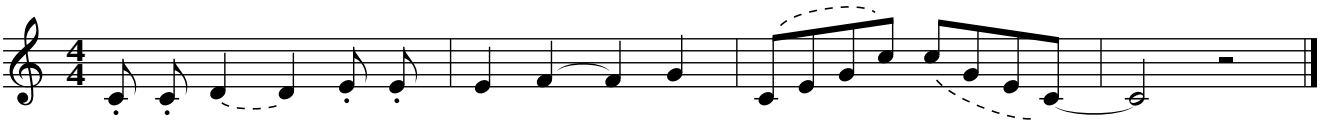
```
X: 1
L: 1/4
K: C
.C/ .C/ D - D .E/ .E/|EF-FG|(C/E/G/c/) (c/G/E/C/)|-C2 z2|]
```



Although ties and slurs are graphically very similar, they have a completely different musical meaning. Avoid the error of using ties to connect notes of different pitch: the MIDI output (see Section 11) would be wrong.

Dashed slurs and ties are indicated by a dot before the open parentheses or hyphen sign. Further, the slur and tie direction may be specified adding a `'` (above) or `,` (below):

```
X: 1
L: 1/4
K: C
.C/ .C/ D .- D .E/ .E/|EF-'FG|.('C/E/G/c/) .(c/G/E/C/)|-C2 z2|]
```



2.7 Tuplets: `(n`

Triplets, quadruplets, quintuplets etc. are coded with an open left parenthesis, immediately followed by the number of notes (2–9), then by the notes.

More precisely, the notation is:

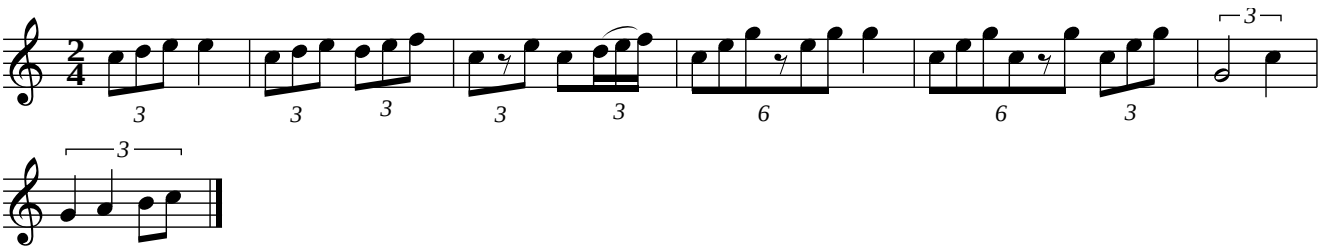
- `(2` : 2 notes in the time of 3
- `(3` : 3 notes in the time of 2
- `(4` : 4 notes in the time of 3
- `(5` : 5 notes in the time of ⟨*n*⟩ (see below)
- `(6` : 6 notes in the time of 2
- `(7` : 7 notes in the time of ⟨*n*⟩
- `(8` : 8 notes in the time of 3
- `(9` : 9 notes in the time of ⟨*n*⟩

The notes must have the same length. If the tune metre is compound, that is 6/8, 9/8, 12/8 etc. , ⟨*n*⟩ is 3; otherwise, it is 2.

More complex tuplets are coded using the extended notation `(`⟨*n*⟩`:`⟨*t*⟩`:`⟨*x*⟩, which means: put ⟨*n*⟩ notes into the time of ⟨*t*⟩ for the next ⟨*x*⟩ notes. The first parameter is the number printed over the tuplet. Extended tuplets are used to include notes of different lengths in a tuplet.

If the second parameter is omitted, the syntax is equivalent to that of simple tuplets. If the third parameter is omitted, it is assumed to be equal to the first.

```
X: 1
M: 2/4
L: 1/8
K: C
(3cde e2 | (3cde (3def | (3cze c(3(d/e/f/)|
(6cegzeg g2| (6cegczg (3ceg |
(3:2:2G4c2 | (3:2:4G2A2Bc |]
```

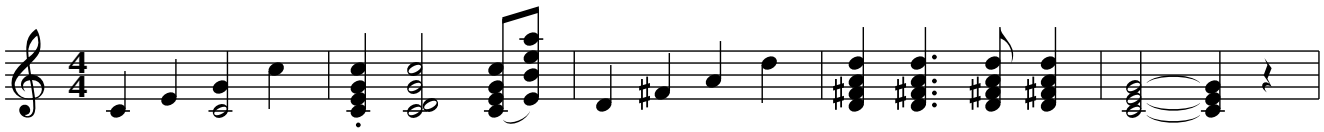


Nested tuplets are slightly more complex, and will be covered in Section 9.5.

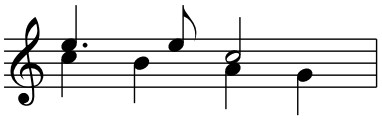
2.8 Chords: `[]`

Chords are written enclosing the notes in square brackets; spaces between the braces and the notes are not allowed. A chord behaves as a single note when you have to add dots, slurs, etc. That is, it can be preceded by a dot for staccato, or by a symbol, and so on. To tie two chords, each note is followed by `-`.
The duration of the chord notes can also be specified adding a number after the closing bracket.

```
X: 1
L: 1/4
K: C
CE [C2G] c| .[CEGc][C2D2G2c2] ([C/E/G/c/][E/a/B/e/])|
D^FAd|[D^FAd][D^FAd]>[D^FAd][D^FAd]|[C2-E2-G2-][CEG]z|
```



Do not mistake chords for something completely different! If you want to get something like this:



these are not chords, but different *voices* on the same staff. This will be the subject of Section 4.

Please note the chord in the first measure. Although `abcm2ps` will allow you to write chords containing notes of different length, doing so is *not recommended* because other ABC PLUS programs do not support this feature.

2.9 Lyrics: `W:` `w:`

Lyrics can be added at the end of the tune, or aligned with the notes under the staff. In the first case, at the end of the body you add lines that start with the `W:` (upper case) field, followed by the lyrics.

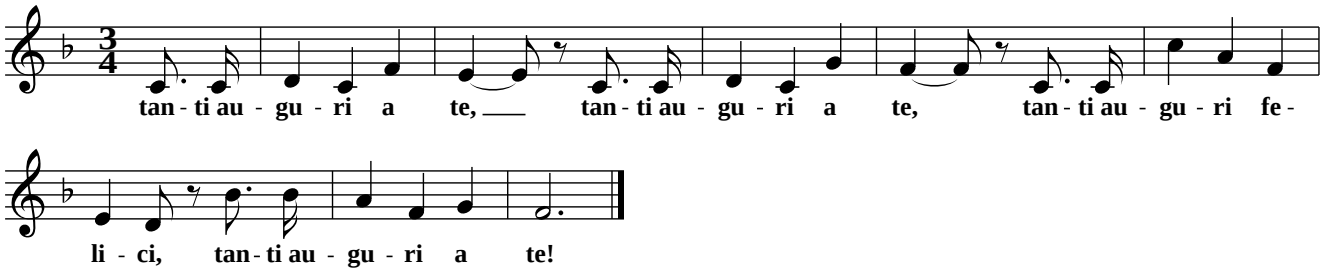
Note-aligned lyrics are obviously more complex to write. Immediately after a line of music, you write one or more lines that start with the `w:` (lower case) field, followed by the lyrics split in syllables. Alignment rules are:

- the `-` character (hyphen) separates syllables of a word. If it is separated from the previous syllable by a space, a note is skipped. $\langle n \rangle$ `-` characters separated from the previous syllable skip $\langle n \rangle$ notes. Spaces after the `-` have no effect, and can be used for better legibility.
- `|` skips to the next measure.
- `_` (underscore) the last syllable is sung for an extra note, and a horizontal line is drawn.
- `*` skips a note.
- `~` (tilde) joins two syllables under a note.
- `\-` inserts a `-` character.

- `\` continuation character; the next `w:` line continues the same line of lyrics.

The following tune is ‘Happy Birthday’ in Italian:

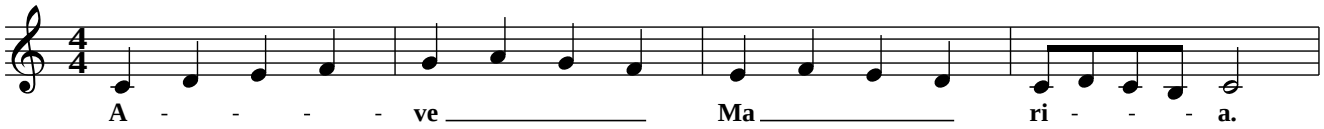
```
X: 1
M: 3/4
K: F
C> C | D2C2F2 | E2-E z C> C | D2C2G2 | F2-F z C> C |
w: tan- ti˜au- gu- ri a te,_ tan- ti˜au- gu- ri a te, * tan- ti˜au-
c2A2F2 | E2D z B> B | A2F2G2 | F6 |]
w: gu- ri fe- li- ci, tan- ti˜au- gu- ri a te!
%
W: Tanti auguri a te, tanti auguri a te,
W: tanti auguri felici, tanti auguri a te!
```



Tanti auguri a te, tanti auguri a te,
tanti auguri felici, tanti auguri a te!

Beware of the difference between `-` (hyphen) and `_` (underscore). Usually, `-` is used to skip syllables within a word, while `_` is used at the end of a word. For instance:

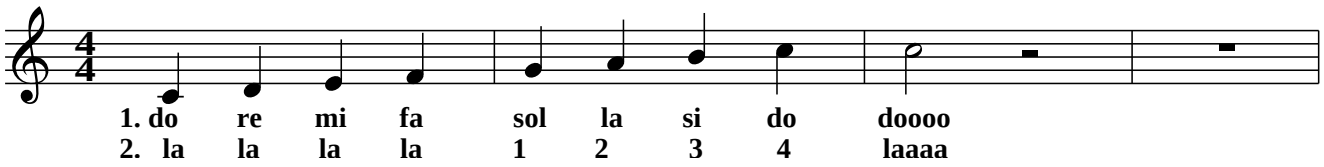
```
X: 1
L: 1/4
K: C
CDEF|GAGF|EFED|C/D/C/B,/ C2|
w: A ---ve___ Ma___ri ---a.
```



In the third measure, `-` (hyphen) should be used.

If a `w:` line contains digits, these will not be aligned with the notes but moved a bit to the left; this feature is used to enumerate subsequent `w:` lines. The digit must be followed by a `~` and joined with the following syllable. If you want to align digits with notes (i.e. for fingerings), all you have to do is insert a `~` character just before the number.

```
X: 1
L: 1/4
K: C
CDEF|GABc|c2z2|z4|
w: 1.˜do re mi fa sol la si do doooo
w: 2.˜˜˜la la la la 1 2 ˜3 ˜4 laaaa
```



Take special care to write a number of syllables that matches the number of notes! Mismatch between notes and syllables is one of the most common causes of error.

2.10 Foreign Characters

As long as you write lyrics in English, no problem. Things get tricky when you have to write lyrics in foreign languages like Italian, German, Hungarian... you will need accented characters that don't appear on standard US keyboards.

The problem is solved by using special character sequences: you start with `\`, then a special character, then the character to be altered. It is easier to do than to explain: please see Table 2. Note the difference between the 'acute' `'` and 'grave' `` ` `` accent.

A complete table of symbols is shown in Appendix D.

Letter	Sequence
à è á é	\`a \`e \'a \'e
Ü ü ë ö	\"U \"u \"e \"o
Ñ ñ	\~N \~n
ß ø Ø å Å	\ss \/o \/O \aa \AA
Ô ô ç Ç	\^O \^o \cc \cC
Æ Œ æ œ	\AE \OE \ae \oe

Table 2: How to obtain characters of foreign languages.

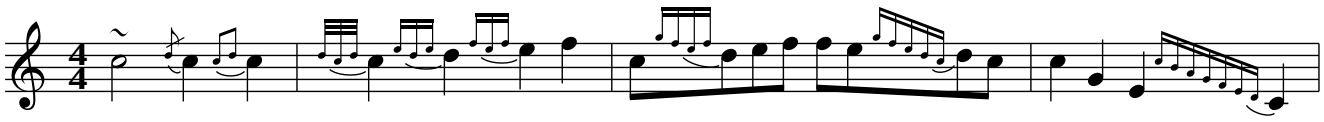
2.11 Grace Notes: ~ {}

The `~` (tilde) character denotes a *generic gracing*. Its meaning and method of execution depend on the player's interpretation. For instance, a fiddler may play a roll or a cut.

To notate grace notes, one or more notes are enclosed in curly braces before the main note. To add a slash to a grace note (*acciaccatura*), the open curly brace is followed by `/`, then by a single note. Grace notes can also be written without tying them to a note.

A grace note length can also be specified. (Music theorists need not apply....) The unit grace note length does not depend on `L:` or `M:`; it is always 1/8 for a single grace note, 1/16 for two or more, and 1/32 in bagpipe tunes.

```
X: 1
L: 1/4
K: C
~c2 {/d}c {c2d2}c|{d/c/d/}c {ede}d {fef}e f|
c/{gfef}d/e/f/ f/e/{gfedc}d/c/|c G E {cBAGFED}C|
```



To remove the slur joining grace notes to the main note, you use the `-G` option of `abcm2ps`, or a formatting parameter that we'll see later.

2.12 Expression Symbols: *!symbol!*

Expression symbols[3] are notated using the general form `!symbol!`: that is, the expression symbol name (***ff***, ***ppp***, *cresc.*...) enclosed in exclamation marks. Symbols are written immediately before the note, and can be applied even to grace notes.

`abcm2ps` also accepts `+symbol+` instead of `!symbol!`.

Some symbols are printed above the staff if there are `w:` lines, under the staff otherwise. We will see in Section 5 how to specify the positioning.

The expression symbols listed in Figure 3 are supported. You will probably notice that some symbols you need are missing. Please read Section 9 before you get upset...

```
X: 1
T: Symbols
L: 1/4
M: none
K: C
!+!c!0!c!1!c!2!c |!3!c!4!c!5!c!D.C.!c|!D.S.!c!accent!c
w: !+! !0! !1! !2! !3! !4! !5! !D.C.! !D.S.! !accent!
c///c///!beambr1!c///c/// c///c///!beambr2!c///c/// !breath!c!coda!c|
w: ** !beambr1! * ** !beambr2! * !breath! !coda!
!crescendo(!c!crescendo)!c!diminuendo(!c!diminuendo)!c|
w: !crescendo(! !crescendo)! !diminuendo(! !diminuendo)!
% these are equivalent
!<(!c!<)!c!>(!c!>)!c|
w: !<(! !<)! !>(! !>)!
!downbow!c!emphasis!c!>!c!fermata!c!f!c|!ff!c!fff!c!ffff!c!fine!c|
w: !downbow! !emphasis! !>! !fermata! !f! !ff! !fff! !ffff! !fine!
!invertedfermata!c!longphrase!c!lowermordent!c
w: !invertedfermata! !longphrase! !lowermordent!
!mediumphrase!c| !mf!c!mordent!c!mp!c!open!c!p!c|!pp!c
w: !mediumphrase! !mf! !mordent! !mp! !open! !p! !pp!
!ppp!c!pppp!c!plus!c!pralltriller!c|
w: !ppp! !pppp! !plus! !pralltriller!
!roll!c!segno!c!sfz!c!shortphrase!c|!snap!c!tenuto!c!thumb!c
w: !roll! !segno! !sfz! !shortphrase! !snap! !tenuto! !thumb!
```

[3] also referred to as 'decorations'.

```
C/!trem1!c/ C!trem2!c C2!trem3!c2 C!trem4!c !trill!c|
w: * !trem1! * !trem2! * !trem3! * !trem4! !trill!
!turn!c!upbow!c!uppermordent!c!wedge!c|
w: !turn! !upbow! !uppermordent! !wedge!
!turnx!c!invertedturn!c!invertedturnx!c
w: !turnx! !invertedturn! !invertedturnx!
!arpeggio![CEGc]!trill(!c| !trill)!c2|]
w: !arpeggio! !trill(! !trill)!
```

Symbols

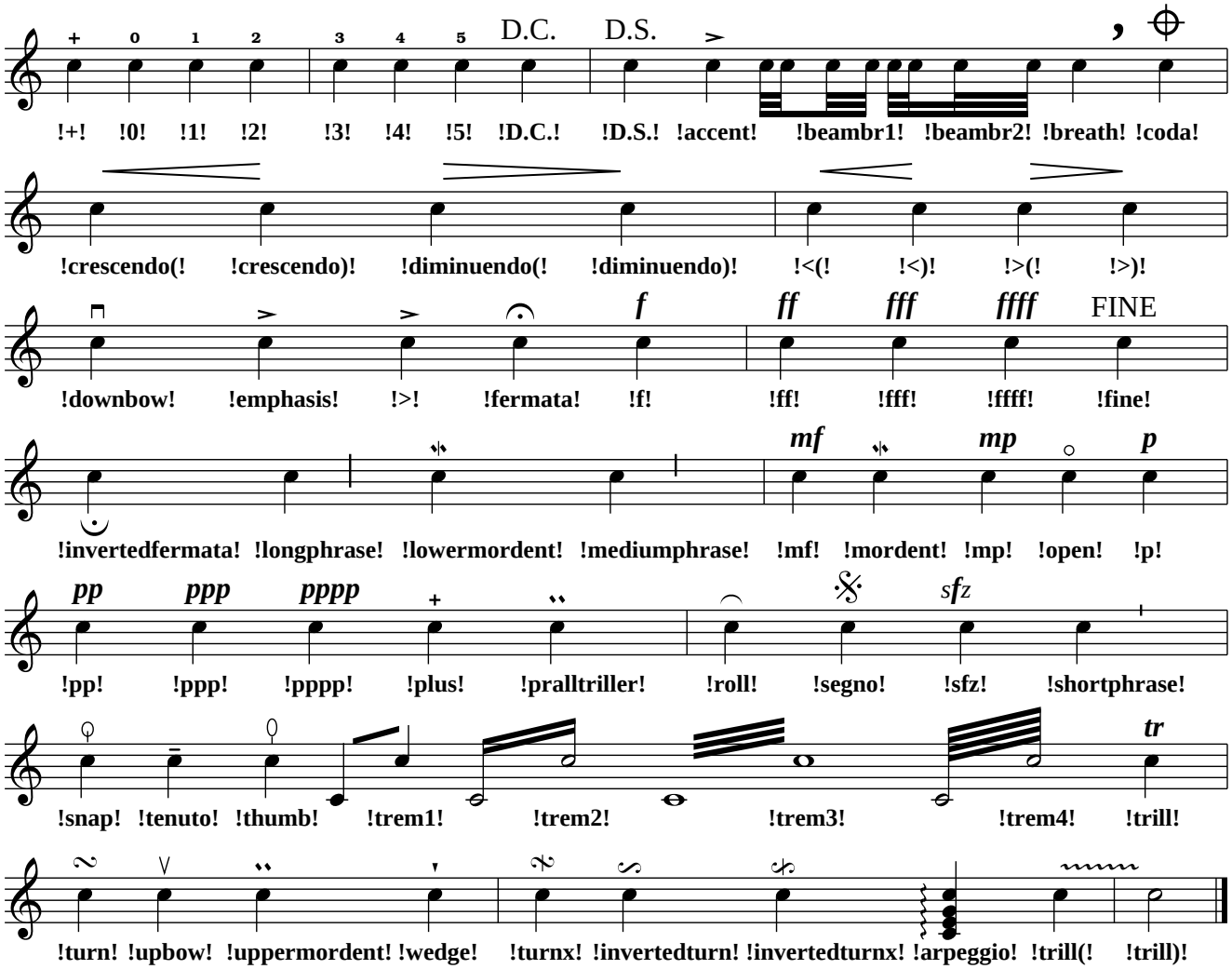


Figure 3: Standard expression symbols.

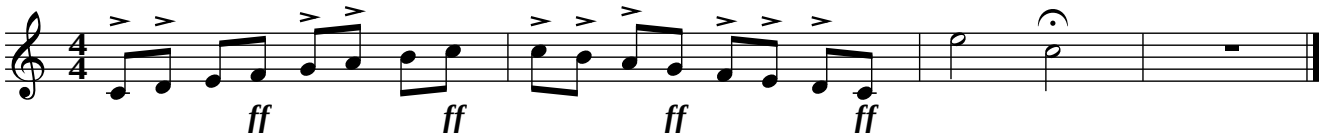
If the tune contains many symbols, an alternative notation may be handy. After a line of music, you write a line that starts with the `d:` field (or, equivalently, `s:`). This line will contain only symbols.

Rules for matching notes and symbols are those explained in Section 2.9. `d:` lines and note-linked symbols can be used at the same time.

Abbreviation	Symbol
u	!upbow!
v	!downbow!
T	!trill!
H	!fermata!
L	!accent! or !emphasis!
M	!lowermordent!
P	!uppermordent!
S	!segno!
O	!coda!

Table 3: Standard abbreviations for common symbols.

```
X: 1
L: 1/4
U: M = !accent!
K: C
C/D/ E/F/ G/A/ B/c/|c/B/ A/G/ F/E/ D/C/|e2!fermata!c2|z4|]
d: M M * !ff! M M * !ff! M M M !ff! M M M !ff! |
```



2.13 Redefinable Symbols: `U:`

Most symbol names are quite verbose, and may make the source difficult to read. To solve this snag, you can assign a single letter to a symbol using the `U:` field.

The field is followed by an upper-case letter from `H` to `Y` or by a lower-case letter from `h` to `w`, then by `=`, then by the symbol. For example, the following `U:` fields define `T` as equivalent to `!trill!`, `H` to `!fermata!`, and `M` to `!tenuto!`:

```
U: T = !trill!
U: H = !fermata!
U: M = !tenuto!
```

To reset the definition of a `U:` field, you use a definition like:

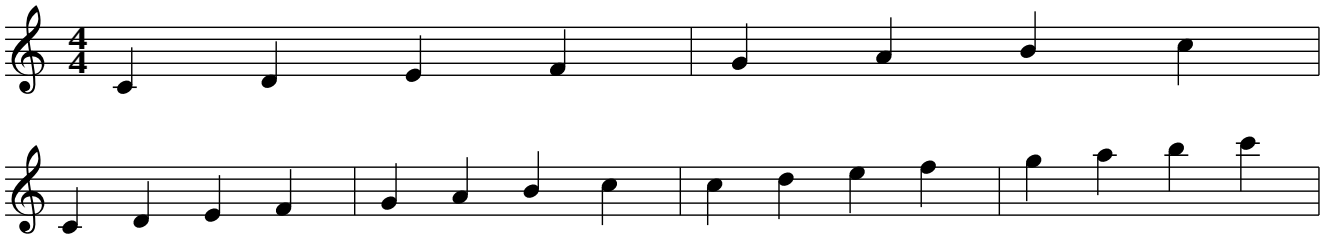
```
U: T = !nil!
U: H = !nil!
U: M = !nil!
```

The letters `uvTHLMPSO` are predefined abbreviations for common symbols; definitions are shown in Table 3.

2.14 Forcing Line Breaks: *!*

If you don't use the `-c` option of `abcm2ps`, you can force staff breaks with a single `!` character. In the next example, we will get two lines of music: the first with two measures, the second with four:

```
X: 1
L: 1/4
K: C
CDEF|GABc|! CDEF|GABc|cdef|gabc'|
```



Needless to say, the above score looks awful… forcing line breaks should be done with care to avoid ugly results.

2.15 Avoiding Line Breaks: \

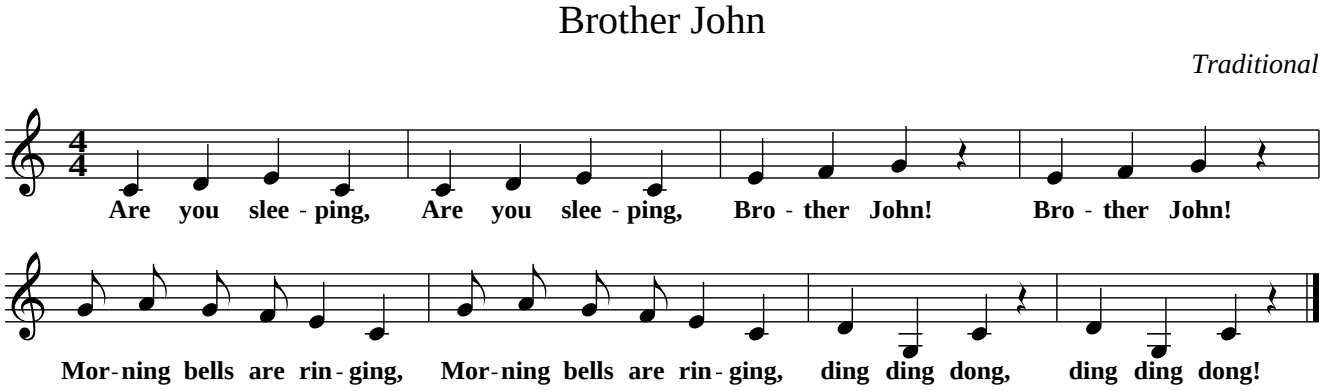
Usually, ⟨*n*⟩ measures in a source line produce ⟨*n*⟩ measures in the score. Sometimes is not convenient to write, say, six measures on the same line, because the source becomes less readable.

In such cases, the `\` character can be added to the end of a line to indicate that the staff is to be continued. Similarly, a lyrics line (`w:`) may be broken into several lines in the same manner.

The first part of a music line that ends in `\` must be followed by the corresponding `w:` line, if any.

The following example yields two staves, four measures each:

```
X: 1
T: Brother John
C: Traditional
L: 1/4
K: C
CDEC|CDEC|EFGz|\ % continues
w: Are you slee-ping, Are you slee-ping, Bro-ther John!\ % continues
EFGz|
w: Bro-ther John!
G/ A/ G/ F/ EC|G/ A/ G/ F/ EC|\ % continues
w: Mor-ning bells are rin-ging, Mor-ning bells are rin-ging,\ % continues
DG,Cz|DG,Cz|]
w: ding ding dong, ding ding dong!
```

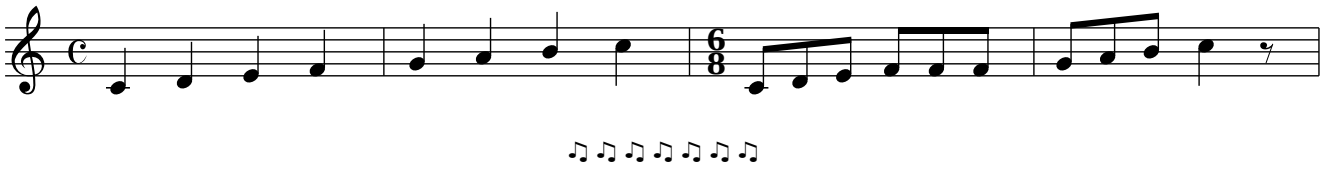


2.16 Inline Fields

When one wants to change metre or other music properties, a new field is entered on a line on its own. However, there's another method that avoids splitting the music into lines: *inline fields*.

Inline fields are inserted enclosing a field in square brackets, with no leading and trailing spaces. Inline fields are used in this example to change the note length and the meter:

```
X: 1
L: 1/4
M: C
K: C
CDEF|GABc| [M:6/8][L:1/8] CDE FFF|GAB c2 z|
```



♫ ♫ ♫ ♫ ♫ ♫ ♫

3 Music Properties

3.1 Key signatures and Clefs: `K:`

So far, we have written our examples in treble clef and C major scale. The `K:` field may be used to alter both the key signature and the clef.

3.1.1 Key Signatures

`K:` must be followed by a note name in upper case, followed by `m` or `min` when the mode is minor. Accidentals are written as `#` for ♯ and `b` for ♭.

I remind you the simple rule to find out the major key according to the number of sharps or flats: *one tone* higher than the last sharp note, or *one fourth* below the last flat note. For your convenience, Table 4 shows the keys that correspond to a specified number of sharps or flats.

Peculiar key signatures are `K:HP` and `K:Hp`, which are used for highland bagpipe music. The latter marks the staff with F♯, C♯ and G♮; both force note beams to go downwards.

Western classical music only uses major and minor modes, but others exist that are still used in other musical traditions. A case in point is Irish traditional music, which widely employs *modal scales*. I am

Keys with sharps	Keys with flats
none: C (Am)	
1 sharp: G (Em)	1 flat: F (Dm)
2 sharps: D (Bm)	2 flats: B♭ (Gm)
3 sharps: A (F♯m)	3 flats: E♭ (Cm)
4 sharps: E (C♯m)	4 flats: A♭ (Fm)
5 sharps: B (G♯m)	5 flats: D♭ (B♭m)
6 sharps: F♯ (D♯m)	6 flats: G♭ (E♭m)
7 sharps: C♯ (A♯m)	7 flats: C♭ (A♭m)

Table 4: Correspondence between the key and the number of sharps or flats.

not going to explain what they are; suffice it to say that you may come across strange key signatures such as `AMix` or `EDor`, that is 'A Mixolydian' and 'E Dorian'. Table 5 lists them all.

Sharps / Flats	Major Ionian	Minor Aeolian	Mixolydian	Dorian	Phrygian	Lydian	Locrian
7 sharps	C#	A#m	G#Mix	D#Dor	E#Phr	F#Lyd	B#Loc
6 sharps	F#	D#m	C#Mix	G#Dor	A#Phr	BLyd	E#Loc
5 sharps	B	G#m	F#Mix	C#Dor	D#Phr	ELyd	A#Loc
4 sharps	E	C#m	BMix	F#Dor	G#Phr	ALyd	D#Loc
3 sharps	A	F#m	EMix	BDor	C#Phr	DLyd	G#Loc
2 sharps	D	Bm	AMix	EDor	F#Phr	GLyd	C#Loc
1 sharp	G	Em	DMix	ADor	BPhr	CLyd	F#Loc
0 sharps	C	Am	GMix	DDor	EPhr	FLyd	BLoc
1 flat	F	Dm	CMix	GDor	APhr	BbLyd	ELoc
2 flats	Bb	Gm	FMix	CDor	DPhr	EbLyd	ALoc
3 flats	Eb	Cm	BbMix	FDor	GPhr	AbLyd	DLoc
4 flats	Ab	Fm	EbMix	BbDor	CPhr	DbLyd	GLoc
5 flats	Db	Bbm	AbMix	EbDor	FPhr	GbLyd	CLoc
6 flats	Gb	Ebm	DbMix	AbDor	BbPhr	CbLyd	FLoc
7 flats	Cb	Abm	GbMix	DbDor	EbPhr	FbLyd	BbLoc

Table 5: Modal scales.

Accidentals can also be specified explicitly by appending them to the key signature. For example, `K:D =c ^g` would set the key of D major but mark every `C` as natural, and every `G` as sharp. Lower case letters must be used, separated by spaces. When present, explicit accidentals always override the accidentals in the key signature.

The keyword `none`, meaning 'no accidentals', can also be used; e.g. `K: G none`. I can't imagine how it could be useful, though.

3.1.2 Clefs

By default, the clef is automatically selected by `abcm2ps` according to the pitch of the notes it encounters. For example, if you start a tune with notes much below the staff (notes with commas), `abcm2ps` will select the bass clef. However, you can set the clef with the `K:` field at the start of the tune; you can also choose not to have any clef at all.

The general syntax of the `K:` field is:

`K:` *[key] [clef=] ⟨clef type⟩ [line number] [+8] [-8] [middle=⟨pitch⟩] [transpose=] [stafflines=⟨number⟩]*

Clef	Field
Treble	`K: treble` (default)
Treble, 1 octave below	`K: treble-8`
Treble, 1 octave above	`K: treble+8`
Bass	`K: bass`
Baritone	`K: bass3`
Tenor	`K: alto4`
Alto	`K: alto`
Mezzosoprano	`K: alto2`
Soprano	`K: alto1`
G on third line	`K: middle=G`
no clef	`K: none`
percussions	`K: perc`

Table 6: Clefs and associated `K:` fields.

[staffscale=⟨number⟩]

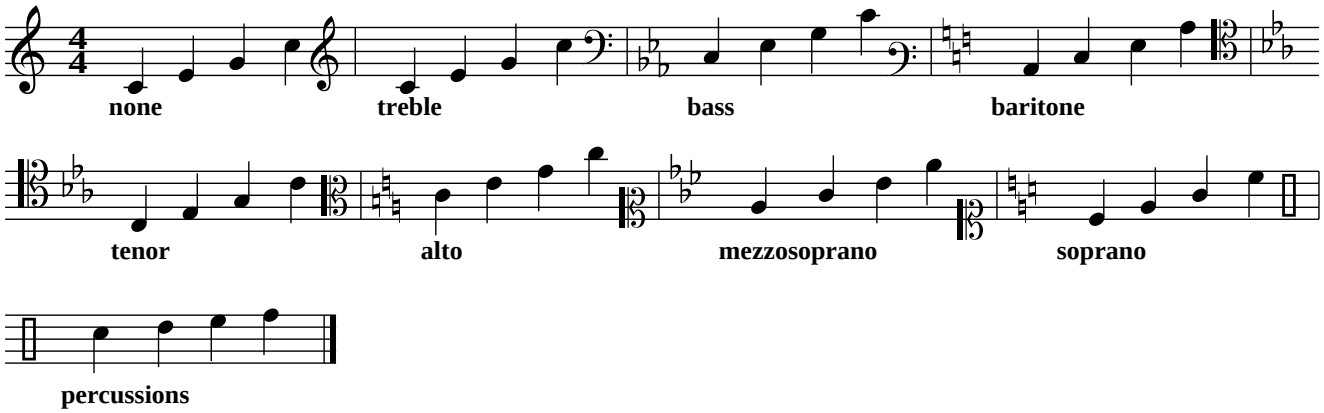
where:

- `clef=` may be omitted before the clef type;
- clef types can be:
  - a note pitch: only `G` (treble clef), `C` (alto clef), and `F` (bass clef) are allowed. These are the notes where the clef sits.
  - a clef name: `treble`, `alto`, `tenor`, `bass`.
  - the word `none` indicates that there is no clef.
  - the word `perc` indicates a clef for percussion instruments.
- ⟨*line number*⟩ indicates the staff line the clef sits on.
- `+8` and `-8` draws '8' above or below the staff.
- `middle=`⟨*pitch*⟩ specifies the note on the third (middle) line of the staff.
- `m=` same as above
- `transpose=` currently does nothing.
- `t=` same as above
- `stafflines=`⟨*n*⟩ sets the number of lines of the associated staff.
- `staffscale=`⟨*n*⟩ sets the scale of the associated staff. The default value is '1'.

To sum up, available clefs and the corresponding fields are listed in Table 6 and in the following example:

```
X: 1
L: 1/4
K: none
CEGc | [K: C treble] CEGc |[K: Cm bass]cegc' |
```

```
w: none | treble | bass |
[K: C bass3]cegc' | [K: Cm alto4]CEGc| [K: C alto]cegc' |
w: baritone | tenor | alto |
[K: Cm alto2]cegc' | [K: C alto1]cegc' | [K: perc] cdef |]
w: mezzosoprano | soprano | percussions |
```



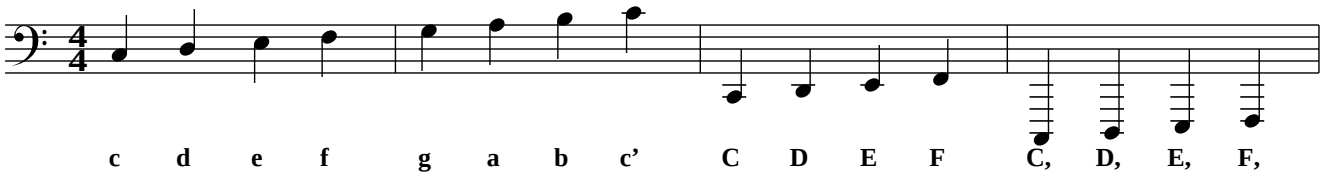
When working with clefs different than treble, `abcm2ps` may automatically transpose the music one or two octaves to fit the clef better. For example, in bass clef the two notes `c` and `C,` may be equivalent, depending on the context. This avoids having to type lots of commas.

I know it sounds confusing. To make the point clear, let's have a careful look at the following example:

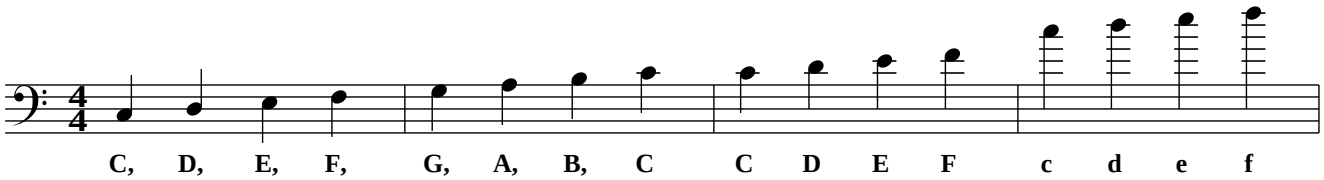
```
X: 1
T: Notes relative to "c"
L: 1/4
K: C bass
cdef|gabc'|CDEF|C,D,E,F,|
w: c d e f g a b c' C D E F C, D, E, F,

X: 2
T: Notes relative to "C,"
L: 1/4
K: C bass
C,D,E,F,|G,A,B,C|CDEF|cdef|
w: C, D, E, F, G, A, B, C C D E F c d e f
```

Notes relative to "c"



Notes relative to "C,"



In theory, it would be preferable to write notes in bass clef using commas. In practice, I admit to breaking this rule systematically…

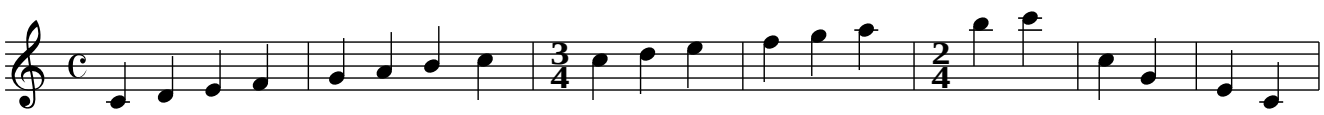
3.2 Metre: *M:*

The `M:` field specifies the tune metre in different ways:

- as a fraction, i.e. `M:4/4` or `M:3/4`. Complex indications can be used, such as `M:5/4 (2/4 3/4)`
- as an integer value: `M:2`
- as a textual indication: `M:C` or `M:C|` denote the metre of 4/4 and cut time (‘alla breve’)
- as an explicit measure duration: `M:C|=2/1`
- if there is no metre, use `M:none`.

Needless to say, the metre can change midtune. In this case, you insert an inline `M:` field in the body:

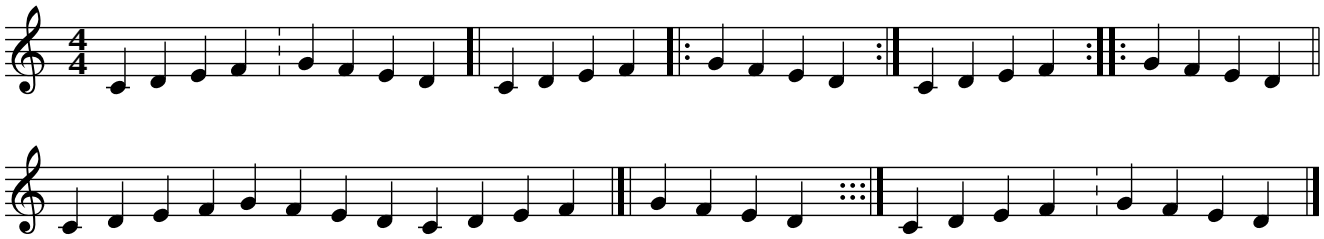
```
X: 1
M: C
K: C
L: 1/4
C D E F |G A B c| [M: 3/4] c d e|f g a| [M: 2/4] b c'|cG|EC|
```



3.3 Bars and Repeats: `|` *`/ : [ ]`*

In addition to the basic measure bar, others types of bars can be obtained using combinations of the `|`, `.`, `[`, `]`, and `:` characters.

```
X: 1
L: 1/4
K: C
CDEF .| GFED [| CDEF |: GFED :|CDEF :: GFED || CDEF [|] GFED
CDEF |[| GFED :::|] CDEF : GFED |]
```



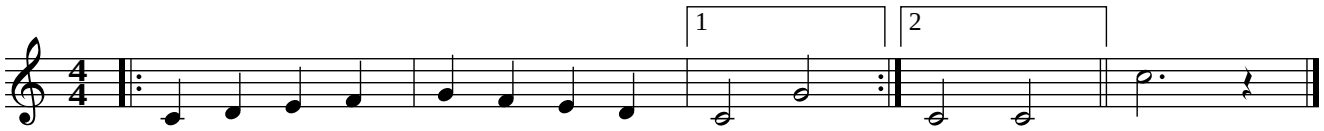
Previous versions of `abcm2ps` used a sequence for repeated bars: `|/|` and `|//|`. This syntax cannot be used anymore, but the same effect can be obtained using customised decorations. Further details in Section 9.

Note that `[|]` prints nothing; it is an *invisible bar*, and it can be used as a placeholder for a decoration. The same can be accomplished using `[]`. Also, note that `:` is the same as `.|`.

Just because you can write something like `::[||||:|[]`, this does not mean that it makes any sense!

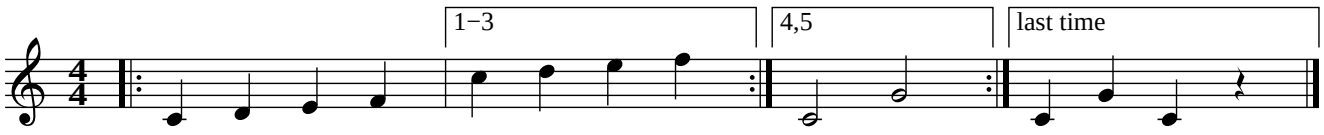
To indicate that a section has two different repeats, use the symbols `[1` and `[2` as in the following example. When the repeats symbols are close to a bar, they can be shortened using `|1` and `|2`. The end of the second repetition is set with `||`.

```
X: 1
L: 1/4
K: C
|: C D E F | G F E D |[1 C2 G2 :|2 C2 C2|| c3 z |]
```



`abcm2ps` also supports other types of repeats. Not only digits, but also dots, commas, hyphen signs and text in double quotes can be used:

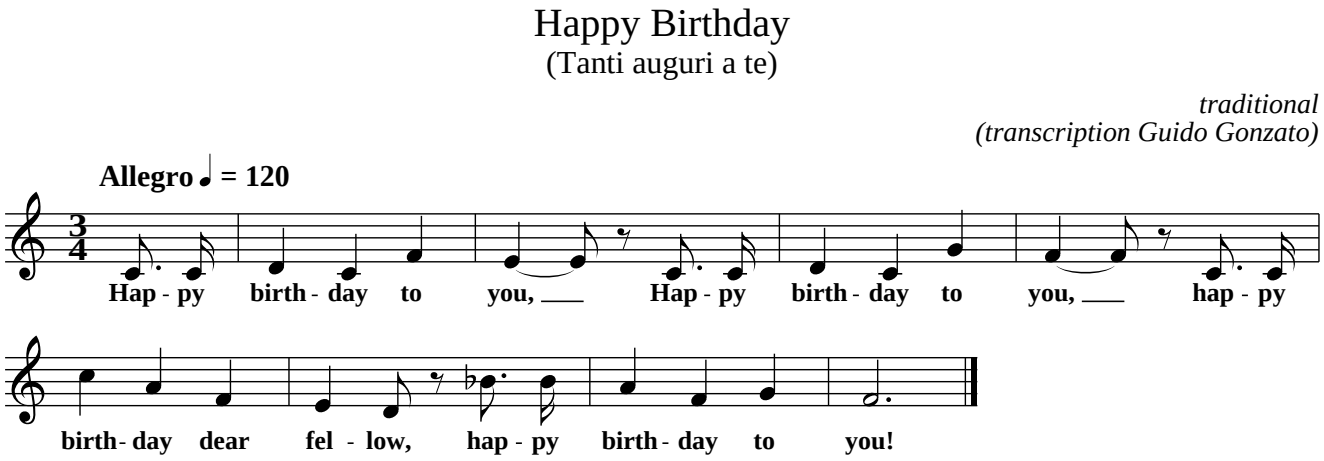
```
X: 1
L: 1/4
K: C
|: C D E F |1-3 c d e f :|4,5 C2 G2 :|["last time" C G C z |]
```



3.4 Title, Composer, Tempo: `T:` `C:` `Q:`

Our scores still miss something... In the next example we introduce the `T:` (title, subtitle), `C:` (composer) and `Q:` (tempo) fields:

```
X: 1
T: Happy Birthday     % title
T: (Tanti auguri a te)  % subtitle
C: traditional        % composer
C: (transcription Guido Gonzato)
M: 3/4
Q: "Allegro" 1/4 = 120 % tempo
K: C
C> C | D2C2F2 | E2-E z C> C | D2C2G2 | F2-F z C> C |
w: Hap-py birth-day to you,_ Hap-py birth-day to you,_ hap-py
c2A2F2 | E2D z _B> B | A2F2G2 | F6 |]
w: birth-day dear fel-low, hap-py birth-day to you!
```



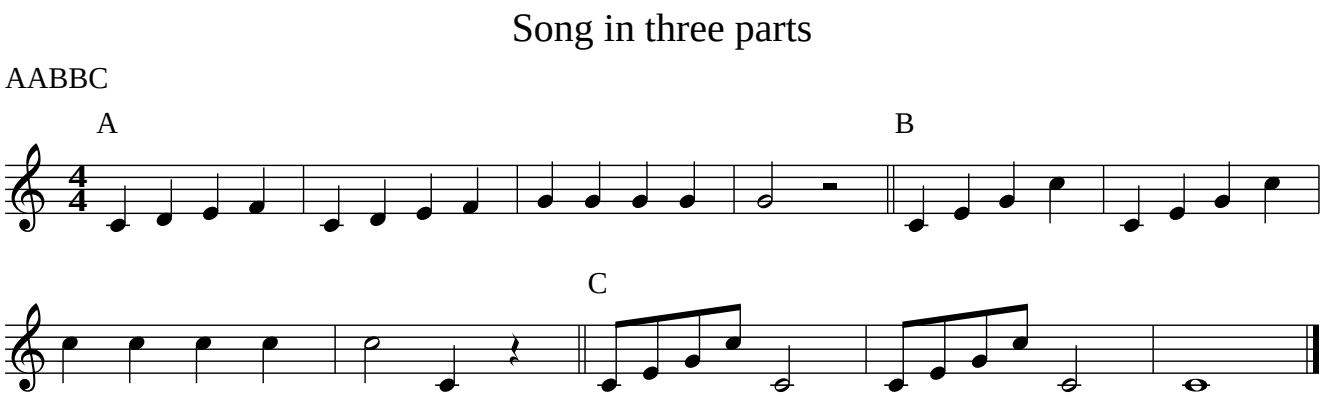
The text indication in the `Q:` field ('Allegro' in our example) can be omitted. In Section 5 we will learn how to change the title fonts.

3.5 Parts: `P:`

Some tunes are made of different parts, possibly repeated in several ways. To specify the order in which parts are played, the `P:` field is used, followed by the part names. In the header, this field specifies the order in which parts should be played; in the body, it marks the beginning of each part.

```
X: 1
T: Song in three parts
L: 1/4
```

```
P: AABBC % or: P: A2.B2.C
K: C
[P: A] C D E F|C D E F|G G G G|G2 z2||
[P: B] C E G c|C E G c|c c c c|c2 Cz||
[P: C] C/E/G/c/ C2|C/E/G/c/ C2|C4|]
```



Note that when the `P:` field is used in the header, the part name may be followed by a number indicating the number of repeats. Thus, `P:A3` is the same as `P:AAA`; `P:(AB)3C2` is equivalent to `P:ABABABCC`. To make the text more readable, dots may be used to separate the parts.

There you are a more complex example: `P:((AB)3.(CD)3)2` is equivalent to `P:ABABABCDCDCDABAB-ABCDCDCD` (count carefully!).

3.6 Accompaniment Chords: `""`

In many songbooks, accompaniment chords (say, for the guitar) are notated as 'A', 'C7', 'Dm', 'F#' etc. above the staff. In ABC PLUS, such chords are notated writing the chord name between double quotes `"` immediately before the note.

An accompaniment chord has this format:

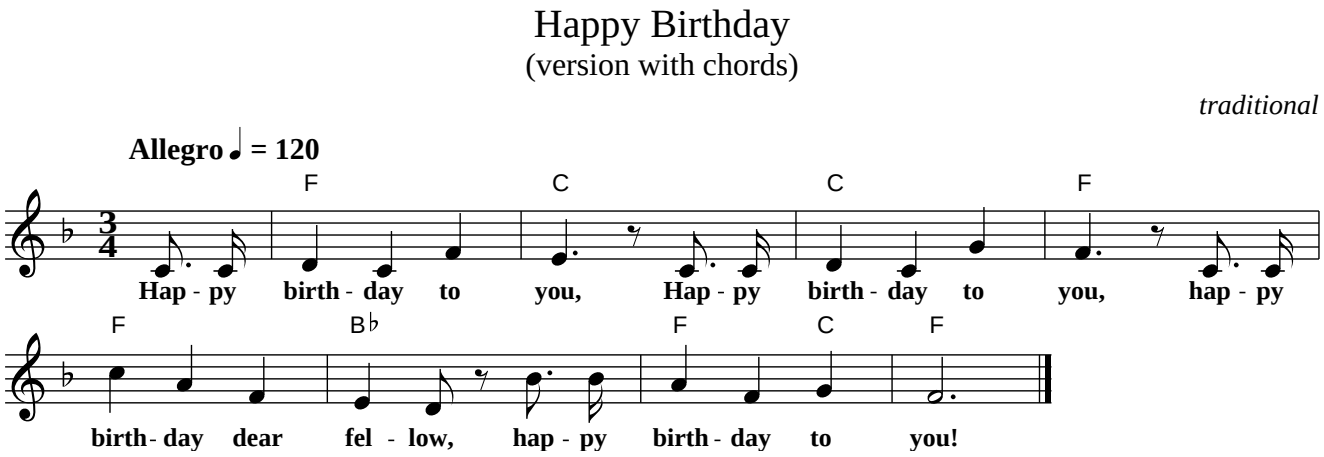
⟨*note*⟩ *[accidental] [type] [/bass note]*

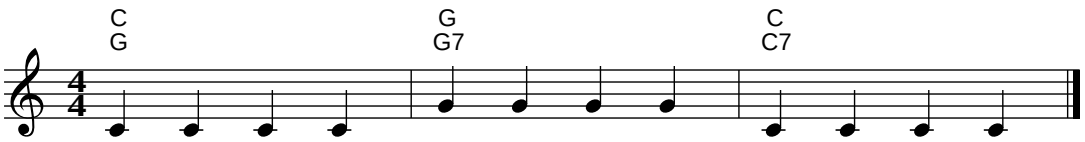
The note is A…G (upper case only); the accidental is indicated with `#` for ♯, `b` for ♭, or `=` for natural; the chord type is one of those listed in Table 7; finally, a slash `/` followed by a note A…G denotes an optional bass note. Spaces between the chord and the following note are not allowed.

```
X: 1
T: Happy Birthday
T: (version with chords)
C: traditional
M: 3/4
Q: "Allegro" 1/4 = 120 % tempo
K: F
C> C|"F"D2C2F2|"C"E3 z C> C|"C"D2C2G2|
w: Hap-py birth-day to you, Hap-py birth-day to
"F"F3 z C> C|"F"c2A2F2|"Bb"E2D z B> B|
w: you, hap-py birth-day dear fel-low, hap-py
"F"A2F2"C"G2|"F"F6|]
w: birth-day to you!
```

Type	Meaning
`m` or `min`	minor
`maj`	major
`dim`	diminished
`+` or `aug`	augmented
`sus`	sustained
`7`, `9`, ...	seventh, ninth, ecc.

Table 7: Types of accompaniment chords.





If you need to write accompaniment chords using Italian notes, i.e. `"Sol7"` instead of `"G7"`, *don't write them this way*. ABC requires that only notes in English notation be used; programs for translating sources into MIDI files conform to this standard. However, `abcm2ps` has a trick for printing accompaniment chords as Italian notes: please refer to Section 9.2.

Multiple chords per note are possible. They can be notated writing two or more consecutive chords before the same note, or using the separating characters `;` or `\n`:

```
X: 1
M: 4/4
L: 1/4
K: C
%
"C""G"CCCC|"G;G7"GGGG|"C\nC7"CCCC|]
```

3.7 Text Annotations: `"^_<>@"`

Text annotations can be added in different ways. The first method is to write the annotation as an accompaniment chord; that is, enclosing it between double quotes, but preceding the text by a special character. Another method is to use the `P:` (part) field. Finally, `Q:` fields can be inserted to specify tempo changes.

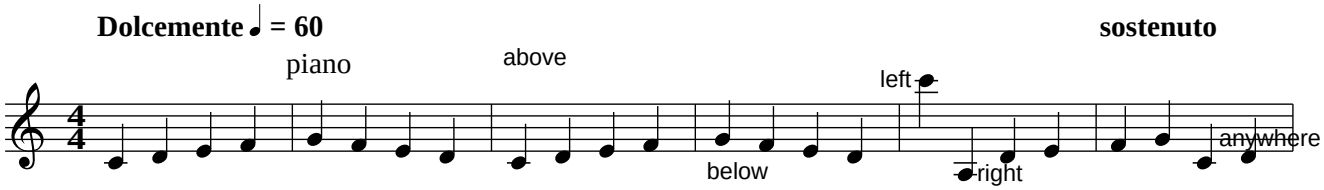
Text annotations should begin with one of these special characters: `^_<>@`. These characters set the logical difference between an annotation and an accompaniment chord, and specify the position of the annotation:

- `^` above the staff;
- `_` below the staff;
- `<` to the left of the note;
- `>` to the right of the note;
- `@` must be followed by two numbers `X` and `Y`, separated by a comma and optionally followed by a space. The annotation will be printed from the centre of the note head (the lowest note, if in a chord), with an offset of `X` horizontal and `Y` vertical points.

To include accidentals in text annotations, use `\#`, `\b`, and `\=`; note the leading `\`. Multiple annotations are notated like multiple chords, as seen in Section 3.6.

Let us see an example that uses all methods:

```
X: 1
Q: "Dolcemente" 1/4=60
L: 1/4
K: C
CDEF|[P:piano]GFED|"^above"CDEF|"_below"GFED|"<left"c'">right"A,DE|
[Q: "sostenuto"] FGC"@-15,5.7 anywhere"D|
```

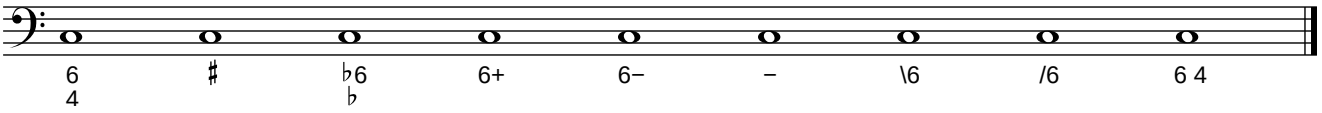


3.7.1 Figured Bass

Figured bass notation can be written using `d:` lines, containing text annotations instead of symbols:

```
X: 1
T: Figured bass
M: none
L: 1/4
K: C bass
%
c4 c4 c4 c4 c4 c4 c4 c4 c4|]
d: "_6;4" "_\#" "_\b6;\b" "_6+" "_6-" "_-" "_\\6" "_/6" "_6 4"
```

Figured bass



Please note that all annotations start with the _ character to print them below the staff.

3.8 Information Fields

In Section 1.7 I explained that ABC files may contain several tunes. This feature, together with the ease of use of ABC, spurred the creation of many ABC music archives on the Internet. As stated before, ABC has become *the* standard to spread folk and traditional music.

There are fields for describing tune properties such as the source area, rhythm, annotations, and more. These information fields can be used when browsing a database for a special kind of music. If you contribute ABC files to public sites such as http://www.thesession.org/, it might be a good idea to include at least the `O:`, `R:`, and `D:` fields.

A: area. Used to specify an area within the country where the tunes originates. Example: `A:Dublin`

B: book. Example: `B:Francis O'Neill: "The Dance Music of Ireland" (1907) no. 662`

D: discography. Example: `D:"The Chieftains 4" by The Chieftains`

F: file name. Example: `F:DrowsyMaggie.abc`

G: group. Usually used to specify the instrument on which the tune is played. Example: `G:whistle, flute`

H: history. Example: `H:this tune was collected by...`

I: information. Example: `I:version without ornamentation`. The `I:` field can also replace command lines, see Section 5.1.

N: notes. Example: `N:sometimes spelt "Drowsey Maggie"`

O: origin. Used to specify the country of origin of the tune. Example: `O:Ireland`

R: rhythm. Example: `R:Reel`

S: source. Used to specify where the ABC PLUS tune was found. Example: `S:from John Chambers' site`

Z: transcription notes. Example: `Z:Transcribed in C, originally in D`

♫ ♫ ♫ ♫ ♫ ♫ ♫

Part III
Harmony

4 Polyphony in ABC PLUS

So far, we only have seen melodies: music written for a single voice or instrument. This is all what ABC was able to do—and it is a lot: folk musicians do not need anything more.

Let us now turn our attention to ABC PLUS and its extensions for polyphonic music, using choral pieces for our examples.

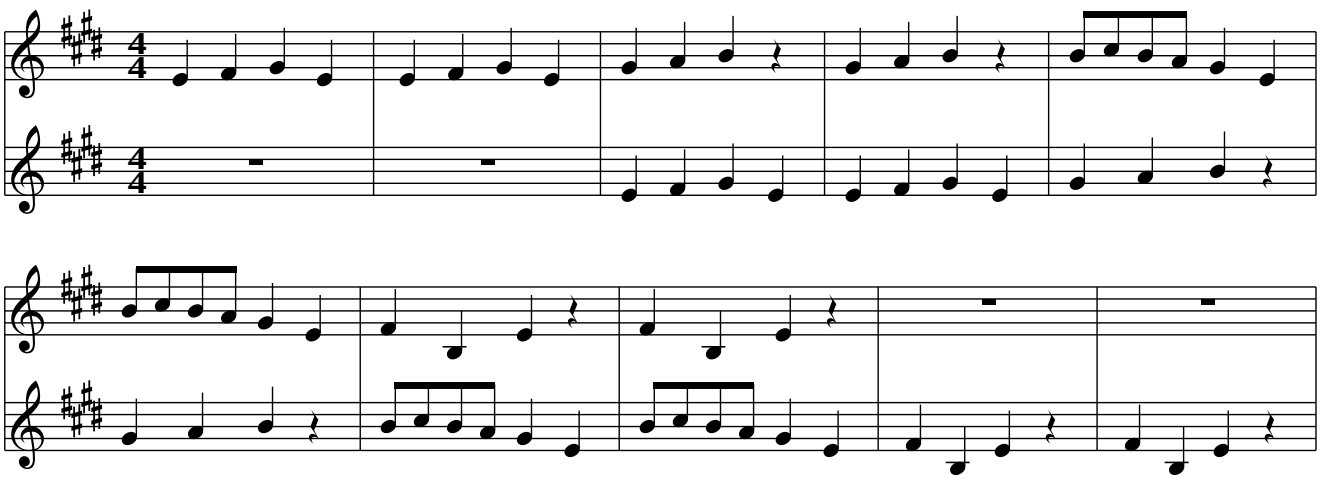
4.1 Voices and Systems: *V:*

Let us review a bit of music theory. There can be one or more lines of music on a staff, that is, one or more *voices*. Voices belong to one or more *instruments*, some of which have a single voice (woodwinds) or more than one (piano, organ). A set of staves related to instruments that play together in the piece is called a *system*.

`abcm2ps` allows to typeset music for up to 16 voices, but this limitation can be easily overcome modifying and recompiling the program sources.

We will begin by writing a piece for two voices on two staves. The `V:` field, followed by a voice name, indicates that the following music belongs to that voice. The voice name may be a number or a string (e.g. 'Tenor'). The `V:` field can be written on a line by itself, or enclosed in square brackets at the start of a note line.

```
X: 1
T: Brother John
C: Traditional
L: 1/4
K:E
V: 1
EFGE|EFGE|GABz|GABz|B/c/B/A/ GE|B/c/B/A/ GE|
V: 2
z4  |z4  |EFGE|EFGE|GABz       |GABz       |
V: 1
FB,Ez      |FB,Ez      |z4   |z4   |
V: 2
B/c/B/A/ GE|B/c/B/A/ GE|FB,Ez|FB,Ez|
```



This score was written alternating the lines of voices 1 and 2, as in real sheet music. We could have written all of the music of voice 1, then all of voice 2: the result would have been the same.

We can add some declarations in the header that specify the properties of each voice. The syntax is:

`V:` ⟨*voice name*⟩ *[definitions]*

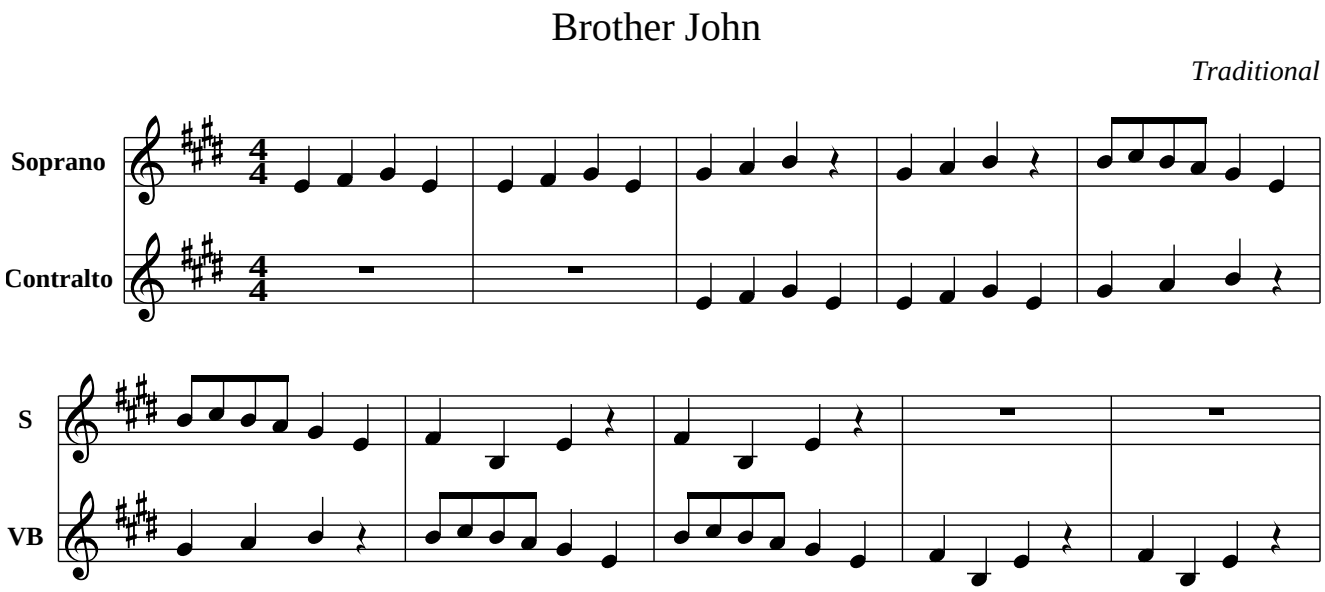
The voice name may be a digit or a word (e.g. 'Tenor'). Possible *definitions* are:

- `clef=` specifies the clef of the voice; you use the same parameters examined in Section 3.1.
- `name=`⟨*name*⟩ or `nm=`⟨*name*⟩ specifies the name that appears at the left of the first staff.
- `sname=`⟨*name*⟩ or `snm=`⟨*name*⟩ specifies the name that appears at the left of all the staves after the first one.
- `merge` indicates that this voice belongs to the same staff as the previous voice.
- `up` or `down` forces the note stem direction.
- `gstem=`⟨*up*⟩, `down` or `auto` forces the grace note stem direction.
- `stem=`⟨*up*⟩, `down` or `auto` forces the note stem direction.
- `dyn=`⟨*up*⟩, `down` or `auto` forces the placement of dynamic marks.
- `lyrics=`⟨*up*⟩, `down` or `auto` forces the placement of the lyrics.
- `staffscale=`⟨*n*⟩ sets the scale of the associated staff. The default value is '1'.
- `stafflines=`⟨*n*⟩ sets the number of lines of the associated staff.

All of these fields are optional. Here is the same tune with some improvements:

```
X: 1
T: Brother John
C: Traditional
L: 1/4
```

```
V: 1 clef=treble name="Soprano" sname="S"
V: 2 clef=treble name="Contralto" sname="VB"
K: E
%
[V: 1] EFGE|EFGE|GABz|GABz|B/c/B/A/ GE|B/c/B/A/ GE|
[V: 2] z4  |z4  |EFGE|EFGE|GABz       |GABz       |
%
[V: 1] FB,Ez      |FB,Ez      |z4   |z4   |
[V: 2] B/c/B/A/ GE|B/c/B/A/ GE|FB,Ez|FB,Ez|
```



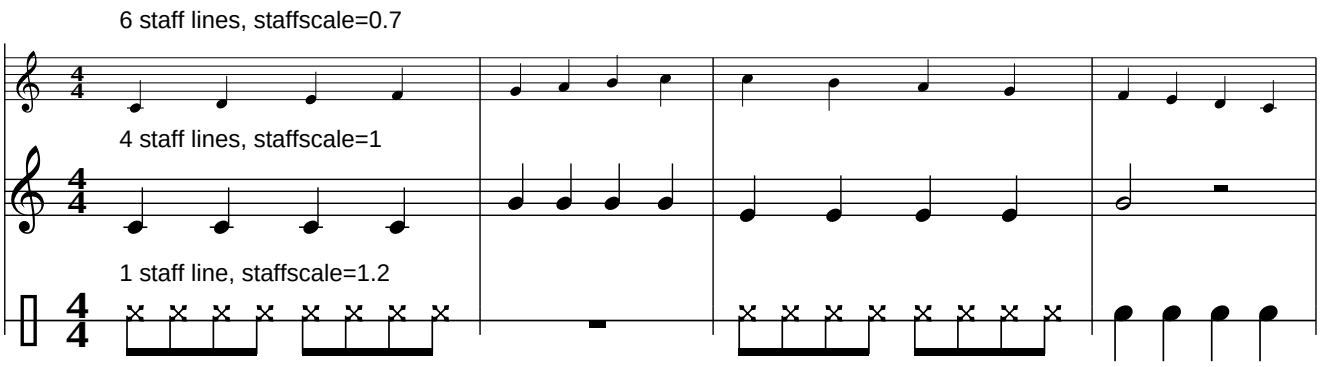
There is a third way to write a `V:` field. You could write it at the start of a music line *without* the square brackets:

```
V:1 CDEF|GABc|
```

This syntax is only accepted by BarFly, an ABC application for the Mac. *Do not ever* write `V:` fields this way, because `abcMIDI` and other applications don't accept this syntax.

Fancy staves can be obtained using `staffscale` and `stafflines`:

```
X: 1
T: Special Staves
M: 4/4
L: 1/4
V: 1 stafflines=6 staffscale=0.7
V: 2 stafflines=4
V: 3 stafflines=1 staffscale=1.2 perc
K: C
%
[V:1] "^6 staff lines, staffscale=0.7"CDEF|GABc|cBAG|FEDC|
[V:2] "^4 staff lines, staffscale=1"CCCC|GGGG|EEEE|G2z2|
[V:3] "^1 staff line, staffscale=1.2"^c/^c/^c/^c/ ^c/^c/^c/^c/|z4| \
      ^c/^c/^c/^c/ ^c/^c/^c/^c/|cccc|
```



4.2 Positioning Voices: `%%staves`

A polyphonic piece is played by several instruments, which have one or two staves associated to them. One or more voices belong to each staff. To specify how voices and instruments are positioned on the score, you use the `%%staves` command.

The `%%staves` command must be followed by voice names, optionally enclosed by a pair of delimiters: `[]`, {}, and `()`. As other commands in the header, `%%staves` must appear before `K:`. In the tune body, if voices exist that were not declared in the header, they will be ignored.

The `%%staves` command starts with `%`, so it should be ignored as a comment. It is not the case; in fact, some commands start with `%%` and are called *meta-comments* or *pseudo-comments*. They are defined this way for compatibility reasons: applications that don't support certain advanced features of ABC PLUS can read the same source just ignoring the meta-comments.
We shall examine meta-comments in detail in Section 5.

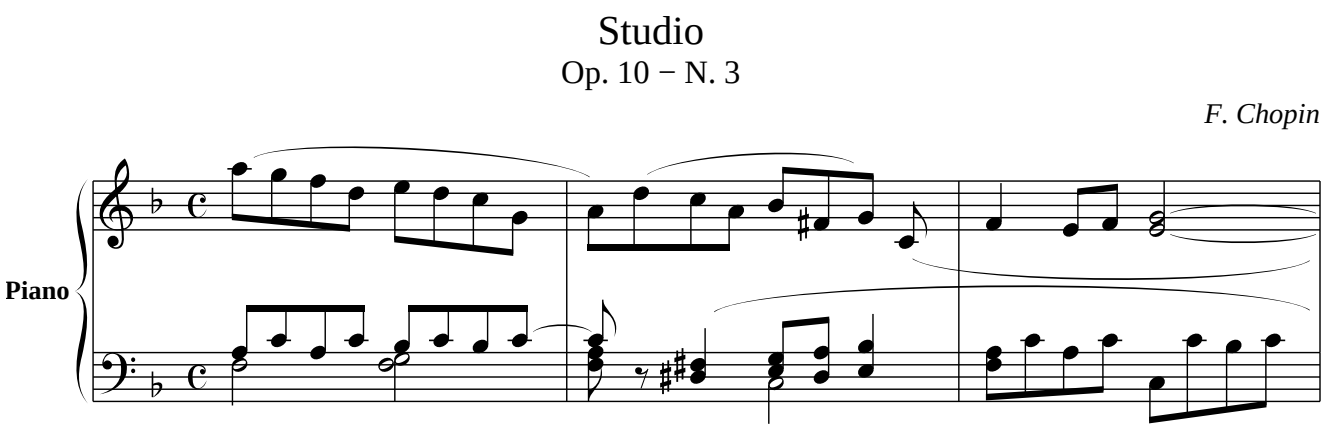
The delimiters are used following these rules:

- when voices are not enclosed by any delimiter, they will be simply printed on separate staves. The uppermost voice in the system will be the first voice in the list. For example: `%%staves SATB`
- when two or more voices are enclosed in square brackets, their staves will be joined by a thick bracket. This arrangement is often used for the choral part of a system. For example: `%%staves [SATB]`
- when two or more voices are enclosed in curly braces, their staves will be joined by a brace. This is typically used for the piano or organ part of a system: `%%staves {MS MD}`
- if two or more voices are enclosed between parentheses, they will be printed on the same staff. For example: `%%staves [(SA) (TB)]`
- by default, measure bars cross the staves. To keep measure bars within each staff, use the character `|` between all voice names: `%%staves [S|A|T|B]`

When two voices are printed on the same staff, the stem direction indicates the first voice (up) or the second (down).

Here is an example of piano music. There are three voices, two of which are played with the left hand. When one of these voices is silent, normal rests are replaced by invisible rests we studied in Section 2.3.

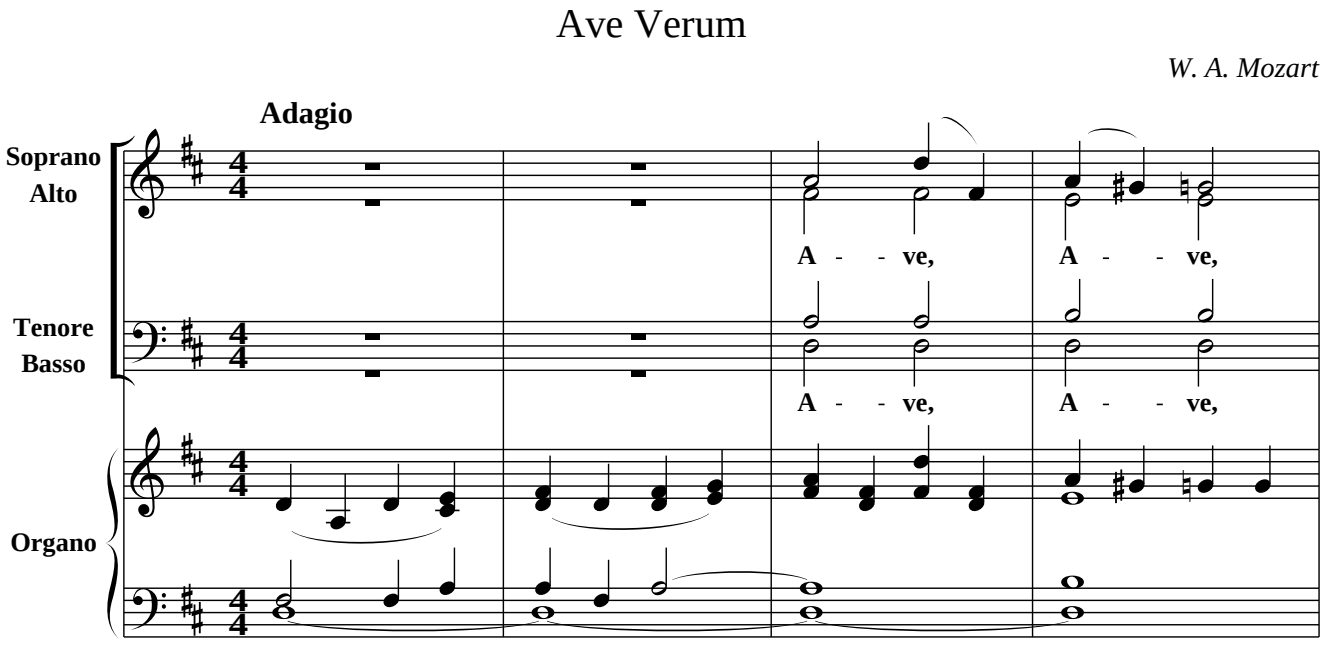
```
X: 1
T: Studio
T: Op. 10 - N. 3
C: F. Chopin
M: C
%%staves {RH1 (LH1 LH2)}
V: RH1 clef=treble name="Piano"
V: LH1 clef=bass
V: LH2 clef=bass
K: F
%
[V: RH1] (agfd edcG    |A)(dcA B^FG) (C            |F2 EF [E4G4]-   |
[V: LH1] ac'ac' bc'bc'-|c'z ([^d2^f2][eg][da][e2b2]|[fa]c'ac' cc'bc'|
[V: LH2] f4 [f4g4]     |[fa] x x2 c4               |x4 x4           |
%
```



Let us now try a more complex piece. These are the first four measures of Mozart's famous 'Ave Verum', for organ and SATB:

```
X: 1
T: Ave Verum
C: W. A. Mozart
M: 4/4
L: 1/4
Q: "Adagio"
%%staves [(S A) (T B)] {(MD1 MD2) (MS1 MS2)}
V: S clef=treble name="Soprano" sname="S"
V: A clef=treble name="Alto" sname="A"
V: T clef=bass name="Tenore" sname="T"
V: B clef=bass name="Basso" sname="B"
V: MD1 clef=treble name="Organo"
V: MD2 clef=treble
V: MS1 clef=bass
V: MS2 clef=bass
K: D
%
```

```
[V: MD1] (DA,D[CE])|([DF]D[DF][EG])|[FA][DF][Fd][DF]|A^G=GG  |
[V: MD2] x4        |x4              |x4              |E4      |
[V: MS1] f2fa      |afa2-           |a4              |b4      |
[V: MS2] d4-       |d4-             |d4-             |d4      |
[V: B] z4          |z4              |d2d2            |d2d2    |
w: A- ve, A- ve,
[V: T] z4          |z4              |a2a2            |b2b2    |
[V: A] z4          |z4              |F2F2            |E2E2    |
[V: S] z4          |z4              |A2(dF)          |(A^G)=G2|
w: A- ve, * A - ve,
```



Note that the voices were intentionally written in reverse order. The `%%staves` command rearranged staves and voices in the right order. Normally, you will want to write voices in the same order as specified in `%%staves`.

The `%%staves` command is a strong point of the ABC PLUS notation compared to graphical programs. For example, in a four voice score laid out for SATB, you only need to modify the `%%staves` command to change the layout to two staves, two voices per staff. With most graphical programs, you would have to rewrite the score from scratch!

In general, writing the voices in the same order as they appear in a real score is preferable.

As a last example, a piece written in an unusual manner: the 'Kyrie' from Andrea Gabrieli's Missa Brevis. This music has no metre, and each voice follows its own tempo: in this situation `M:none` must be used. The length of each measure is different for each voice, consequently the `!longphrase!` symbol replaces measure bars. We also want 'cut time' tempo indicated. This is how the piece is written:

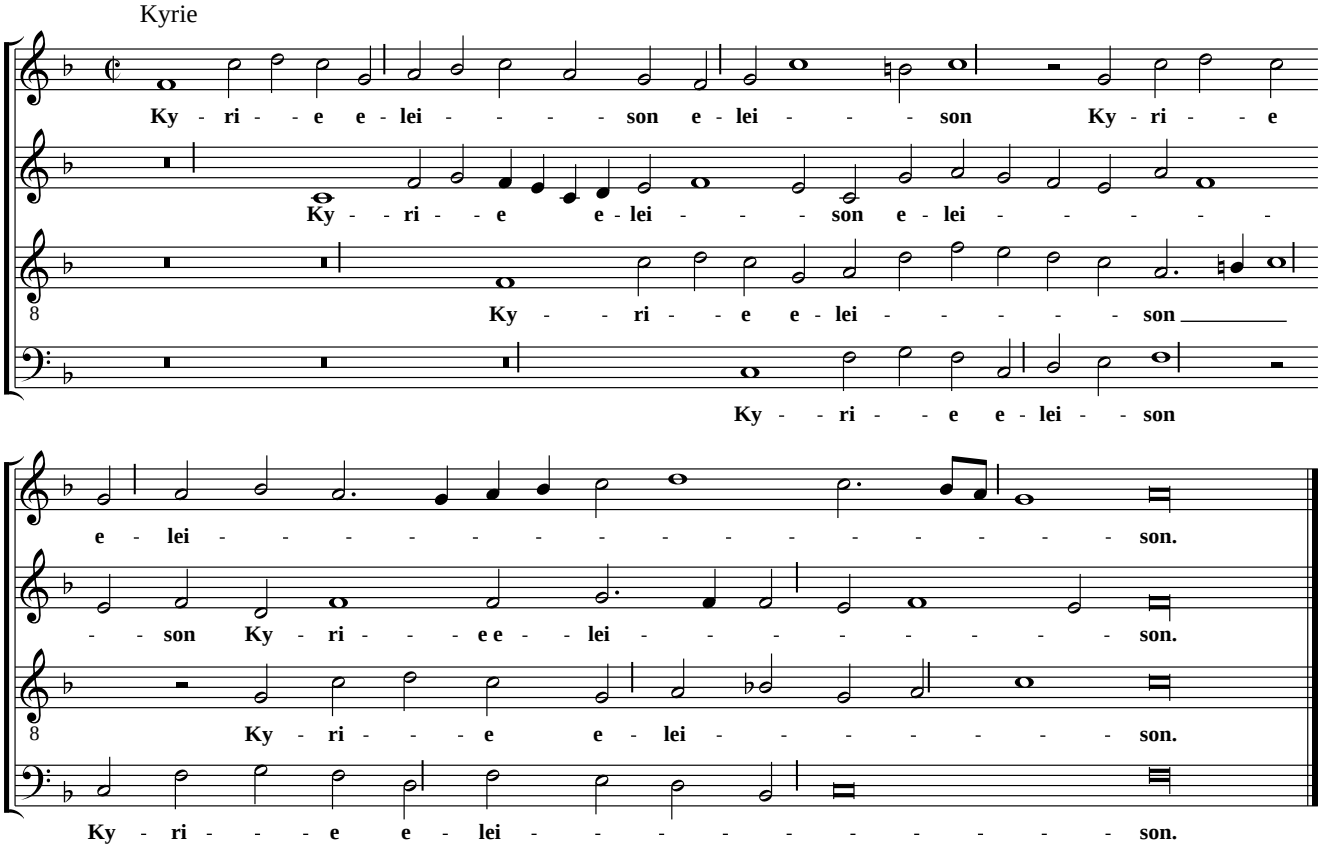
```
X: 1
T: Missa Brevis
C: Andrea Gabrieli (1510? - 1586)
M: C|
L: 1/4
```

```
%%staves [1 2 3 4]
V: 1 clef=treble
V: 2 clef=treble
V: 3 clef=treble-8
V: 4 clef=bass
U: L = !longphrase!
K: F
%
[P: Kyrie]
[V: 1] [M:none] F4 c2d2c2LG2 A2B2c2A2G2LF2 G2 c4 =B2 Lc4 z2 G2
w: Ky- ri - e e- lei - - - son e- lei - - son Ky-
[V: 2] [M:none] Lz8 C4 F2G2 FECD E2 F4 E2C2G2A2G2F2E2
w: Ky- ri - e * * e- lei - - son e- lei - - -
[V: 3] [M:none] z8 Lz8 F4 c2d2c2G2A2d2f2e2d2c2
w: Ky- ri - e e- lei - - - - -
[V: 4] [M:none] z8 z8 Lz8 c4 f2g2f2Lc2 d2e2
w: Ky- ri - e e- lei -
%
[V: 1] c2d2c2LG2 A2B2A3 GAB c2 d4 c3 B/LA/ G4 A16 |]
w: ri - e e- lei - - - - - - - - - - - son.
[V: 2] A2 F4 E2F2D2 F4 F2 G3 F LF2 E2 F4 E2 F16 |]
w: - - - son Ky- ri- e˜e- lei - - - - - son.
[V: 3] A3 =B Lc4 z2 G2c2d2c2LG2 A2_B2G2LA2 c4 c16 |]
w: son__ Ky- ri - e e- lei - - - - son.
[V: 4] Lf4 z2 c2f2g2f2Ld2 f2e2d2LB2 c8 f16 |]
w: son Ky- ri - e e- lei - - - - son.
```

Missa Brevis

Andrea Gabrieli (1510? − 1586)

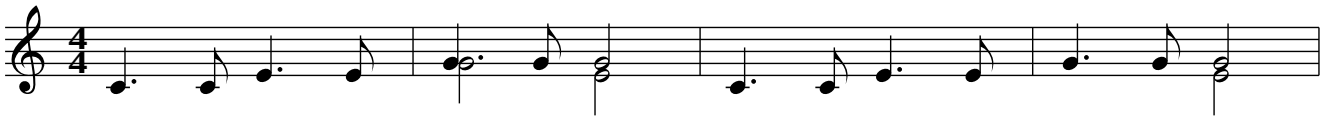
Kyrie



4.3 Voice Splitting: *&*

In some pieces of music, a voice splits in two in some measures only. To avoid introducing a supplementary, almost identical voice, you can use the `&` symbol. When placed within a measure, it splits the current voice and attributes the notes that follow to the 'second' voice. This is also called 'voice overlay'.

```
X: 1
L: 1/4
K: C
C>CE>E|G>GG2 & G2E2|C>CE>E|G>GG2 & x2E2|
```



To end a voice overlay, `&)` may be used.

This is how the above piece can be equivalently written:

```
X: 1
L: 1/4
%%staves (1 2)
K: C
[V:1] C>CE>E|G>GG2|C>CE>E|G>Gc2|
[V:2] x4    |G2E2 |x4    |x2E2 |
```

4.4 Change of System

In pieces of some complexity (say, for soloist, choir, and orchestra), not all instruments play at the same time. Writing all parts, mostly containing rests, would be a waste of time and space when only one instrument is playing.

The `%%staves` field can be changed as needed, specifying only the instruments that are playing. Here is as example 'Riu riu chiu', a well-known 16th century villancico. The result is shown in Figure 4.

Please note that strange `\241` in the title. It is the octal code of the `¡` character in the ISO 8859-1 (Latin1) character set. We will cover this topic in Appendix D.

```
%%scale 0.68
%%barsperstaff 6
X: 1
T: Riu, riu, chiu, \241la guarda ribera!
C: Villancico (Spain, XVIth century)
M: C|
L: 1/2
Q: 1/2 = 240
%%staves 3
V: 3 clef=treble-8 name="Tenor\nBass"
K: Am
```

```
% MEN ONLY
[V: 3] [M:none] AAGA|F2ED2EFG|A2A2|
w: Ri-u, ri-u, chi-u, \241la guar-da ri-be-ra!
[V: 3] AAGA         |F2EG2GEF|D2D2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra,
[V: 3] AAGA         |F2EG2GEF|D2D2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra.
% SYSTEM CHANGE: ALL
%%staves [1 2 3 4]
V: 1 clef=treble   name="S" sname="S"
V: 2 clef=treble   name="A" sname="A"
V: 3 clef=treble-8 name="T" sname="T"
V: 4 clef=bass     name="B" sname="B"
[V: 1]AAGA|F2ED2EFG |A2A2z2|
w: Ri-u, ri-u, chi-u, la guar-da ri-be-ra!
[V: 2]FFEC|D2EF2EDD |C2C2z2|
[V: 3]cccG|A2AA2ADD |E2E2z2|
w: Ri-u, ri-u, chi-u, la guar-da ri-be-ra!
[V: 4]ffcf|d2Ad2c_BB|A2A2z2|
%
[V: 1] z4      |AAGA|F2EF2FEE|D2D2z2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra,
[V: 2] z2EE    |DCEC|D2CD2DCC|D2D2z2|
w: Di\'os guar-d\'o el lob', el lo-bo de nue-stra cor-de-ra,
[V: 3] ccBc    |A2BA|A2AA2AAA|A2A2z2|
w: Di\'os guar-d\'o el lo-bo, el lo-bo de nue-stra cor-de-ra,
[V: 4] aaga    |f2ef|d2Ad2dAA|d2d2z2|
%
[V: 1] z4  |AAGA|F2ED2DCC    |D2D2  |
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra.
[V: 2] z2EE|DCEC|D2CA,2A,A,A,|A,2A,2|
w: Di\'os guar-d\'o el lob', el lo-bo de nue-stra cor-de-ra.
[V: 3] ccBc|A2BA|A2AF2FEE    |D2D2  |
w: Di\'os guar-d\'o el lo-bo, el lo-bo de nue-stra cor-de-ra.
[V: 4] aaga|f2ef|d2Ad2dAA    |d2d2  |
% SYSTEM CHANGE: MEN ONLY
%%staves 3
[V: 3] AAGA|F2EG2GEF|D4|AAGA|
w: El lo-bo ra-bio-so la qui-so mor-der, Mas Di\'os po-de-
[V: 3] F2FEGGEF|D4|AAGA|F2FEDEFG|
w: ro-so la su-po de-fen-der; qui so-le ha-ce que no pu-die-sce pe-
[V: 3] A4|AAGA|F2FEGGEF|D2D2|
w: car: ni~aun o-ri-gi-nal e-sta Vir-gen no tu-vie-ra.
```

An alternative way to write this piece would be to split it into three separate tunes, each corresponding to a system. You can write it this way if you wish, but the disadvantage is that MIDI conversion will produce three different files instead of one.

♫♫♫♫♫♫♫

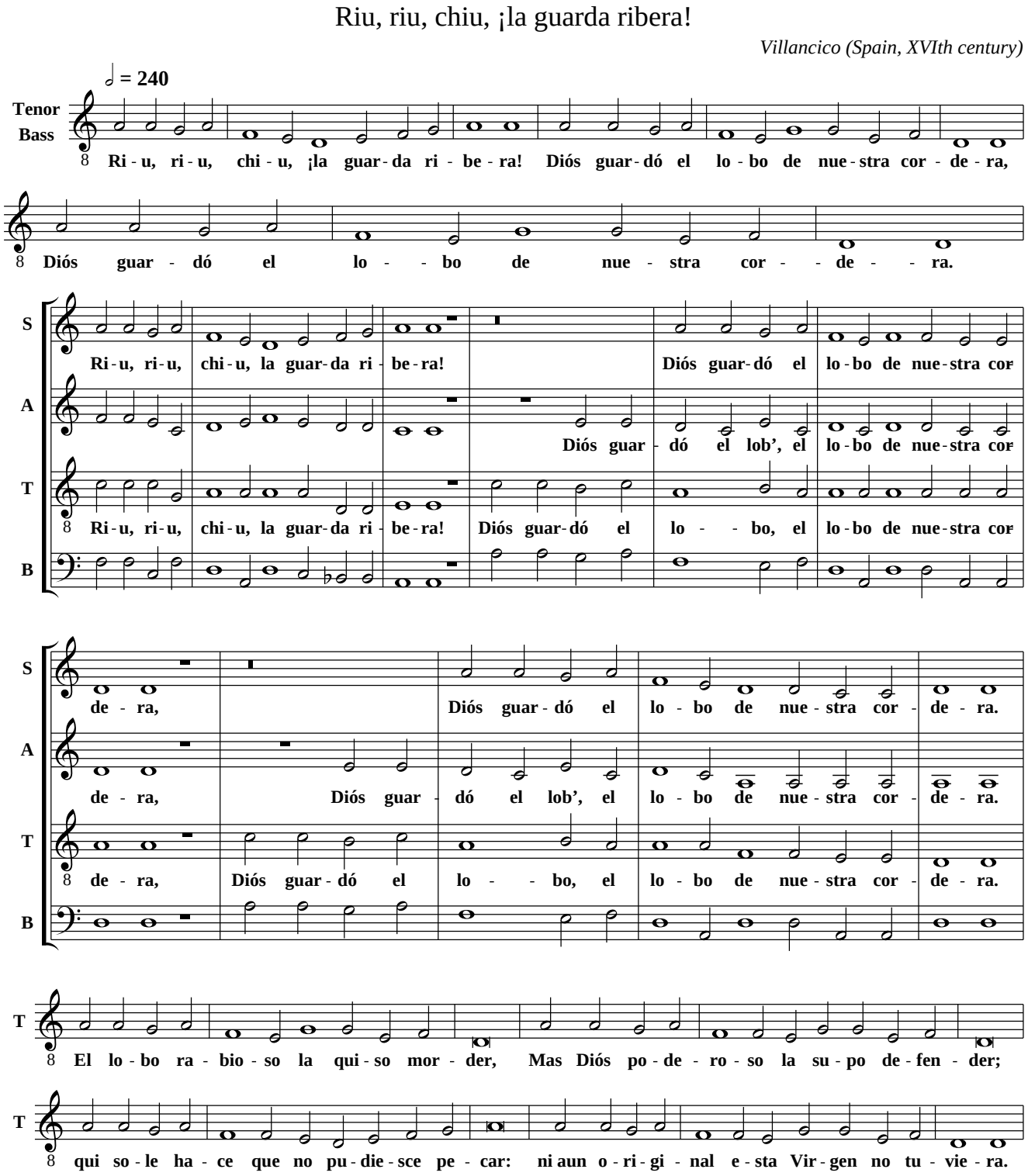


Figure 4: A piece where the system changes three times.

Part IV
Page Layout

5 Formatting Parameters

We have learned how to write polyphonic music. Now we will want to set the page layout, the fonts, etc. `abcm2ps` has several commands for customising formatting parameters. These commands are written in the source as meta-comments, or in external files known as *format files*.

Meta-comments (from now on, *commands*) are lines that start in `%%`, like `%%staves`. These are written in the header or in the body. There exist several commands: some specify the page layout, fonts, spacing, and so on. Many commands accept a parameter of one of these types:

- a *unit of length*, set in centimeters (cm), inches (in), or points (pt): for instance, `30pt`, `1cm`, `0.3in`;
- a *logical value* 'yes or no', expressed using the words `true` or `false` or, equivalently, with 1 or 0;
- a *string*, like `Times-Roman 24`;
- a *number*, either integer or real (that is, with decimals).

Let us now see a rather complete example. The following piece (the first ten measures of Mozart's Ave Verum) contains the most commonly used commands:

```
% PAGE LAYOUT
%
%%pageheight     29.7cm
%%pagewidth      21cm
%%topmargin      1cm
%%botmargin      1cm
%%leftmargin     1cm
%%rightmargin    1cm
% SPACING
%%topspace       0cm     % space before the piece
%%titlespace     0cm     % space before the title
%%subtitlespace  0.2cm   % space before the subtitle
%%composerspace  0.5cm   % space before the composer line
%%musicspace     0.5cm   % space before the first staff
%%vocalspace     1.5cm   % additional space after lyrics lines
%%sysstaffsep    1cm     % space between staves in the same system
%%staffsep       2cm     % space between different systems
% FONT
%%titlefont      Times-Bold 32
%%subtitlefont   Times-Bold 24
%%composerfont   Times-Italics 16
%%vocalfont      Times-Roman 14  % for lyrics
%%gchordfont     Times-Bold 14   % for chords
% MISC
%%measurebox     true  % measure numbers in a box
%%measurenb      0     % measure numbers at first measure
%%exprabove      true  % expressions above the staff
%%barsperstaff   5     % number of measures per staff
%%scale          0.7   % magnification
%
```

```
X: 1
T: Ave Verum
T: per coro e organo
C: W. A. Mozart (1756-1791)
M: 4/4
L: 1/4
Q: "Adagio"
%%staves [(1 2) (3 4)] {(5 6) (7 8)}
V: 1 clef=treble name="Soprano" sname="S"
V: 2 clef=treble name="Alto" sname="A"
V: 3 clef=bass name="Tenore" sname="T"
V: 4 clef=bass name="Basso" sname="B"
V: 5 clef=treble name="Organo"
V: 6 clef=treble
V: 7 clef=bass
V: 8 clef=bass
K: D
% 1 - 5
[V: 1] z4        |z4            |A2(dF)          |(A^G)=G2|(GB)(AG)        |
w: A- ve,_ a - ve, ve - rum_
[V: 2] z4        |z4            |F2F2            |E2E2    |(EG)(FE)        |
[V: 3] z4        |z4            |a2a2            |b2b2    |a2a2            |
w: A- ve, a- ve, ve- rum
[V: 4] z4        |z4            |d2d2            |d2d2    |c2c2            |
[V: 5] (DA,D[CE])|([DF]D[DF][EG])|[FA][DF][Fd][DF]|A^G=GG  |[EG][GB][FA][EG]|
[V: 6] x4        |x4            |x4              |E4      |x4              |
[V: 7] f2fa      |afa2-         |a4              |b4      |a4              |
[V: 8] d4-       |d4-           |d4-             |d4      |c4              |
% 6 - 10
[V: 1] (GF)F2          |E3E |FFGG         |(G2F)F      |E4    |
w: cor - pus na- tum de Ma- ri- a Vir - gi- ne,
[V: 2] (ED)D2          |C3C |DDEE         |(E2D)D      |C4    |
[V: 3] a2a2            |a3a |aaaa         |a3a         |a4    |
w: cor- pus na- tum de Ma- ri- a Vir- gi- ne,
[V: 4] d2d2            |A3A |ddcc         |d3d         |A4    |
[V: 5] [EG][DF][DF][FA]|AEEA|[FA][df][eg]G|[E2G2][D2F2]|[C4E4]|
[V: 6] x4              |C2C2|DAAD         |x4          |x4    |
[V: 7] a4              |a4  |a3a          |a4          |x4    |
[V: 8] d4              |A4  |d2c2         |ddfd        |Aaec  |
```

The result is shown in Figure 5. Quite a big change, isn't it? The difference should be clear. A complete list of available commands is presented in Appendix E.

5.1 Changing parameters

Once the parameters are set, their values remain the same throughout the entire piece. But of course, parameters can be redefined in the body of the tune.

There are two ways to change a parameter: either write a new command line, or an equivalent `I:` inline field. This field accepts a special syntax that has the same effect of a command, with the additional advantage of being able to appear anywhere in the tune body. For example, `I:topmargin 1cm` is equivalent to `%%topmargin 1cm`.

Let's use both methods to change the `vocalfont` parameter in a tune:

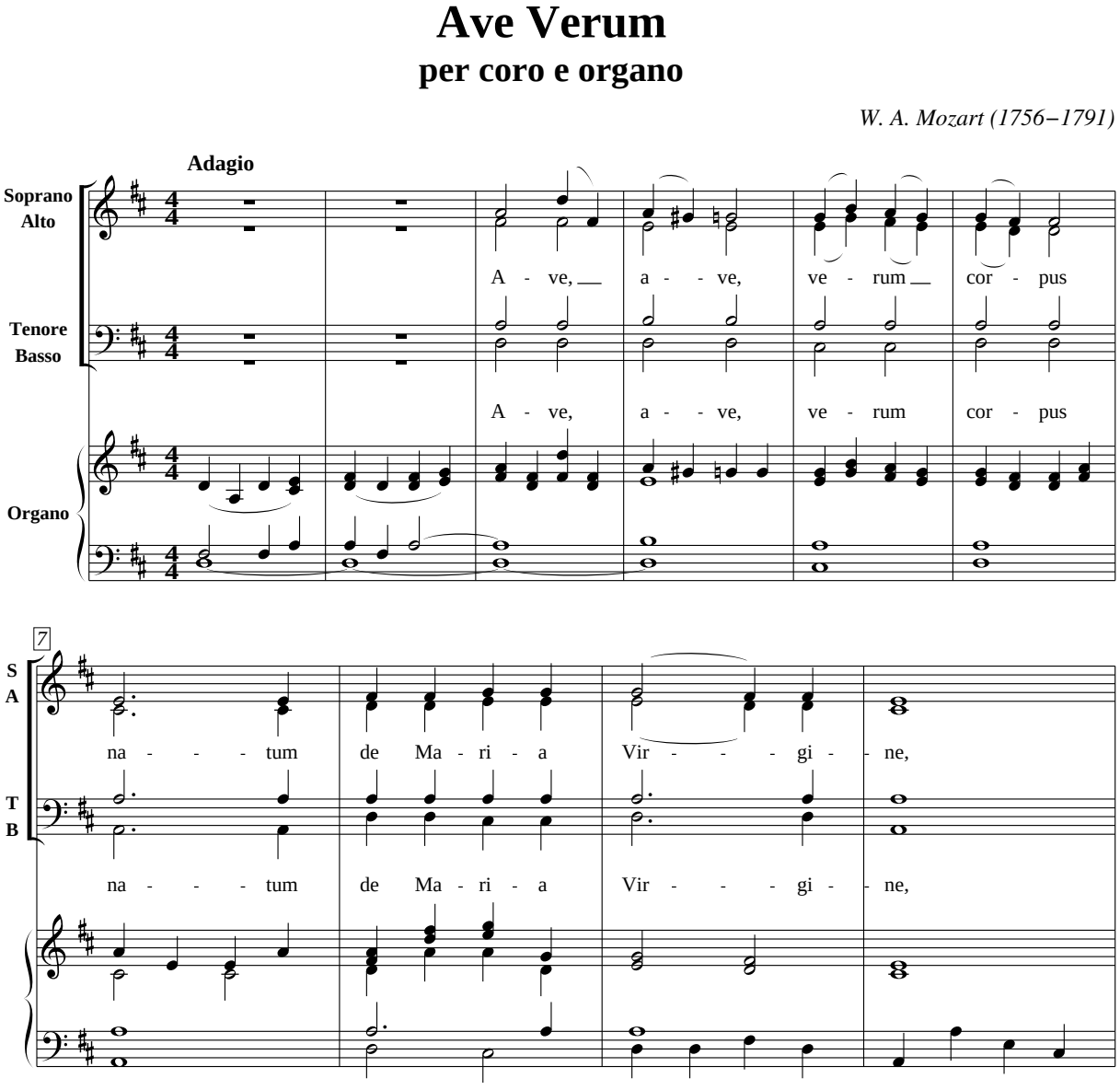
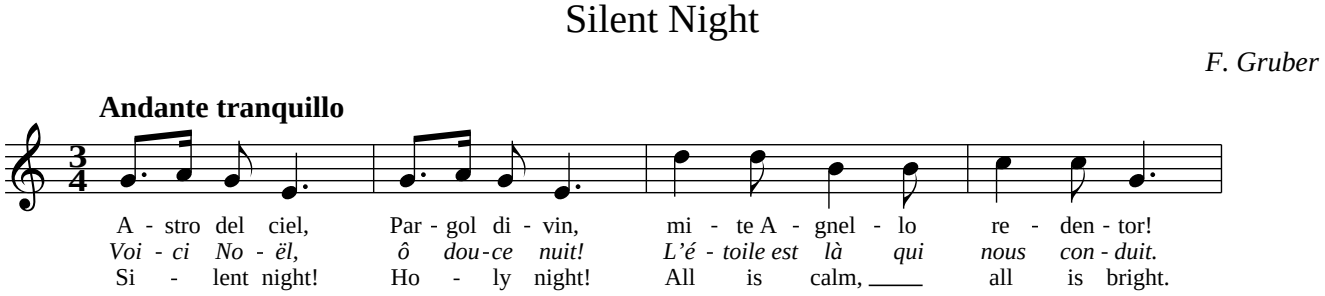


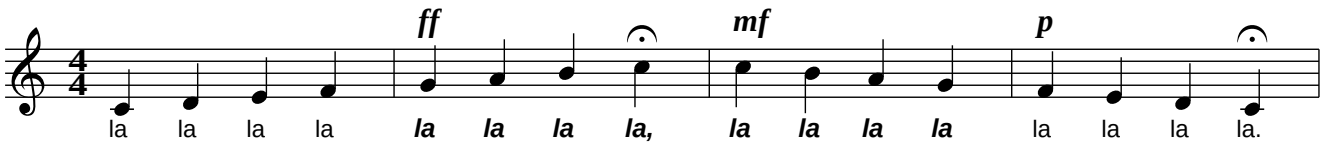
Figure 5: Ave Verum with formatting parameters.

```
X: 1
T: Silent Night
C: F. Gruber
M: 3/4
Q: "Andante tranquillo"
K: C
%
G>A G E3|G>A G E3|d2 d B2 B|c2 c G3|
%%vocalfont Times-Roman 12
w: A- stro del ciel, Par- gol di- vin, \
w: mi- te˜A- gnel- lo re- den- tor!
[I: vocalfont Times-Italic 12]
w: Voi- ci No- \"el, \^o dou- ce nuit! \
w: L'\'e- toile˜est l\`a qui nous con- duit.
%%vocalfont Times-Roman 12
w: Si - lent night! Ho - ly night! All is calm,_ all is bright.
```



This method can also be used to change the font in the same line:

```
%%font Helvetica
%%font Helvetica-BoldOblique
X: 1
L: 1/4
K: C
CDEF|!ff!GAB!fermata!c|!mf!cBAG|!p!FED!fermata!C|
%%vocalfont Helvetica 12
w: la la la la\
%%vocalfont Helvetica-BoldOblique 13
w: la la la la, la la la la\
%%vocalfont Helvetica 12
w: la la la la.
```



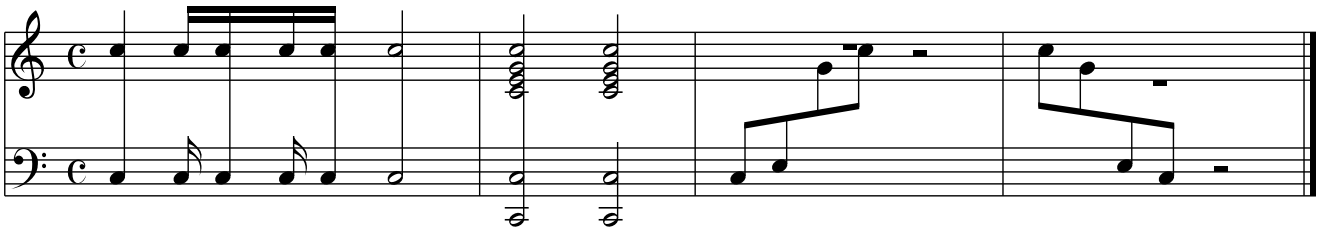
A better way to change the fonts will be explained in Section 5.3.

5.2 The Grand Staff

In piano music, the most commonly used system consists of two staves. Notes and stems are allowed to cross them.

The `I: staff` ⟨*n*⟩ field is used to force the position of notes on a specific staff, while `!xstem!` draws a stem up to the note on the previous staff. Top and bottom staves are numbered 1 and 2.

```
X:1
M:C
L:1/4
K:none
%%staves 1 2
V:1
cc//c//c//c2 | [CEGc]2[CEGc]2|
V:2 bass
!xstem!C,C,//!xstem!C,//C,//!xstem!C,//!xstem!C,2 | !xstem![C,,C,]2[C,,C,]2|
V:1
z4|c/G/[I: staff 2]E,/[I: staff 2]C,/ z2|]
V:2
C,/E,/[I: staff 1]G/[I: staff 1]c/ z2|z4|]
```



5.3 Using Fonts

`abcm2ps` supports nearly all POSTSCRIPT fonts, which are listed in Appendix G. Three fonts are especially important: Times, Helvetica, and Courier; all with italics and bold variants. Times is equivalent to Windows' Times New Roman, Helvetica is equivalent to Arial, and Courier to Courier New.

These are predefined fonts, which you can use anytime with any font command. To use other fonts, you have to *declare* them inserting the `%%font` command at the top of the source. Here is an example that demonstrates `abcm2ps`'s capability of alternating text in different fonts with pieces of music. The result is shown in Figure 6.

```
% declare non-predefined fonts
%%font AvantGarde-Book
%%font Bookman-Light
%
%%titlefont Times-Italic 21
%%musicspace -0.5cm
%%textfont Helvetica 26
%%center Typesetting example
%%vskip 0.4cm
%%textfont Bookman-Light 14
%%begintext justify
```

```
This is an example of text inserted into an ABC file. This abcm2ps
feature allows for the writing of songbooks, music collections or other
publications without having to resort to a word processor. Not bad, is
it? Now let's write a brief musical example.
%%endtext
X: 1
T: Etude
M: 4/4
L: 1/4
Q: "Dolcemente"
K: C
%
!p!CCGG|AA!mf!G2|!diminuendo(!FFEE|DD!diminuendo)!C2|

%%vskip 0.4cm
%%textfont AvantGarde-Book 14
%%begintext align
Now we'll have a look at something more lively. To start with, let's
switch fonts: from Bookman-Light to AvantGarde-Book. Here is the same
Etude with a few small variations to make it more interesting:
%%endtext
X: 2
T: Etude
T: second version
M: 4/4
L: 1/4
Q: "Adagio"
K: C
%
.C{DCB,}C.G{AGF}G|A>AG2|.F{GFE}F.E{FED}E|D>DC2|
%%sep 0.4cm 0.4cm 6cm
% the following line increases the character size
%%textfont * 20
%%center End of the example.
%%sep 0.4cm 0.4cm 6cm
```

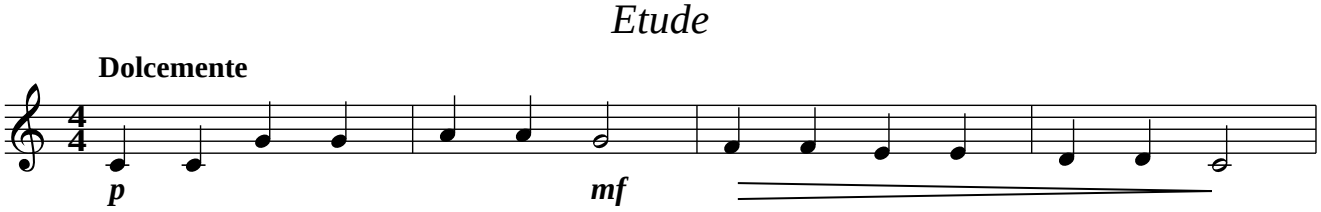
Virtually all printed music uses Times-Roman or an equivalent font. However, Helvetica is more readable at equal font size.

To use different fonts in string, you specify an alternate set of fonts with the commands `%%setfont-1`, `%%setfont-2`, `%%setfont-3`, and `%%setfont-4`, followed by a font name and a font size. These can be referred to in strings as `$1 $2 $3 $4`, and have the effect of changing the text font:

```
%%setfont-1 Times-Roman 20
%%setfont-2 Times-Italic 26
%%setfont-3 Helvetica-Bold 18
%%setfont-4 AvantGarde-Demi 24
X: 1
K: C
%%text Hello. This is the default font, $1this is font 1,
%%text $2this is font 2, $3this is font 3, $4this is font 4,
```

Typesetting example

This is an example of text inserted into an ABC file. This abcm2ps feature allows for the writing of songbooks, music collections or other publications without having to resort to a word processor. Not bad, is it? Now let's write a brief musical example.



Now we'll have a look at something more lively. To start with, let's switch fonts: from Bookman–Light to AvantGarde–Book. Here is the same Etude with a few small variations to make it more interesting:

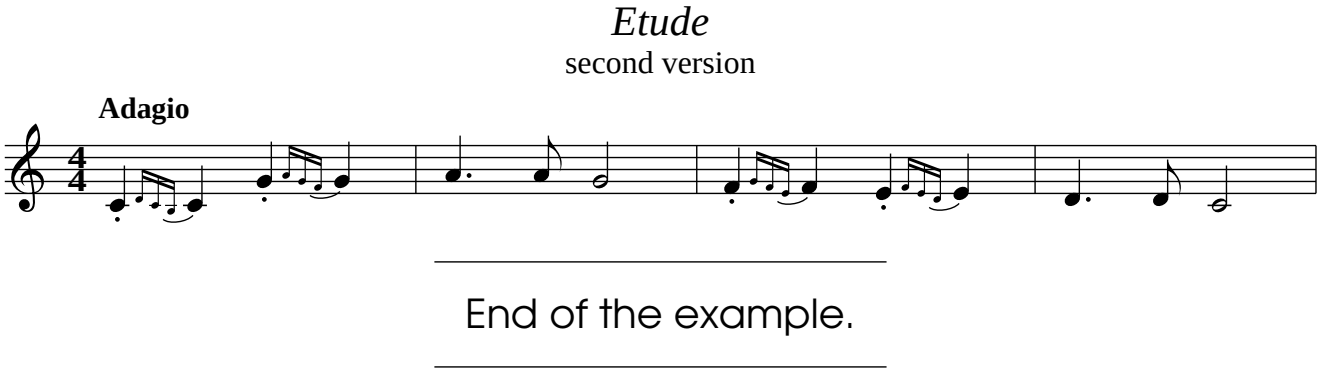
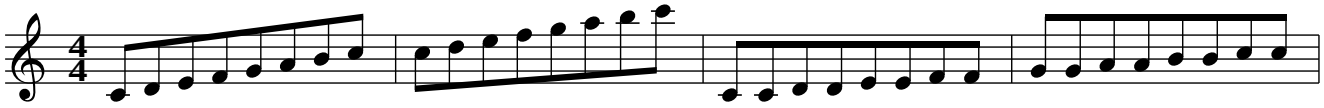


Figure 6: Alternating text with music.

```
%%text $4and now $0let's go back to default font.
CDEFGABc|cdefgabc'|CCDDEEFF|GGAABBcc|
W: It also works in $3W: fields!
W: It's useful to emphasise $0some parts.
```

Hello. This is the default font, this is font 1,
this is font 2, **this is font 3**, **this is font 4**,
and now let's go back to default font.



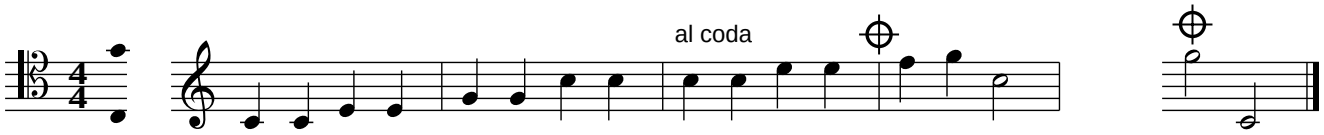
It also works in **W: fields!**
It's useful to emphasise some parts.

Figure 7: Using different fonts in strings

5.4 Staff Breaks

To give an indication at the beginning of a piece (for example, the original key signature and extension), or write a coda, the staff can be interrupted with the `%%staffbreak` command:

```
X: 1
L: 1/4
K: C alto4
%
[C0g0]\
%%staffbreak 0.3cm
K: C treble
CCEE|GGcc|"^al coda"ccee!coda!|fgc2|\
%%staffbreak 1.5cm
!coda!g2C2|]
```



If the staff is part of a system, then the staff break must be applied to all staves in the system.

5.5 Multi-column Output

Text and music can be laid out in multiple columns on the page. The `%%multicol start`, `%%multicol new` and `%%multicol end` commands define the columns.

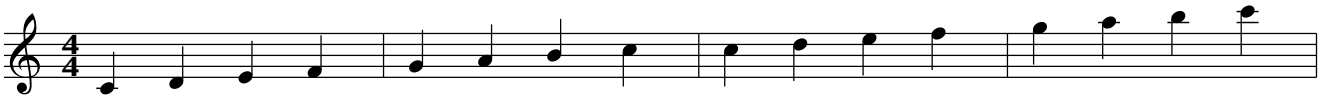
`%%multicol start` saves the current page margins and sets the vertical position for the beginning of a column. At this point you can change the margins and print the material in the first column.

`%%multicol new` moves the vertical position to the beginning of a new column, resetting the margins. Change the margins again and print the material in this new column. This sequence may be repeated as many times as you wish.

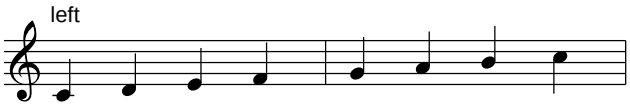
Finally, `%%multicol end` reinitializes the page margins to the values prior to `%%multicol start` and moves the horizontal position below the columns that were printed.

It sounds difficult, doesn't it? Don't worry, it is easier than it sounds. Here is an example:

```
%%pagewidth 21cm
%%leftmargin 1cm
%%rightmargin 1cm
X: 1
L: 1/4
K: C
CDEF|GABc|cdef|gabc'|
%%multicol start
%%rightmargin 11cm
%%begintext justify
%%Sator arepo tenet opera rotas. Sator arepo tenet opera rotas.
%%Sator arepo tenet opera rotas. Sator arepo tenet opera rotas.
%%endtext
"^left"CDEF|GABc|
%%text Left column (margins: 1, 11)
%%text Width: 21 - 1 - 11 = 9 cm
%%multicol new
%%leftmargin 13cm
%%rightmargin 2cm
%%begintext justify
%%Sator arepo tenet opera rotas. Sator arepo tenet opera rotas.
%%Sator arepo tenet opera rotas.
%%endtext
"^right"cdef|gabc'|
%%text Right column (margins: 13, 2)
%%text Width: 21 - 13 - 2 = 6 cm
%%multicol end
CDEF|GABc|cdef|gabc'|
```



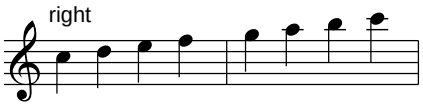
Sator arepo tenet opera rotas. Sator arepo tenet opera rotas. Sator arepo tenet opera rotas. Sator arepo tenet opera rotas.



Left column (margins: 1, 11)

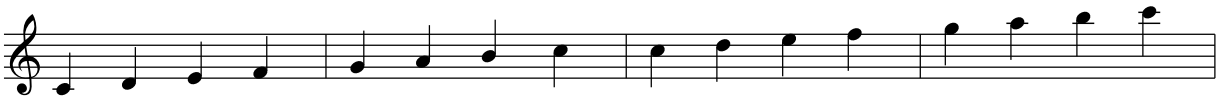
Width: 21 − 1 − 11 = 9 cm

Sator arepo tenet opera rotas. Sator arepo tenet opera rotas. Sator arepo tenet opera rotas.



Right column (margins: 13, 2)

Width: 21 − 13 − 2 = 6 cm



5.6 Customising Titles

In addition to the plain `T:` line(s), the tune title may be composed by several fields.

Complex titles are specified using the `%%titleformat`, followed by a set of letters, digits, and commas. Letters refer to ABC PLUS fields, and may be ‘A’, ‘B’, ‘C’, ‘D’, ‘H’, ‘N’, ‘O’, ‘P’, ‘R’, ‘S’, ‘T’, ‘X’, or ‘Z’; including a letter, the corresponding field will be printed. Digits may follow a letter, meaning ‘0’ for centre, ‘1’ for right align, or ‘-1’ for left align. A comma `,` forces a newline, and unrecognized characters are ignored.

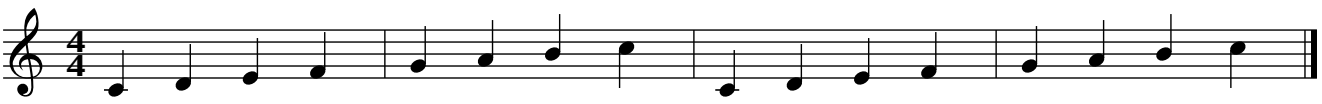
```
%%titleformat T-1 R1, T-1 T-1 C1, H1
X: 1
T: First Title
T: Second Title
T: Last Title
C: Anonymous
H: Written many many years ago...
L: 1/4
R: march
K: C
CDEF|GABc|CDEF|GABc|]
```

First Title *march*

Second Title *Anonymous*

Last Title

Written many many years ago...



5.7 Headers and Footers

The following commands define the text that is to appear automatically on every page: `%%header` for the page header, and `%%footer` for the page footer. These commands, followed by text, will print it by default centred on the page.

Three areas may be defined: left, centre, and right, each with different text. If you define areas, the line of text should be enclosed in double quotes. Furthermore, the text may use special symbols to insert specific information about the piece:

- `$D` prints the current date and time;
- `$F` prints the name of the current file;
- `$T` prints the title of the current tune;
- `$P` prints the page number;
- `$P0` and `$P1` print the page number, but only if it is even or odd;
- `$V` prints `abcm2ps-` followed by the version number;
- `\n` indicates the start of a second line of text;
- `$d` prints date and time of the last modification of the current input file.

The three fields must be separated by a *tab character* (see Section 1.6.) If you use JEDABC, I suggest that you use the lines `%%header` and `%%footer` obtained from the Mode/abcm2ps Options/page Layout menu, and if necessary, modify them.

Here is an example of the command `%%footer` used to print the even page numbers on the left, the name of the piece in the centre, and the odd page numbers on the right. Please note that the areas are *not* separated by spaces, but by tabs!

```
%%footer "$P0    $N      $P1"
```

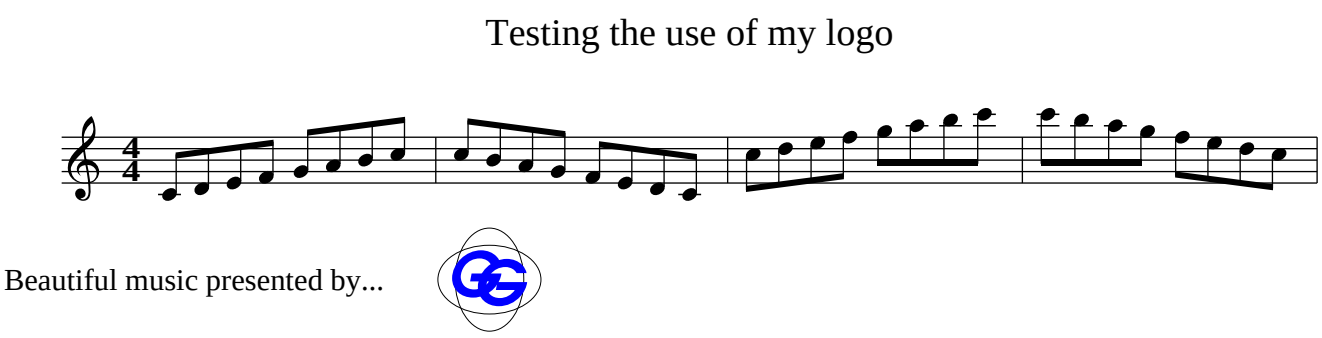
If you need to change headers and/or footers after a new page, insert their new definition *before* the `%%newpage` command.

5.8 Inserting Graphics Files

Another interesting possibility is the addition of external EPS files in the source, perhaps to add a logo or a drawing to the score. You use the `%%EPS` command followed by the name of the file to insert:

```
X: 1
T: Testing the use of my logo
K: C
CDEF GABc |cBAG FEDC |
cdef gabc'|c'bag fedc|
%%multicol start
%%leftmargin 1cm
%%rightmargin 10cm
```

```
%%text
%%text Beautiful music presented by...
%%multicol new
%%leftmargin 7cm
%%rightmargin 1cm
%%EPS logo.eps
%%multicol end
```



If the file to be included is in another standard graphics format (e.g. JPG), you will have to convert it to EPS using an appropriate program. Please consult Section 17.

♫♫♫♫♫♫♫

6 Format files

Although you can insert formatting parameters in the source, it may be more practical to write them in an external file that is used by `abcm2ps` when it formats the piece. This file is called a *format file.*

This is how a format file is written:

```
% format file

scale 0.8
topmargin 2 cm
titlefont Helvetica-Bold 13
subtitlefont Helvetica-Bold 10
% etc.\ ..
% end
```

As you can see, this is nothing more than writing formatting parameters without the leading double percent characters.

To format a piece of music using the format contained in the file `example.fmt`, that you saved in the same folder as the source file, use the option `-F` of `abcm2ps` in the command line:

```
abcm2ps -O= -c -F example tune.abc
```

If you keep your format files in a folder, i.e. `c:\music\format`, you will also have to specify the `-D` parameter followed by the folder name:

```
abcm2ps -O= -c -D c:\music\format -F example tune.abc
```

If you wish, you may specify two or more format files on the same command line.

Using a format file is the best solution when you want to typeset a series of pieces that share the same style. Moreover, abcm2ps can be extended defining additional symbols, as we will see in Section 9. Format files containing libraries of symbols can thus be employed when needed.

♫ ♫ ♫ ♫ ♫ ♫ ♫

7 Numbering Measures and Pages

Usually, in a piece only the first measure of each line is numbered: this can be done with %%measurenb 0. To number all measures, use %%measurenb 1, while to put a number every ⟨*n*⟩ measures, you use %%measurenb ⟨*n*⟩. Measures are numbered starting from 1, unless the first measure is incomplete (anacrusis); in this case, the anacrusis will count as measure 0.

Page numbering is controlled with the option -N ⟨*number*⟩ from the abcm2ps command line. Possible values are:

- 0: page numbering deactivated.
- 1: page number above on the left.
- 2: page number on the right.
- 3: page number on the left for even pages, on the right for odd pages.
- 4: page number on the right for even pages, on the left for odd pages.

7.1 Measure Control

The number of measures per line may be controlled in various ways:

- the most precise is to insert the exact number of measures in each line;
- in most cases, it is fine to just let abcm2ps do the work with the -c option (see Section 2);
- when you want each staff to contain ⟨*n*⟩ measures, use the command %%barsperstaff ⟨*n*⟩ in the source or the option -B ⟨*n*⟩ in the abcm2ps command line;
- %%alignbars ⟨*int*⟩ aligns the bars of the next ⟨*int*⟩ lines of music. It only works on single-voice tunes.

If the last line contains fewer measures that do not extend to the entire width of the page, you can force the alignment using the command %%stretchlast.

It is generally recommended not to be too concerned with the number of measures per line. You will be better off concentrating on the music and letting abcm2ps do the formatting with the -c option.

If you decide to set the number of measures yourself, be careful not to write too many or too few per line! If you write too few, the score will look ugly; if you write too many, abcm2ps will rework the line at its discretion.

♫ ♫ ♫ ♫ ♫ ♫ ♫

8 Saving Space

A common problem is printing the score on the least possible number of pages. Once the page layout and the margins have been set, parameters that can reduce the space are:

- first of all, the powerful command `%%scale` ⟨*factor*⟩. By default, the score is produced with a scaling factor of 0.7. A greater value will enlarge the score, a smaller value will reduce its size.
- reduce the space between staves with `%%staffsep` and `%%sysstaffsep`, and use commands for setting the vertical spacing of title, subtitle, lyrics, etc.
- if the `-c` option is used, the `%%maxshrink` ⟨*factor*⟩ can be used to reduce the horizontal spacing between notes. Compression is maximum with ⟨*factor*⟩ = 1, minimum with ⟨*factor*⟩ = 0.
- to flatten slurs, use the `%%slurheight` command specifying values lesser than 1;
- sometimes, using `%%notespacingfactor` along with `%%maxshrink` might be effective. Normally, the spacing of notes is proportional to their length, but using `%%notespacingfactor 1` all notes are equally spaced.
- in a file containing several tunes, use `%%topspace 0`.

Please remember that not everybody has an eagle-like sharp sight: printing a score at too small a scale will make life hard for the musicians! Further, bear in mind that inkjet printers cannot print beyond the the page lower margin of 2 cm.

♫ ♫ ♫ ♫ ♫ ♫ ♫

9 Advanced Customisation (Experts Only!)

`abcm2ps` has a very powerful feature: the possibility to modify and/or add POSTSCRIPT routines and new symbols. To do so, the user includes a series of commands that define the new symbol or routine in the source, using POSTSCRIPT routines defined in `abcm2ps` or adding new ones.

It goes without saying that only musician-programmers will be able to use this feature. Furthermore, it is necessary to study the `abcm2ps` source code and look at the POSTSCRIPT code it produces.

9.1 New POSTSCRIPT Routines

The `%%postscript` command, followed by code in the POSTSCRIPT language, adds new routines or redefines existing ones. For example, the following commands redefine the routine `dlw` so that all lines in the score will be drawn thinner:

```
%%postscript /dlw
%%postscript {0.2 setlinewidth} bdef  % default: 0.7
```

The POSTSCRIPT routines in `abcm2ps` are defined in the source file `syms.c`.

9.2 Accompaniment Chords in Italian Notation

A nice application of `%%postscript` is the redefinition of the routine that prints accompaniment chords, in order to obtain them printed using Italian notes. The following code[4] should be saved in a format file called `italian.fmt`:

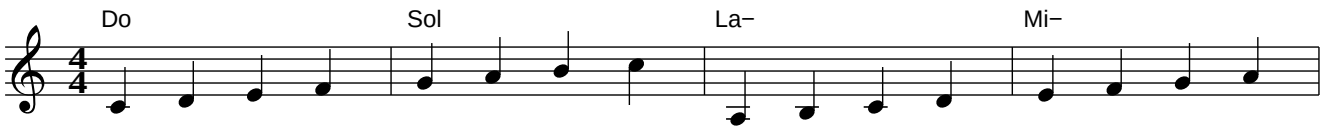
[4] kindly provided by Christopher Lane.

```
% italian.fmt
% written by Christopher Lane

% -- latin guitar chords
postscript /gcshow { % string gcshow
postscript -5 0 RM
postscript dup 0 get
postscript dup dup 65 ge exch 71 le and {
postscript 65 sub [(La) (Si) (Do) (Re) (Mi) (Fa) (Sol)] exch get show
postscript } { currentfont /Encoding get exch get glyphshow
postscript } ifelse
postscript dup length 1 sub 1 exch getinterval
postscript %
postscript dup mark exch (m) search {
postscript   (di) search { cleartomark } {
postscript     length exch pop exch (aj) anchorsearch { cleartomark } {
postscript       pop /tempstr 4 2 roll cleartomark
postscript       def tempstr exch (-) putinterval tempstr
postscript     } ifelse
postscript   } ifelse
postscript } {
postscript cleartomark } ifelse
postscript % for abcm2ps 5.4.* change this next line to just "show }!"
postscript /gchshow load cshow }!
% End of file italian.fmt
```

This is how the usual scale will be printed if we add `-F italian` to the command line:

```
X: 1
L: 1/4
K: C
"C"CDEF|"G"GABc|"A-"A,B,CD|"E-"EFGA|
```



9.3 Defining New Symbols

The `%%deco` command adds new expression symbols, using POSTSCRIPT routines defined by `abcm2ps` or possibly new ones written by the user. The syntax is the following:

`%%deco` ⟨*name*⟩ ⟨*type*⟩ ⟨*ps*⟩ ⟨*h*⟩ ⟨*wl*⟩ ⟨*wr*⟩ ⟨*string*⟩

where:

- ⟨*name*⟩ is the name of the new symbol, without the exclamation marks;
- ⟨*type*⟩ is an integer that specifies the symbol type. Values from 0 to 2 indicate a symbol near the note and within the staff, from 3 to 5 near the note but outside of the staff, and 6 and 7 are expressions linked to the staff. To give an idea of symbol positioning, here is a listing:

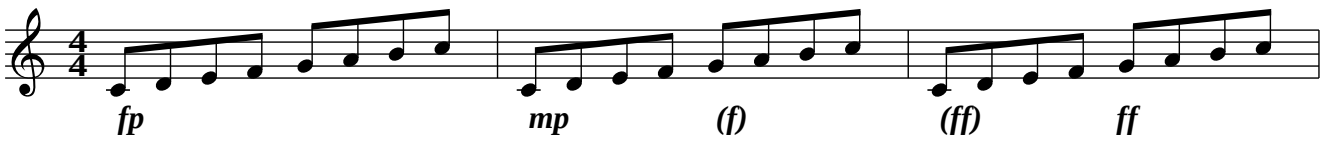
  - 0: like `!tenuto!` or the staccato dot;
  - 1: like `!slide!`;
  - 2: like `!arpeggio!`;
  - 3, 4: generic expressions;
  - 5: like `!trill(!` or `!trill)!`;
  - 6: generic;
  - 7: like long dynamics symbols.

- ⟨*ps*⟩ is the name of the POSTSCRIPT routine that draws the symbol. This may be a routine defined by the user or one provided by `abcm2ps`;
- ⟨*h*⟩ expression height in points;
- ⟨*wl*⟩ and ⟨*wr*⟩ are not used;
- finally, ⟨*string*⟩ is an optional text string.

Let us have a look at an example taken from the file `deco.abc` supplied with `abcm2ps`. We will add a few new symbols for dynamics using the predefined `pf` routines:

```
%%deco fp    6 pf   20 0 0 fp
%%deco mp    6 pf   20 0 0 mp
%%deco (f)   6 pf   20 0 0 (f)
%%deco (ff)  6 pf   20 0 0 (ff)
X: 1
T: New dynamics symbols
K: C
!fp!CDEF GABc|!mp!CDEF !(f)!GABc|!(ff)!CDEF !ff!GABc|
```

New symbols

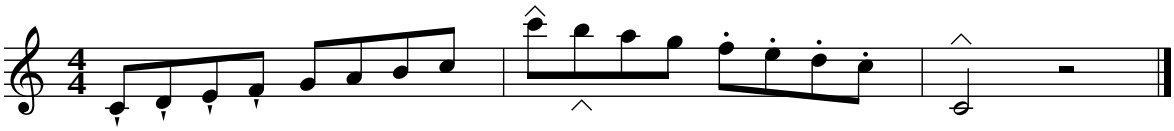


The `%%deco` line implements four new symbols: `!fp!`, `!mp!`, `!(f)!` and `!(ff)!`.

Let us see another example. The following source adds three new symbols: one note-linked and two staff-linked, one above the staff and one below. The first symbol `!tu!` is a triangle-shaped staccato symbol. `!tu!` uses the new routine `newdot`. The other symbols are `!rtoe!` and `!ltoe!` which use the routine `toe` and add a symbol similar to a `^` above and below the staff.

```
%%postscript /newdot { % usage: x y newdot
%%postscript   M 1.2 2.5 rmoveto -2.4 0 rlineto
%%postscript   1.2 -5 rlineto fill } bdef
%%deco tu 0 newdot 5 0 0
%%postscript /toe {  % usage: x y toe
```

```
%%postscript    M 5 0 rmoveto
%%postscript    -5 5 rlineto -5 -5 rlineto currentpoint stroke
%%postscript } bdef
%%deco rtoe 6 toe 5 0 0
%%deco ltoe 3 toe 5 0 0
X: 1
K: C
!tu!C!tu!D!tu!E!tu!F GABc|!ltoe!c'!rtoe!bag .f.e.d.c|!ltoe!C4 z4|]
```



9.4 Adding Fonts

Standard Ghostscript fonts are usually enough for most users. However, if you wish to add a special touch to your scores you can add new fonts. Recent versions of Ghostscript support both POSTSCRIPT and TrueType fonts. For more details, please see Appendix E.3.

An excellent site boasting a wide collection of free and high-quality POSTSCRIPT fonts is http://www.moorstation.org/typoasis/typoasis1.htm. Under the 'Designers' section, select Dieter Steffmann's page.

Let us see how to add a new font called Haenel Fraktur, downloaded as `FetteHaenel.zip`. Obviously, the procedure will be very similar with other fonts.

Unzip the archive and copy `FHaenelf.pfb` to the directory containing the Ghostscript fonts. (Other acceptable file types are those ending in `.gsf` and `.pfa`.) Supposing that you installed Ghostscript and its fonts in default locations, this directory is `C:\gs\gs8.00\font` on Windows systems (change the version number if needed), while it is `/usr/share/fonts/default/ghostscript/`[5] on Linux and other Unix variants.

Now edit Ghostscript's font list. On Windows, this file is `C:\gs\gs8.00\lib\Fontmap.GS`, on Linux it is `/usr/share/ghostscript/6.52/lib/Fontmap.GS`. Move to the bottom of the file and add this line:

```
/Haenel-Fraktur      (FHaenelf.pfb)    ;
```

which defines a new font called `Haenel-Fraktur`. If you wish, you can also define an *alias*, i.e. an alternate name for the same font:

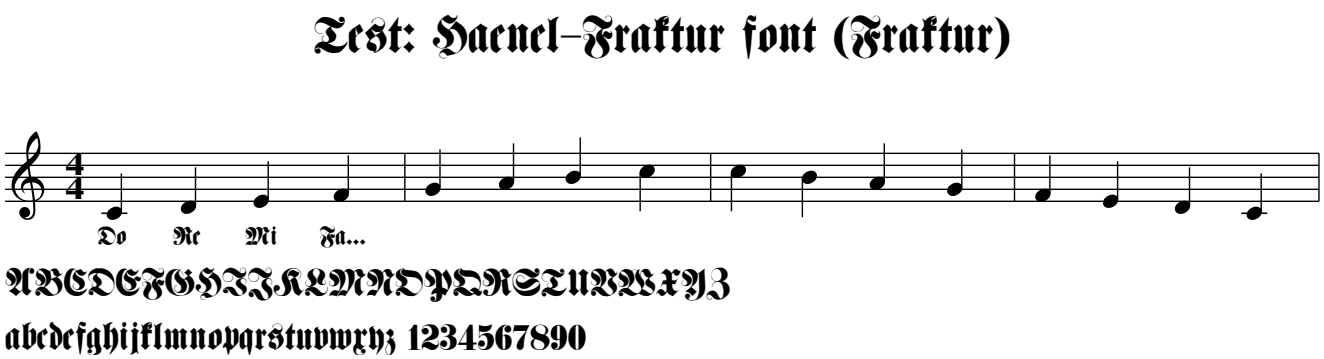
```
/Fraktur             /Haenel-Fraktur   ;
```

We are now ready to use the new font in ABC PLUS files. First of all, declare it using the `%%font` command followed by the font name. This is an example:

```
%%font Haenel-Fraktur
%%titlefont Haenel-Fraktur 24
%%textfont Haenel-Fraktur 18
%%composerfont Fraktur 16 % alias
%%vocalfont Fraktur 12
```

[5] Ubuntu variants use another path: `/usr/share/fonts/type1/gsfonts/`

```
X: 1
T: Test: Haenel-Fraktur font (Fraktur)
L: 1/4
K: C
%
CDEF|GABc|cBAG|FEDC|
w: Do Re Mi Fa... ||||
%%text ABCDEFGHIJKLMNOPQRSTUVWXYZ
%%text abcdefghijklmnopqrstuvwxyz 1234567890
```



Bear in mind that some fonts you can find on the Internet are not complete (they may only have capital letters, or miss some characters); not all are free; and not all are of good quality.

9.5 Customising Tuplets

The `%%tuplets` command allows for fine-grained tuplets. The syntax is:

`%%tuplets` ⟨*where*⟩ ⟨*what*⟩ ⟨*value*⟩

The parameters are:

- `where` is a number that indicates where to draw the tuplet indication. `0` = automatic, `1` = never, `2` = always.
- `what` is a number that indicates what to draw. `0` = draw a bracket, `1` = draw a slur.
- `value` indicates the number to print. `0` = the value of ⟨*n*⟩ in the tuplet, `1` = no value at all, `2` = a ratio ⟨*n*⟩:⟨*t*⟩.

10 Tin Whistle Fingerings

True to the folk music origins of ABC, `abcm2ps` can print tin whistle fingerings. For those who don't know, the tin whistle is a sort of small six-holed 'recorder', widely employed in Irish, English and other traditional music.

It is cheap, easy to learn (easier than the recorder), and very fun to play. I warmly suggest that you buy one!

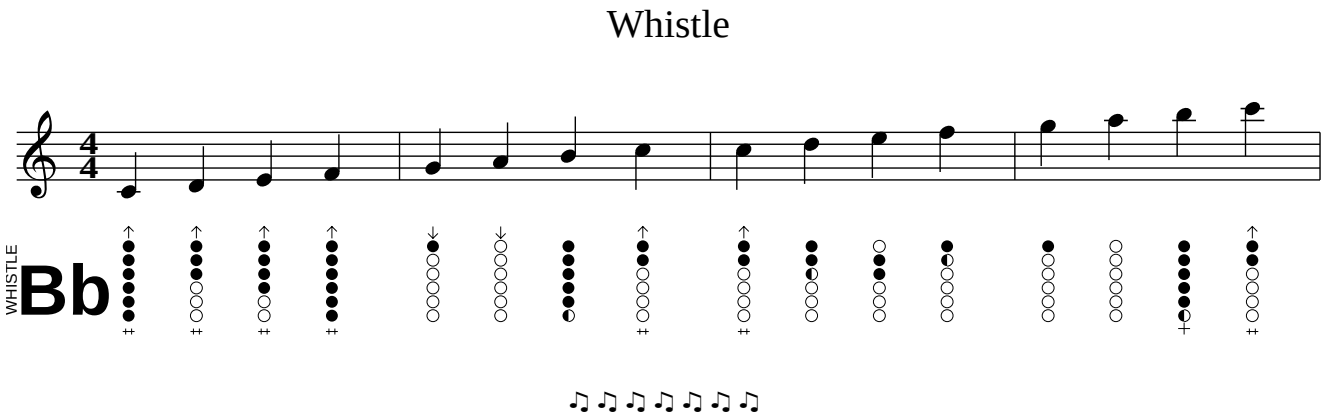
Specifying the following parameters in the `abcm2ps` command line:

`-W` ⟨*voice*⟩⟨*key*⟩

you will get the fingerings corresponding to voice ⟨*voice*⟩ (if multiple voices are not defined, `1`) for the tin whistle in ⟨*key*⟩, printed below the staff. The whistle key is denoted by the corresponding note in

upper case, and the accidental (if any) in lower case. For example, for the B♭ tin whistle you will write `Bb`. Spaces between the voice and the key are not allowed.

As usual, we'll convert a simple scale. Adding `-W 1Bb` to the command line parameters, we will get this:



Part V
Playing

11 MIDI Conversion

A MIDI file is, roughly speaking, an electronic score. It contains instructions that tell MIDI instruments (or a software MIDI player) what notes to play and how to play them. It is not as high-level as sheet music; electronic instruments and computers need to be told *exactly* what and how to play. Please note that real scores carry a bit of ambiguity. For instance, just how long a fermata is? MIDI files are not as sophisticated as a human player. Moreover,

beware: while a score in PDF format will look the same on any computer, the same does not hold true for MIDI files! In fact, the quality of a MIDI file output depends on the sound card of the computer and the player software used to listen to it.

Having a MIDI version of your ABC PLUS music is convenient, because you get an immediate feedback of what you wrote. To this end, the free `abc2midi` program is helpful. Typing a command line will convert the source to a MIDI file, which can be played with any multimedia program.

`abc2midi` creates as many MIDI files as many tunes are in `file.abc`, adding the index number of the `X:` field to each file name: `file1.mid`, `file2.mid`, ... `abc2midi` is integrated in JEDABC.

`abc2midi` is only one of the programs that are part of the `abcMIDI` package:

- `abc2abc`: verification, formatting and transposition of ABC PLUS source files;
- `midi2abc`: conversion of MIDI files to ABC PLUS;
- `yaps`: a command-line formatter analogous to `abcm2ps`, but less powerful.

`abc2midi` uses meta-comments for its low-level details. To be more precise, only the meta-comment `%%MIDI`, followed by different parameters, is actually used.

Some Linux users find it difficult to play MIDI files, because of hardware configuration problems of many sound cards. The MIDI player of choice in this situation is Timidity++ (`http://timidity.sourceforge.net/`). However, make sure your Linux distribution does not ship with a crippled version of this program.

11.1 `%%MIDI` Commands

Just as `abcm2ps` provides commands for changing page layout details, `abc2midi` provides several commands for audio effects.

Commands can be written in two ways. One is the usual meta-comment syntax: a single line containing `%%MIDI` ⟨*command*⟩ ⟨*parameters*⟩ is entered. The second way is an extension to the `I` field: `[I:MIDI =` ⟨*command*⟩ ⟨*parameters*⟩`]`. Note the `=`.

To clarify, the two following sources are equivalent:

```
X: 1
T: MIDI commands as meta-comments
```

```
L: 1/4
K: C
%%MIDI program 1
CDEF|
%%MIDI program 109
GABc|
```

```
X: 1
T: MIDI commands as inline I: fields
L: 1/4
K: C
[I:MIDI = program 1] CDEF|[I:MIDI = program 109] GABc|
```

The second method is useful to make the source more readable.

11.2 Voices and Instruments

Let us consider the Ave Verum we examined in Section 4.2. Converting it with `abc2midi`, we obtain a MIDI file in which the music output is played by the MIDI instrument 1: acoustic piano. In many cases, we don't need anything else: to study a part before a concert, the MIDI is just fine. But `abc2midi` can do much more.

One of the most important `abc2midi` commands is `%%MIDI program`, which associates a voice with a particular instrument. Let us add these commands to associate each voice with the right instrument in the Ave Verum:

```
X: 1
T: Ave Verum
C: W. A. Mozart
M: 4/4
L: 1/4
Q: "Adagio"
%%staves [(S A) (T B)] {(MD1 MD2) (MS1 MS2)}
V: S clef=treble name="Soprano" sname="S"
V: A clef=treble name="Alto" sname="A"
V: T clef=bass name="Tenor" sname="T"
V: B clef=bass name="Bass" sname="B"
V: MD1 clef=treble name="Organ"
V: MD2 clef=treble
V: MS1 clef=bass
V: MS2 clef=bass
K: D
%
%%MIDI program 1 53 % Choir Oohs
%%MIDI program 2 53
%%MIDI program 3 53
%%MIDI program 4 53
```

```
%%MIDI program 5 19 % Church Organ
%%MIDI program 6 19
%%MIDI program 7 19
%%MIDI program 8 19
... body of transcription ...
```

The eight voices `S A T B MD1 MD2 MS1 MS2` are automatically assigned by `abc2midi` the numbers 1 to 8. Then the `%%MIDI program` commands follow that associate each voice with an appropriate MIDI instrument, MIDI 54 ('Choir Oohs') or 20 ('Church Organ').

The numbers used to identify each instrument are listed in Appendix H. Note that the numbers in the list range from 1 to 128, whereas `abc2midi` numbers them 0 to 127. With `abc2midi` you will have to subtract 1 and use the numbers from 0 to 127. If no MIDI program is specified, voices will be assigned by default the General MIDI instrument 1.

If you listen to the MIDI file, you will notice that something is wrong: we'll find out why in the next section.

11.3 The Bass Clef

The most significant difference between `abcm2ps` and `abcMIDI` is the way different clefs are dealt with.

We have seen in Section 3.1 that `abcm2ps` will typeset music in bass clef with the simple indication `K:bass`; the notes are printed two lines lower on the staff. `abcMIDI`'s approach is completely different: the note pitch always remains the same, regardless of the clef. `c` is *only* interpreted as the third space note in treble clef; `C,` is the note on the second space in bass clef, etc.

This means that if a piece is written in bass clef using notes without commas, it will be played *two octaves higher* when it is converted to MIDI. Fortunately, there is a simple the solution for this problem: add a `octave=-2` definition to the `K:` or `V:` field. For example, the following short passage will both print and play correctly:

```
X: 1
L: 1/4
K: C bass octave=-2
cdef|gabc'|
```

When you write music with voices in bass clef and you don't use commas , you will have to remember to add the `octave=-2` in the related `V:` field. When using commas, this is not needed.

11.4 Accompaniment Chords

The accompaniment chords described in Section 3.6 are used by `abc2midi` to generate an accompaniment to the main melody. The types of chords currently recognised are:

```
m 7 m7 maj7 M7 6 m6 aug + aug7 dim dim7 9
m9 maj9 M9 11 dim9 sus sus9 7sus4 7sus9 5
```

Additional chords can be defined with `%%MIDI chordname`, explained in Section 11.7.

Accompaniment chords are rendered by `abc2midi` as a *sequence of* `fbcz` for each measure. `f` stands for the fundamental or root of the chord, `b` for the fundamental and the chord played together, `c` for

the chord itself, and `z` for a rest. A `fbcz` sequence is designed to match a time signature: for example, in 4/4 time the accompaniment is `fzczfzcz`: fundamental, rest, chord, rest, fundamental, rest, chord, rest.

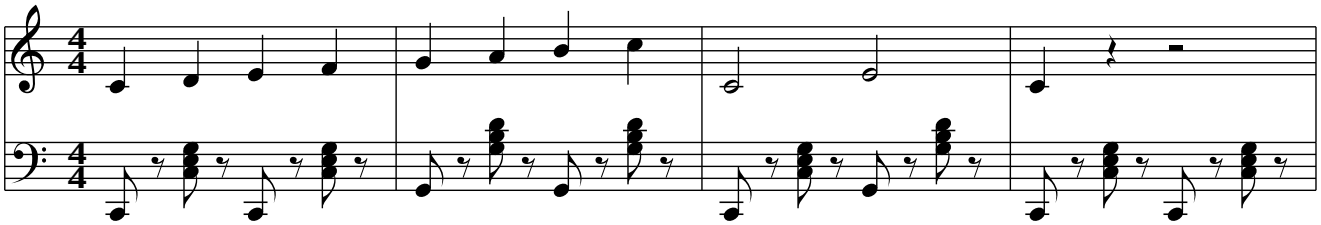
`abc2midi` associates specific `fbcz` sequences to the more common time signatures: `fzczcz` for 3/4 time, `fzcfzc` for 6/8, `fzcfzcfzc` for 9/8, and `fzcfzcfzcfzc` for 12/8.

Beware: the `fbcz` sequence *does not correspond to the beats in a measure*, and the length of the elements *does not depend on the value of the `L:` field.* Sequences are adapted to match one measure; thus, `fcz`, `d2c2z2` and `f4c4z4` have equivalent meaning.

In practice, the following piece:

```
X: 1
M: 4/4
L: 1/4
K: C
%
"C"CDEF|"G"GABc|"C"C2"G"E2|"C"Czz2|
```

will sound as if it were written like this:



If you don't hear accompaniment chords, your tune might have a time signature for which a predefined `fcz` sequence does not exist: for example, 3/8, 5/4 or 7/8. The sequence can be easily added, though. For example, a sequence for 7/8 is `fzbczcz`.

The `fcz` sequence can be modified when desired with the `%%MIDI gchord` command. This is the same tune with a simpler accompaniment:

```
X: 1
M: 4/4
L: 1/4
K: C
%%MIDI gchord c4c4
%
"C"CDEF|"G"GABc|"C"C2"G"E2|"C"Czz2|
```

Here we changed the sequence to `c4c4` to obtain two chords in every measure.

To modify the instrument associated with the chord, the `%%MIDI chordprog` command is used; for the fundamental, `%%MIDI bassprog`. To specify the volume, use `%%MIDI chordvol` for the chord and `%%MIDI bassvol` for the fundamental (from 0 to 127). Finally, to disable temporarily accompaniment chords use `%%MIDI gchordoff`, and `%%MIDI gchordon` to turn chords back on.

Note that chords will continue to be played even when the melody stops. The following tune has no melody, but only accompaniment chords:

```
X: 1
T: La Folia
M: 3/4
L: 1/4
Q: 80
K: Dm
%%MIDI gchord ccz
%%MIDI chordprog 24 % guitar
"Dm"z3|"A"z3|"Dm"z3|"C"z3|"F"z3|"C"z3|"Dm"z3|\
%%MIDI gchord c3
"A"z3|
%%MIDI gchord czc
"Dm"z3|"A"z3|"Dm"z3|"C"z3|"F"z3|"C"z3|"A"z3|\
%%MIDI gchord c3
"Dm"z3|]
```

Listen to the accompanying MIDI file folia.mid.

Let us now look at a piece that has 4/4 time but a very different rhythm: 'The Girl from Ipanema', a famous Brazilian song written by Antônio Carlos Jobim:

```
X: 1
T: Garota De Ipanema
T: (The Girl From Ipanema)
C: Antonio Carlos Jobim
M: 4/4
L: 1/8
K: F
P:A
"Fmaj7" !p!G2 GE E2 ED|G2 GE EE DG-|"G7"G2 GE EE DG-|
G2 GE EE DF-|"Gm7"F2 FD DD CE-|"Gb7"E2 EC CC B,C-|
[1"Fmaj7"C8|"Gb7"z8 :|[2"Fmaj7"C8 |z8||
P:B
"Gb7"F8-|(3F2_G2F2 (3:2:3_E2F2E2|"B7"_D3 _E-E4-|_E6 z ^G-|
"F#m7"^G8-|(3^G2A2G2 (3^F2G2F2|"D7"E3 ^F-F4-|^F6 z A-|
"Gm7"A8-|(3A2B2A2 (3:2:3G2A2G2|"Eb7"F3 G-G4-|G4 (3z2A2B2|
"Am7"(3c2C2D2 (3E2F2G2|"D7"^G3 A3 z2|"Gm7" (3B2B,2C2
(3D2E2F2|"C7" ^F3 G3 z2 ||
P:C
"Fmaj7"G3 E EE DG-|G2 GE- EE DG-|"G7"G2 GE EE DG-|G2 GE EE DA-|
"Gm7"A2 AF FF Dc-|"Gb7" c2 cE (3E2E2D2|"Fmaj7" E8-|E2 z6|
P:D
z8|]
```

When converted to MIDI, it sounds pathetic... a bossa nova has a completely different rhythm. We need to specify another `fcz` sequence that corresponds to a bossa nova. Let us insert these lines after the `K:` field:

```
%%MIDI program 67       % Baritone Sax
```

```
%%MIDI gchord fzcffczc  % bossa nova (approximate)
%%MIDI chordvol 80
%%MIDI bassvol 80
%%MIDI chordprog 25     % Steel String Guitar
%%MIDI bassprog 25
```

and this one immediately following the `P:D` field:

```
%%MIDI gchord c2
```

Reconverting to MIDI we now have a bossa nova worth its salt: garota.mid.

11.5 Customising Beats

MIDI files usually sound artificial and expressionless, but there are several ways to improve them. The command `%%MIDI beatstring` ⟨*fmp*⟩ provides a way of specifying where the strong, medium and weak stresses fall within a bar.

`f` indicates a strong beat, `m` a medium beat, and `p` a soft beat. For example, let's consider an Irish jig, which has a 6/8 time. The corresponding `fmp` sequence would be `fppmpp`.

To fine-grain the volume of the single notes in a measure, the `%%MIDI beat` ⟨*vol1*⟩ ⟨*vol2*⟩ ⟨*vol3*⟩ ⟨*pos*⟩ command can be used. `vol1`, `vol2`, and `vol3` specify the volume of notes that fall on a strong, medium, and weak beat, while `pos` indicates the position of strong beats in the measure. `abc2midi` provides default values for all volume specifiers such as `!p!` or `!ff!`.

The following example is an Irish jig:

```
X:1
T:The Swallowtail Jig
R:Jig
M:6/8
L:1/8
Q:180
K:D
%%MIDI program 1 22
%%MIDI beat 105 90 60 3
|:E/F/|"Em"GEE BEE |GEG BAG |"D"FDD ADD    |dcd "Bm"AGF|
       "Em"GEE BEE |GEG B2c |"D"dcd "Bm"AGF|"Em"GEE E2:|
     B|"Em"Bcd e2 f|e2 f edB|Bcd e2 f      |edB "D"d2 c|
       "Em"Bcd e2 f|e2 f edB|"D"dcd "Bm"AGF|"Em"GEE E2:|
```

that converts into this MIDI file: swallowtail.mid. Removing the `%%MIDI beat` line would result in a less lively MIDI file.

11.6 Arpeggios

In addition to `fcz` sequences, you can also specify *`ghijz` sequences* that allow you to play the individual notes comprising the guitar chord. This allows you to play broken chords or arpeggios.

The new codes `ghijGHIJ` reference the individual notes, starting from the lowest note of the chord. For example, for the C major chord, `g` refers to C, `h` refers to E and `i` refers to G. Upper case letters refer to the same notes one octave lower, `z` to a rest.

The following example plays the C major chord as an arpeggio of CEGE:

```
%%MIDI gchord ghih
```

Furthermore, you can use fcz and ghij sequences together, like fcbghijGHIJz.

11.7 New accompaniment chords

The %%MIDI chordname command allows you to change the notes of an accompaniment chord, or define new chords. The syntax is:

%%MIDI chordname ⟨*chord name*⟩ ⟨*n1*⟩ ⟨*n2*⟩ ⟨*n3*⟩ ⟨*n4*⟩ ⟨*n5*⟩ ⟨*n6*⟩

where 'chord name' is a name such as those given in Section 11.4, ⟨*n1*⟩ is the chord fundamental and and the other notes (up to 6) are expressed as semitones above the fundamental.

These lines define the chords '4' and '5+':

```
%%MIDI chordname 4   0 5 7 12  % e.g. C F G c
%%MIDI chordname 5+  0 4 8 12  % e.g. C E ^G c
```

Now we can apply these new chords to any note: 'C4', 'G5+' and so forth.

11.8 Broken Rhythm

A typical rhythm of traditional Irish music is the *hornpipe*, which consists of series of dotted notes (broken rhythm):

```
X: 1
T: Broken rhythm
M: 2/4
L: 1/8
K: C
C>D E>F  | G>A B>c | c>d e>f | g>a b>c' |
c'>b a>g | f>e d>c | c>B A>G | F>E D>C  |
```

Writing a piece this way can be tedious. There is a shortcut though: adding the R:hornpipe field will instruct abc2midi to set the broken rhythm automatically. This effect will only work if the note length is set to 1/8.

Let us rewrite the scale:

```
X: 1
T: Broken rhythm
R: hornpipe
M: 2/4
L: 1/8
K: C
CD EF  | GA Bc | cd ef | ga bc' |
c'b ag | fe dc | cB AG | FE DC  |
```

This is the resulting MIDI file is: broken.mid.

11.9 Drum Patterns

In addition to accompaniment chords, we can add a percussion accompaniment to our music with `%%MIDI drum` command, which has a syntax somewhat similar to `%%MIDI gchord`.

The `%%MIDI drum` command is followed by a sequence of `dz`, where `d` represents a percussion beat and `z`, predictably, a rest. After the sequence, you write the codes for the desired percussion instruments (see Appendix H.2) and their volumes expressed as numbers from 0 to 127.

A drum accompaniment is turned on with `%%MIDI drumon` and turned off with `%%MIDI drumoff`. The following tune has a bass drum and hi-hat accompaniment:

```
X: 1
M: 4/4
L: 1/4
K: C
%            sequence  instrument    volume
%%MIDI drum  dddd      36 46 36 46   80 100 80 100
% bass drum 1, open hi-hat
%%MIDI drumon
CDEF|GABc|\
%%MIDI drumoff
cdef|\
%%MIDI drumon
gabc'|
```

You cannot specify a fractional length: to indicate a rhythm such as `(3ddd d/d/d/d/` you need to use the sequence `d4d4d4d3d3d3d3`.

Here is a fun and more complex example (riff.mid):

```
X: 1
M: 4/4
T: Riff
%%MIDI program 1 25 % Steel String Guitar
Q: 1/4=160
K: C
%%MIDI drum dzddd2dz  35 39 39 35 39  127 80 80 127 80
% Bass Drum 1 + Electric Snare
%%MIDI gchord cccccccc
%%MIDI drumon
"C"CC EE GG AA|_BB AA GG EE|
CC EE GG AA|_BB AA GG EE|
"F"FF AA cc dd|"F7"_ee dd cc AA|
"C"CC EE GG AA|_BB AA GG EE|
%%MIDI gchordoff % no chords
"G"GG BB dd Bd|"F7"FF AA cc Ac|
%%MIDI gchordon  % turn chords back on
"C"CC EE GG AA|_BB AA GG EE|
%%MIDI gchord c8
%%MIDI program 1 60 % Brass Section
```

```
%%MIDI drumoff
!fermata!"C"[C8E8G8c8]|
```

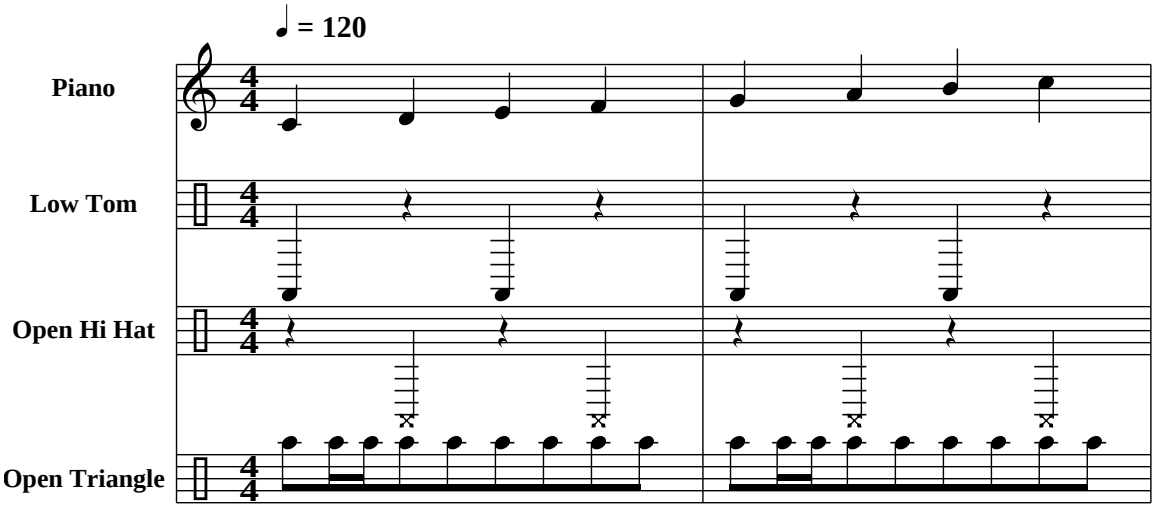
11.10 Percussion Instruments

MIDI files have different *channels*, which can hold several *tracks*. Normally, you don't bother with these details; all you need to know is that, by default, different voices correspond to different tracks on the same channel.

In normal MIDI channels, all notes belong to the associated instrument. However, the special MIDI channel 10 has a peculiar feature: each note is associated with a different percussion instrument. Considering the instrument list in Appendix H.2, the following tune will have a melody played by piano, and low-tom, hi-hat and triangle accompaniment (accomp.mid):

```
X: 1
L: 1/4
Q: 1/4 = 120
V: 1 name="Piano"
V: 2 clef=perc name="Low Tom"
V: 3 clef=perc name="Open Hi Hat"
V: 4 clef=perc name="Open Triangle"
K: C
%
[V: 1] CDEF          |GABc                 | % Piano
[V: 2] \
%%MIDI channel 10
A,,zA,,z             |A,,zA,,z             | % Low Tom
[V: 3] \
%%MIDI channel 10
z^A,,z^A,,           |z^A,,z^A,,           | % Open Hi Hat
[V: 4] \
%%MIDI channel 10
a/a//a//a/a/a/a/a/a/|a/a//a//a/a/a/a/a/a/| % Open Triangle
```

Note that the `%%MIDI channel 10` command must be written *after* the `V:` field that starts a new voice. Unfortunately, if we typeset this piece we will not get what we would expect:



In fact, `abcm2ps` has limited support for percussions. Future releases will provide full support for percussions instruments.

11.11 Advanced Use of `P:`

Consider the following piece:

```
X: 1
T: Repeats
T: printed only
L: 1/4
K: C
|:: CDEF|GABc ::|C2c2|Czz2|
```

The first two measures are to be played three times (indicated with `|:: ::|`); then, the two following measures follow.

When we listen to the MIDI file, we realise that something is wrong. In fact, `abc2midi` doesn't correctly interpret the double colons, and plays the first two measures only twice. This is but one example where correct notation in the printed score is not interpreted correctly in MIDI output.

In such cases, `abc2midi` can be instructed by using `P:` (Section 3.5). Remember that `P:` is not only used to indicate parts, but it can also indicate the order in which parts are to be played.

Rewriting the previous piece as:

```
X: 1
T: Repeats
T: printed and played
L: 1/4
P: A3.B % <- play part A 3 times, then B
K: C
|:: [P:A]CDEF|GABc ::|[P:B]C2c2|Czz2|
```

The only snag is that the part indications will be printed in the score. If you don't want them to appear, use the `%%printparts 0` command.

11.12 Drone

Bagpipe, medieval and other kinds of music are often accompanied by one or more drone notes. `abc2midi` supports drones using these commands:

- `%%MIDI drone` ⟨*instrument*⟩ ⟨*pitch1*⟩ ⟨*pitch2*⟩ ⟨*velocity1*⟩ ⟨*velocity2*⟩ specifies the drone characteristics;
- `%%MIDI droneon` starts the drone accompaniment;
- `%%MIDI droneff` stops the drone.

The parameters of the `%%MIDI drone` command are ⟨*instrument*⟩, that specifies the MIDI instrument for the drone; ⟨*pitch1*⟩ and ⟨*pitch2*⟩ are the MIDI pitches of the drone notes; ⟨*velocity1*⟩ and ⟨*velocity2*⟩ are the MIDI 'velocities', that is the volume, of the drone notes.

The pitches are not specified as ABC *notes*, but as standard MIDI pitches. In short, these are numeric codes (1–127) that correspond to notes, as shown in Table 8. To obtain notes higher or lower than the octave shown in the table, simply add or subtract 12 to the note.

The default values of `%%MIDI drone` are 71 (bassoon), 45 (A,,), 33 (A,,,), 80 and 80.

Note	A,,	^A,,	B,,	C,	^C,	D,	^D,	E,	F,	^F,	G	^G,	A,
MIDI pitch	45	46	47	48	49	50	51	52	53	54	55	56	57

Table 8: Standard notes and corresponding MIDI pitches.

Let's put it into practice. This is Amazing Grace, written in G major with bagpipe drone accompaniment:

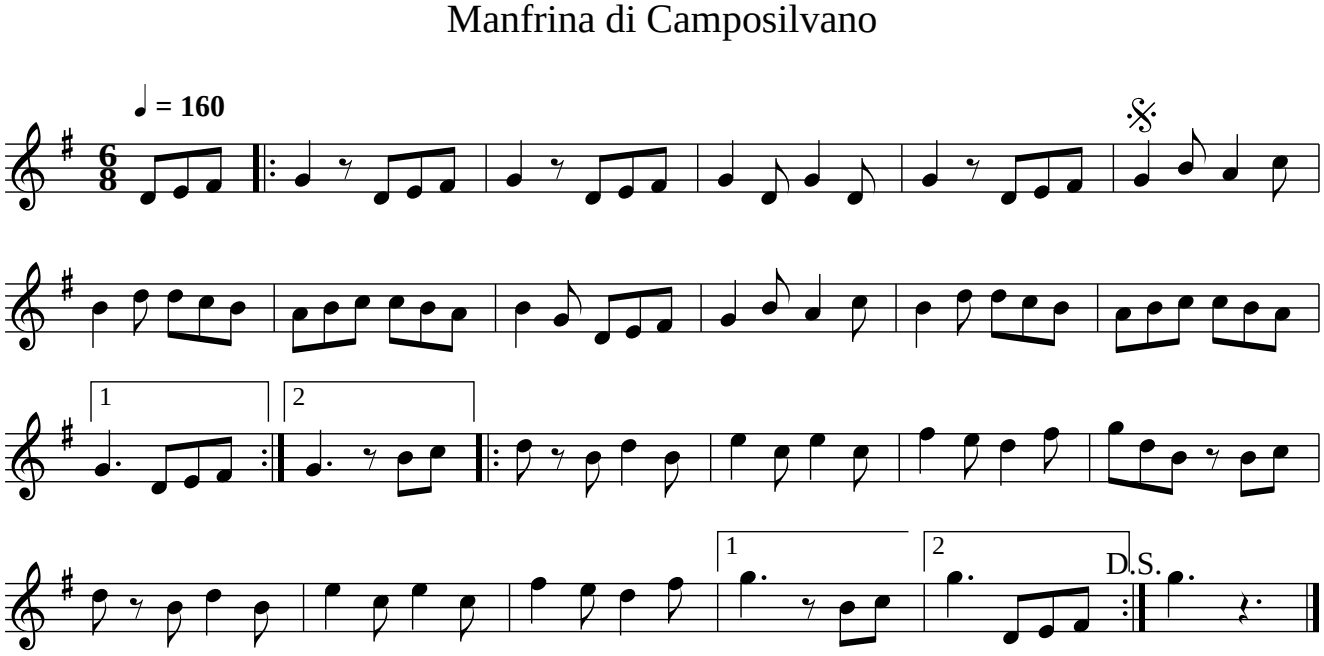
```
X:1
T:Amazing Grace
M:3/4
L:1/8
Q:40
K:G
%%MIDI gracedivider 16
%%MIDI program 109
%%MIDI drone 109 43 31 70 70
%%MIDI droneon
|z3 z2D|{/A}G2 B/G/ B2 A|{/A}G2 {F}E D2 D|
{/A}G2 {/C}B/G/ {/C}B2 A/B/|d3-d2 B|
d2 B/G/ B2 A|G2 E {/E}D2 D|
{/A}G2 B/G/ B2 A|{/A}G3-G2 |]
%%MIDI droneoff
```

The resulting MIDI file is amazinggrace.mid.

11.13 Beware of Repeats

I began Part V stating that MIDI files are not as smart as a printed score. The following tune is a case in point:

```
X: 1
T: Manfrina di Camposilvano
M: 6/8
L: 1/8
Q: 1/4 = 160
K: G
DEF|: G2z DEF|G2z DEF|G2DG2D|G2z DEF|!segno!G2BA2c|B2d dcB|
ABc cBA|B2G DEF|G2BA2c|B2d dcB |ABc cBA|1 G3 DEF:|2 G3z Bc|
|: dz B d2B|e2ce2c|f2e d2 f|gdBz Bc|d zB d2B|e2c e2c|
f2e d2f|1 g3zBc|2 g3 DEF !D.S.!:|g3 z3|]
```



Apparently, no problem. But if you convert it to MIDI, the output will be wrong. The problem is due to `abc2midi` being unable to figure out what `!segno!` and `!D.S.!` mean, although this is very clear to a human player. Moreover, `!segno!` is set within a repeated section, but not at the beginning.

What you have to do here is to count the logical parts that make up the tune, then specify them as `P:` fields. Try and figure it out yourself. Alternatively, you should rewrite the source in a clearer way.

The working source is:

```
%%printparts 0
X: 1
T: Manfrina di Camposilvano
M: 6/8
L: 1/8
Q: 1/4 = 160
P: ABCD.BCE.FGFH.CDCE.FGFI
K: G
[P:A] DEF|:[P:B] G2z DEF|G2z DEF|G2DG2D|G2z DEF|
[P:C] !segno!G2BA2c|B2d dcB|ABc cBA|B2G DEF|G2BA2c|
```

```
B2d dcB |ABc cBA|1 [P:D] G3 DEF:|2 [P:E] G3z Bc|
|:[P:F] dz B d2B|e2ce2c|f2e d2 f|gdBz Bc|d zB d2B|
e2c e2c|f2e d2f|1 [P:G] g3zBc|2 [P:H] g3 DEF !D.S.!:|
[P:I] g3 z3|]
```

Please listen to manfrina.mid. If you are not convinced, try and see what happens converting the source without `P:`'s.

If you think that this `P:` thing is too difficult, don't be put off: complex examples like the above are actually difficult to find!

11.14 `midi2abc`

This program converts a MIDI file to the corresponding ABC PLUS source. It does the job with good approximation, but the source should be edited to add voice layout and formatting parameters.

The command line is simply:

```
midi2abc file.mid -o file.abc
```

`midi2abc` has many command-line parameters, which we will not examine for now. By default, the resulting ABC PLUS file will contain only one measure per line.

The best way to get an ABC PLUS source from a MIDI file is using `runabc.tcl` as shown in Figure 8. Select extras/midi2abc, then fill the appropriate fields. The `voice interleave` radio button will write different voices interleaved, instead of one after another.

Press the button `midi2abc` to create the ABC PLUS source.

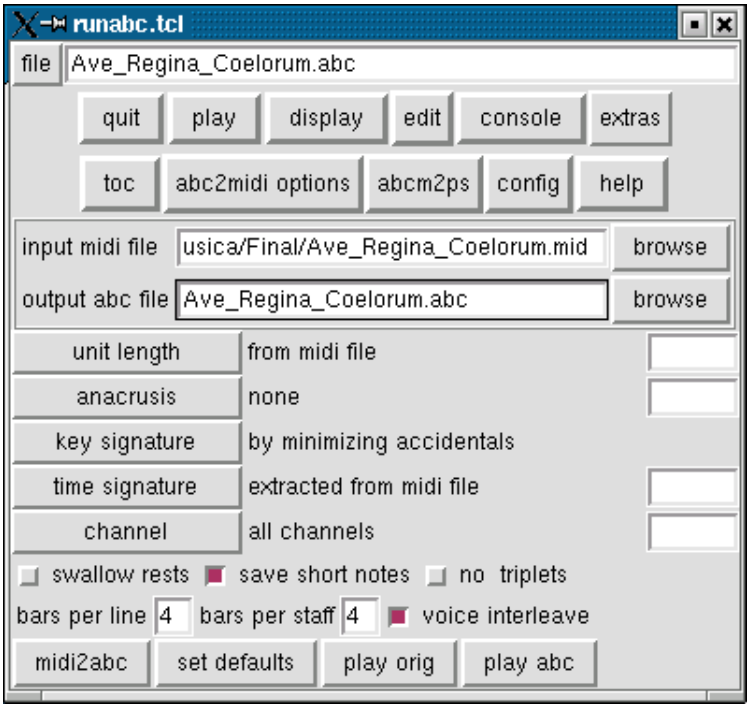


Figure 8: Converting a MIDI file to ABC PLUS with `runabc.tcl`.

Don't expect to obtain perfect sources all the time! In fact, while sheet music (be it written in ABC PLUS or in any other format) can always be translated into MIDI, the opposite does not always hold

true. Remember what I explained in Section 11. For example, you will see that trills are translated as sequences of short notes; repeats will just duplicate measures; and many other problems. Sometimes, note length will look crazy.

Bearing in mind that some of these limitations cannot be avoided, further development is being planned to improve the output of `midi2abc`.

♫ ♫ ♫ ♫ ♫ ♫ ♫

12 Differences and Incompatibilities

Unfortunately, `abcm2ps` and `abc2midi` are not completely compatible with each other, because the first program accepts a more extended syntax than the second. Moreover, it should be pointed out that some indications only make sense in a printed score. Consequently, when writing music in ABC PLUS bear in mind that:

- tremolo decorations don't produce audible effect;
- if a system change occurs in the middle of a piece, `abc2midi` gets lost and generates an incorrect MIDI file;
- in `U:` fields, `abc2midi` only accepts uppercase `H` ... `Z`;
- ... there may be others.

Because of these small incompatibilities, we have the problem of writing music that can be converted by both `abcm2ps` and `abc2midi`. In theory, we should write two source files, one for `abcm2ps` and another for `abc2midi`: this is obviously unacceptable. An alternative is to use the `abcpp` preprocessor, which is explained in the next section.

♫ ♫ ♫ ♫ ♫ ♫ ♫

Part VI
Converting

13 The abcpp Preprocessor

A *preprocessor* is a program that modifies a text file, according to commands contained in the file. `abcpp` is a preprocessor expressly designed for ABC PLUS files. It allows to

- exclude or include parts of a piece according to specified conditions;
- define *macros*, i.e. symbols and sequences of customised commands;
- rename commands, symbols, and notes;
- include parts of other files.

Needless to say, `abcpp` is a command-line program. You run it specifying the names of input and output files, and possibly defining *symbols*.

13.1 Basic Usage

Let us look at an example. We will write a portable ABC PLUS file, which can be read correctly by `abcm2ps` and `abc2midi`. Save this source as `test.abp`:

```
X: 1
T: Test with abcpp
#ifdef ABCMIDI
T: (version for abc2midi)
Q: 1/4 = 120
#else
T: (version for abcm2ps)
Q: "Allegro" 1/4 = 120
#endif
K: C
cdef gabc'|c'bag fedc|
```

Note the lines that start in `#`: these are *directives* (commands) to the preprocessor.

The first directive means: "if the symbol ABCMIDI is defined, then..." If the condition is true, the source continues with the next two lines; otherwise, with the lines that follow the `#else` directive. The `#endif` directive terminates the condition.

To convert the source to make it acceptable to `abc2midi`, we'll run `abcpp` with this command line:

```
abcpp -ABCMIDI test.abp test-midi.abc
```

This way we define the `ABCMIDI` symbol, and a new ABC PLUS file will be created:

```
X: 1
T: Test with abcpp
T: (version for abc2midi)
Q: 1/4 = 120
K: C
cdef gabc'|c'bag fedc|
```

If we run abcpp without defining any symbols, we'll get the right source for abcm2ps:

```
abcpp test.abp test-ps.abc

X: 1
T: Test with abcpp
T: (version for abcm2ps)
Q: "Allegro" 1/4 = 120
K: C
cdef gabc'|c'bag fedc|
```

Let us consider another example. Some ABC applications don't support invisible rests. To make it possible to use them portably, we have to insert these lines in the source:

```
#ifdef OLD
#define !x! z
#else
#define !x! x
#endif
```

In plain English: "if the OLD symbol is defined, then turn the !x! decoration into z; otherwise, !x! will become x". As you write the tune, you will use !x! to denote invisible rests. When you convert the source for abcm2ps or other programs, the !x! symbol will be turned into x or z according to the presence of the symbol OLD.

13.2 Advanced Usage

We have seen in Figure 4 an example of ABC PLUS file in which the system changes. Unfortunately, abc2midi doesn't correctly handle this source, because the number of voices is variable.

Let us see how we can use abcpp to obtain a version compatible with abc2midi. The idea is to write *all the voices*, even those that contain only rests, then provide specific instructions for abcm2ps and abc2midi. We will obtain a source where only voice 3 of parts A and C is printed, while the second will contain all voices.

```
X: 1
T: Riu, riu, chiu, la guarda ribera!C: "Villancico" (Spain, XVIth century)
M: C|
L: 1/2
Q: 1/2 = 240
#ifdef MIDI
P: ABCB
#endif
%%staves 3
V: 3 clef=treble-8 name="Tenor\nBass"
K: Am
% ONLY THE MEN
#ifdef MIDI
P: A
[V: 1] [M:none] z4              |z4z4    |z6    |
[V: 2] [M:none] z4              |z4z4    |z6    |
[V: 4] [K: Am octave=-1]\
[M:none] aaga            |f2ed2efg|a2a2z2|
#endif
[V: 3] [M:none] AAGA            |F2ED2EFG|A2A2z2|
```

```
w: Ri-u, ri-u, chi-u, la guar-da ri-be-ra!
%
#ifdef MIDI
[V: 1] z4               |z4z4    |z6    |
[V: 2] z4               |z4z4    |z6    |
[V: 4] aaga             |f2eg2gef|d2d2z2|
#endif
[V: 3] AAGA             |F2EG2GEF|D2D2z2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra,
%
#ifdef MIDI
[V: 1] z4               |z4z4    |z4  |
[V: 2] z4               |z4z4    |z4  |
[V: 4] aaga             |f2eg2gef|d2d2|
#endif
[V: 3] AAGA             |F2EG2GEF|D2D2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra.
% WOMEN AND MEN
#ifndef MIDI
%%staves [1 2 3 4]
V: 1 clef=treble   name="S" sname="S"
V: 2 clef=treble   name="A" sname="A"
V: 3 clef=treble-8 name="T" sname="T"
V: 4 clef=bass     name="B" sname="B"
#else
P: B
#endif
[V: 1]AAGA|F2ED2EFG |A2A2z2|
w: Ri-u, ri-u, chi-u, la guar-da ri-be-ra!
[V: 2]FFEC|D2EF2EDD |C2C2z2|
[V: 3]cccG|A2AA2ADD |E2E2z2|
w: Ri-u, ri-u, chi-u, la guar-da ri-be-ra!
[V: 4]ffcf|d2Ad2c_BB|A2A2z2|
%
[V: 1] z4      |AAGA|F2EF2FEE|D2D2z2|
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra,
[V: 2] z2EE    |DCEC|D2CD2DCC|D2D2z2|
w: Di\'os guar-d\'o el lob', el lo-bo de nue-stra cor-de-ra,
[V: 3] ccBc    |A2BA|A2AA2AAA|A2A2z2|
w: Di\'os guar-d\'o el lo-bo, el lo-bo de nue-stra cor-de-ra,
[V: 4] aaga    |f2ef|d2Ad2dAA|d2d2z2|
%
[V: 1] z4  |AAGA|F2ED2DCC    |D2D2z2  |
w: Di\'os guar-d\'o el lo-bo de nue-stra cor-de-ra.
[V: 2] z2EE|DCEC|D2CA,2A,A,A,|A,2A,2z2|
w: Di\'os guar-d\'o el lob', el lo-bo de nue-stra cor-de-ra.
[V: 3] ccBc|A2BA|A2AF2FEE    |D2D2z2  |
w: Di\'os guar-d\'o el lo-bo, el lo-bo de nue-stra cor-de-ra.
[V: 4] aaga|f2ef|d2Ad2dAA    |d2d2z2  |
% ONLY THE MEN
#ifdef MIDI
P: C
[V: 1] z4  |z8      |z4|z4  |
[V: 2] z4  |z8      |z4|z4  |
[V: 4] aaga|f2eg2gef|d4|aaga|
#else
%%staves 3
#endif
[V: 3] AAGA|F2EG2GEF|D4|AAGA|
w: El lo-bo ra-bio-so la qui-so mor-der, mas Di\'os po-de-
%
#ifdef MIDI
[V: 1] z8      |z4|z4  |z8      |
```

```
[V: 2] z8      |z4|z4  |z8      |
[V: 4] f2feggef|d4|aaga|f2fedefg|
#endif
[V: 3] F2FEGGEF|D4|AAGA|F2FEDEFG|
w: ro-so la su-po de-fen-der; qui so-le ha-ce que no pu-die-sce pe-
%
#ifdef MIDI
[V: 1] z4|z4  |z8      |z4  |
[V: 2] z4|z4  |z8      |z4  |
[V: 4] a4|aaga|f2feggef|d2d2|
#endif
[V: 3] A4|AAGA|F2FEGGEF|D2D2|
w: car: ni˜aun o-ri-gi-nal e-sta Vir-gen no tu-vie-ra.
```

♫ ♫ ♫ ♫ ♫ ♫ ♫

14 abc2abc

it is part of the abcMIDI package. This command-line program is used to modify the ABC PLUS source in several ways. abc2abc is followed by the name of the file to modify, and then by one of these options:

-n ⟨*x*⟩ reformats the source with ⟨*x*⟩ measures per line.

-t ⟨*n*⟩ transposes the music by ⟨*n*⟩ semitones. ⟨*n*⟩ may be a negative number.

-d doubles the note lengths.

-v halves the note lengths.

-V ⟨*x*⟩ outputs only voice ⟨*x*⟩ of a polyphonic file.[6]

-X ⟨*n*⟩ for a file with several pieces, renumbers the X: field starting with ⟨*n*⟩.

As usual, here is an example. Let us modify this scale:

```
X: 1
L: 1/4
K: C
CDEF|GABc|cdef|gabc'|c'cCz|
```

starting abc2abc with this command line:

```
$ abc2abc cde.abc -n 2 -t 2
```

that is, we are reformatting the source to get two measures per line and transposing by two semitones up. This is what we obtain:

[6] The program abc2prt (Section 18) works better.

```
X:1
L:1/4
K:Eb
%
EFGA|Bcde|
efga|bc'd'e'|
e'eEz|
```

The transposing feature will be extremely useful to players of clarinet and other transposing instruments.

♫ ♫ ♫ ♫ ♫ ♫ ♫

Part VII
Other Possibilities

15 Inserting Music in Other Programs

Sheet music in POSTSCRIPT format can be easily converted to other formats suitable for word processing, web pages, etc. In practice, there are only two recommended formats: JPG and PNG. The latter is the best.

To convert POSTSCRIPT to PNG, Windows users can simply use GhostView. Select File/Convert, choose `png16` in the `Device` field, then select the pages you wish to convert.

Resolution is a very important parameter. The higher the resolution, the better the quality of the output; but the file size also grows exponentially. A resolution of 300 dots per inch is fine.

Unix users could use the low-level Ghostscript interpreter. The following script converts an input file to PNG:

```
#!/bin/sh
FILE=$(basename $1 .ps)
gs -dNOPAUSE -q -dBATCH -sPAPERSIZE=a4 \
-sDEVICE=pnggray \
-dTextAlphaBits=4 -dGraphicsAlphaBits=4 \
-r300x300 \
-sOutputFile=$FILE-%003d.png \
$1
```

A similar method is using `convert`, a command provided by the ImageMagick package (http://www.imagemagick.org/). You use it as in this example:

```
convert -density 300x300 file.ps file.png
```

The `-density` parameter specifies the resolution.

♫ ♫ ♫ ♫ ♫ ♫ ♫

16 Inserting Music in LaTeX

This guide is written in LaTeX, which you may want to use instead of a word processor. To insert ABC PLUS music in LaTeX documents, you have to decide whether your final format will be POSTSCRIPT or PDF.

In both cases, you will have to convert the score to EPS (encapsulated POSTSCRIPT). This is done either by specifying the `-E` switch in the `abcm2ps` command line, or using the command `ps2epsi`. This one is part of Ghostscript. When you are done, you will include the `graphicx` package in your document and include the music as in this example:

```
\documentclass[a4paper,12pt]{article}
\usepackage{graphicx}
\begin{document}
This is some ABC music:
```

```
\medskip
\includegraphics[width=\linewidth]{music.eps}
\end{document}
```

If you wish to use `pdflatex`, you will have to convert the score from EPS to PDF using `epstopdf`, then insert the PDF file in the LaTeX source.

16.1 Using `abc.sty`

Prof. Enrico Gregorio of University of Verona, Italy, has written a package that enables the inclusion of ABC PLUS code in LaTeX documents. The relevant archive, called `abc.zip`, can be freely obtained from CTAN mirrors. Please see Section A for details.

Once installed, the file `abc.sty` provides the following facilities:

- the `abc` environment
- the `\abcinput` command
- the `\abcwidth` parameter.

More explanations are included in the package documentation. This is a sample LaTeX document that employs `abc.sty`:

```
\documentclass[a4paper,12pt]{article}
\usepackage[generate,ps2eps]{abc}
\usepackage{mathptmx}

\begin{document}

\title{Example of ABC in \LaTeX{}}
\author{Guido Gonzato}
\date{}
\maketitle

This is a short piece.

\medskip

\begin{abc}
X:4
T:Cronin's Hornpipe
R:hornpipe
S:Keenan and Glackin
E:7
M:C|
L:1/8
K:G
BA|GABc dBde|gage dega|bage dBGB|cABG A2BA|
GABc dBde|gage dega|bage dBAB|G2G2 G2:|
fg|afd^c d2ga|bged e2ga|(3bag  (3agf gedB|(3cBA AG AcBA|
GABc dBde|~g3e dega|bage dBAB|G2G2 G2:|
\end{abc}
```

```
\medskip

Let's now include a tune we have in the current directory as
\texttt{tune.abc}:

\medskip

\abcinput{tune}

\end{document}
```

♫ ♫ ♫ ♫ ♫ ♫ ♫

17 Converting Graphics to EPS

Very often, graphic files are in JPG, GIF or PNG format. Converting such files into EPS files suitable for inclusion with the `%%EPS` command is best done with yet another command-line program, `bmeps`. It is available from http://www.ctan.org/tex-archive/support/bmeps; I suggest that Windows users download the provided static binary.

`bmeps` is used as in this example:

```
$ bmeps -c myfile.png myfile.eps
```

If you omit `-c`, the resulting EPS file will be in black and white.

♫ ♫ ♫ ♫ ♫ ♫ ♫

18 Parts Extraction

Another useful tool is `abc2prt`, which extract voices from polyphonic sources. You use `abc2prt` to create new ABC PLUS files containing the single voices.

For example, let us suppose we want to extract the tenor part (voice 3) from the Ave Verum listed in Section 5. All you have to do is run `abc2prt` this way:

```
abc2prt -3 aveverum.abc aveverum-3.abc
```

A new ABC PLUS file called `aveverum-3.abc`, containing only voice 3, will be created.

Needless to say, `abc2prt` is integrated in JEDABC.

♫ ♫ ♫ ♫ ♫ ♫ ♫

19 Limitations of `abcm2ps`

Although it is a very powerful program, `abcm2ps` currently has a few limitations to be aware of:

- no manual control over symbol positioning;
- some formatting parameters cannot change within the same tune;

- doesn't support special notations like percussions or Gregorian Chant;
- ...

This list was longer some versions ago... Most likely, the missing features will be implemented in future releases.

♫ ♫ ♫ ♫ ♫ ♫ ♫

20 Final Comments

This guide is written and copyrighted by Guido Gonzato, <guido,dot,gonzato,at,poste,dot,it>, and is released under the GNU GPL license. This means that this guide is available at no cost, and is freely distributable and modifiable. However, if you make modifications to the text you must make them publicly available. Please feel free to report bugs, suggestions, comments, and so on.

A big 'thank you!' to the author of abcm2ps, Jean-François Moine, for writing such a beautiful and useful program; to Michael Methfessel for writing the original abc2ps; to James Allwright for writing the original abcMIDI, and to Seymour Schlien for maintaining and improving it; to Chris Walshaw for creating ABC; to Norman Schmidt who helped me translating parts of this manual into English.

Thanks to my friend Maestro Sandro Pasqualetto and to Gianni Cunich for their suggestions on how to improve this guide. Last but not least, thanks to all people who contribute to ABC PLUS!

20.1 Please, make a donation...

I say again, this manual is free. That said, if the results of my work on ABC PLUS are useful to you, it would be a good thing if you made a donation. I used to ask for a little amount of money, which I don't need anymore now that the mortgage's over. So: please make a donation to a charity of your choice, and let me know. You will gain some good karma.

May I ask that you send me a postcard, too? I'm especially fond of natural landscapes. Mountain views or geology images will make my day.

This is my home address: Guido Gonzato, Via Monte Ortigara 2/a, 37126 Verona, Italy.

Thank you so much!

20.2 In Loving Memory of Annarosa Del Piero, 1930–2000

I had the privilege to be a friend of Annarosa's, without whom I would be a different person.

No rhetoric, Annarosa was unique. She profoundly loved and enjoyed art and music. She shared her love with me when I was just a kid, giving me records of opera arias as presents. She took me by train to visit Venice for the first time in my life, and she introduced me to the beauty of the mountains.

She confronted her fatal illness with courage and dignity. Till the end she listened to her favourite music, till the end she gave me beautiful records of operas as present. This guide is dedicated to her memory: a tiny leaf born from the seed she throwed when she had a six-year-old kid listen to Rigoletto, so many years ago. Ciao, Annarosa.

♫ ♫ ♫ ♫ ♫ ♫ ♫

Part VIII
Appendix

A Web Links

The ABC PLUS home page:

http://abcplus.sourceforge.net

The original ABC page:

http://www.walshaw.plus.com/abc/

The ABC Project page:

http://abc.sourceforge.net/

A very nice web-based ABC PLUS converter. It's a convenient interface to `abcm2ps` and `abc2midi`:

http://www.folkinfo.org/songs/abcconvert.php

The Philip's Music Writer home page is about a powerful and easy-to-use textual music notation format, which could be a valuable alternative to ABC PLUS:

http://www.quercite.com/pmw.html

Lilypond is another high-quality music notation format, very powerful but rather complex:

http://lilypond.org/web/

The `abc` package for LaTeX:

http://www.ctan.org/tex-archive/macros/latex/contrib/abc/

Finally, the MusiXTeX home page. This is a TeX-based, very complex notation that spurred the creation of two spinoffs, PMX and M-Tx:

http://icking-music-archive.org/

B ABC PLUS Fields

Field	Where	Notes and Example
A:	header	Area. `A:Liverpool`
B:	header	Book. `B:Groovy Songs`
C:	header	Composer. `C:The Beatles`
D:	header	Discography. `D:The Beatles Complete Collection`
d:	body	Decorations. `d:!pp! * * !mf! * !ff!`
F:	header	File name. `F:http://www.beatles.org/help.abc`
G:	header	Group. `G:guitar`
H:	header	History. `H:This song was written...`
I:	header	Information. `I:lowered by a semitone`
K:	last in header	Key. `K:C`
L:	header, body	Note length. `L:1/4`
M:	header, body	Metre. `M:3/4`
N:	header	Notes. `N:See also...`
O:	header	Origin. `O:English`
P:	header, body	Part. `P:Start`
Q:	header, body	Tempo. `Q:1/2=120`
R:	header	Rhythm. `R:Reel`
S:	header	Source. `S:Collected in Liverpool`
T:	second in header	Title. `T:Help!`
U:	header	User defined. `U:T=!trill!`
V:	header, body	Voice. `V:1`
W:	body	Lyrics at end. `W:Help! I need...`
w:	body	inline lyrics. `w:Help! I need...`
X:	start of header	Index number. `X:1`
Z:	header	Transcription notes. `Z:Transcribed by ear`

♫ ♫ ♫ ♫ ♫ ♫ ♫

C Glossary

editor: a program to write 'ASCII text', that is with no special format. Windows' Notepad (ugh), `emacs` and `vim` are some well-known editors.

font: type of character; for instance, Times or Helvetica.

GPL: a software license that disciplines the use of many programs available from the Internet. Briefly, a GPL'ed program can be freely used, modified and shared, without having to pay for it. Please visit `http://www.gnu.org/` for more details.

MIDI: roughly speaking, the audible equivalent of sheet music. You can listen to a MIDI file using dedicated players.

PDF: file format invented by Adobe, very common on the Internet to distribute documentation. It is a spinoff of POSTSCRIPT.

POSTSCRIPT: file format invented by Adobe. Unlike graphic files like JPG, PNG or others, POSTSCRIPT is a *vector* format. This means that one can magnify the image at will, without losing in details.

system: set of staves relative to the instruments that play together in a musical piece.

string: word, set of characters.

♫ ♫ ♫ ♫ ♫ ♫ ♫

D Character Sets

One of the nicest things in the world is diversity. Different languages use different characters, but this can lead to incompatibility problems when, say, an Italian musician wants to send his or her American friend an ABC PLUS source.

Fortunately, there exist a character set called ISO 8859-1 (Latin1) that includes accented characters used by many languages: all of central Europe, and all English-speaking countries. These characters have an ASCII code between 160 and 255, and can be typed as sequences `\xxx` when not available on the keyboard.

The Latin1 characters and the corresponding octal code are shown below. I *recommend* that you use the `\xxx` sequence even if you do have these characters on your keyboard, because this makes the source more portable.

ISO 8859−1 (Latin1)

\240	¡ \241	¢ \242	£ \243	¤ \244	¥ \245	¦ \246	§ \247
¨ \250	© \251	ª \252	« \253	¬ \254	- \255	® \256	¯ \257
° \260	± \261	² \262	³ \263	´ \264	µ \265	¶ \266	· \267
¸ \270	¹ \271	º \272	» \273	¼ \274	½ \275	¾ \276	¿ \277
À \300	Á \301	Â \302	Ã \303	Ä \304	Å \305	Æ \306	Ç \307
È \310	É \311	Ê \312	Ë \313	Ì \314	Í \315	Î \316	Ï \317
Ð \320	Ñ \321	Ò \322	Ó \323	Ô \324	Õ \325	Ö \326	× \327
Ø \330	Ù \331	Ú \332	Û \333	Ü \334	Ý \335	Þ \336	ß \337
à \340	á \341	â \342	ã \343	ä \344	å \345	æ \346	ç \347
è \350	é \351	ê \352	ë \353	ì \354	í \355	î \356	ï \357
ð \360	ñ \361	ò \362	ó \363	ô \364	õ \365	ö \366	÷ \367
ø \370	ù \371	ú \372	û \373	ü \374	ý \375	þ \376	ÿ \377

♫ ♫ ♫ ♫ ♫ ♫ ♫

E Formatting Commands

Command parameters will be specified as follows:

Parameter	Type
length	unit length indicated in `cm`, `in` or `pt`
text	generic text
logical	logical value, `yes` or `no`, or 1 or 0
int	integer number
float	number with decimals
str	character string

E.1 Page Format

These commands set the page geometry.

%%botmargin ⟨***length***⟩**:** sets the page bottom margin to ⟨*length*⟩.

%%footer ⟨***text***⟩**:** sets the text to be printed as footer on each page.

%%header ⟨***text***⟩**:** sets the text to be printed as header on each page.

%%indent ⟨***length***⟩**:** sets the indentation for the first line or system to ⟨*length*⟩.

%%landscape ⟨***logical***⟩**:** if 1, sets the page layout as landscape.

%%leftmargin ⟨***length***⟩**:** sets the page left margin to ⟨*length*⟩.

%%multicol ⟨***command***⟩**:** defines columns. ⟨*command*⟩ may be `start`, `new`, and `end`. See Section 5.5 for details.

%%pageheight ⟨***length***⟩**:** sets the page height to ⟨*length*⟩. For European A4 paper, the right value is `29.7cm`; for US Letter, `11in`.

%%pagewidth ⟨***length***⟩**:** sets the page width to ⟨*length*⟩. For European A4 paper, the right value is `21cm`; for US Letter, `8.5in`.

%%rightmargin ⟨***length***⟩**:** sets the page right margin to ⟨*length*⟩.

%%staffwidth ⟨***length***⟩**:** used as an alternative to the commands `%%pageheight` and `%%pagewidth`.

%%topmargin ⟨***length***⟩**:** sets the page top margin to ⟨*length*⟩.

E.2 Text

These commands are used to write text lines in and between the tunes. The font and spacing are set with other commands that we will examine later on.

%%begintext...%%endtext : the pair `%%begintext` and `%%endtext` includes a group of text lines. These lines will be printed. If no text follows `%%`, the line is a paragraph separator. For example:

```
%%begintext
Spanish folk song, usually
accompanied by guitar and cymbals.
%%endtext
```

The command `%%begintext` can be given a parameter to change the text alignment:

%%begintext obeylines prints text as is;
%%begintext fill (or `ragged`) formats the text to the page margins;
%%begintext justify (or `align`) as above, but aligns to the page right margin;
%%begintext skip ignores the following lines.

%%center ⟨***text***⟩**:** centers the following text.

%%text ⟨***text***⟩**:** writes the following text. For example:

```
%%text Spanish folk song
```

%%textoption ⟨***string***⟩**:** sets the default text option to be used between `%%begintext`/`%%endtext`.

E.3 Fonts

These commands specify the character fonts used in various parts of a score. Please note that the common True Type fonts used by Windows *are not the same fonts* used by `abcm2ps`. In fact, `abcm2ps` uses the POSTSCRIPT fonts, provided for and managed by Ghostscript.

Standard fonts are shown in Appendix G. I remind you that indications for adding new fonts are given in Section 9.4.

`%%annotationfont` ⟨***string***⟩**:** font of annotations.

`%%composerfont` ⟨***string***⟩**:** `C:` field font.

`%%footerfont` ⟨***string***⟩**:** font of `%%footer` lines.

`%%font` ⟨***string***⟩**:** declares a font for later usage.

`%%gchordfont` ⟨***string***⟩**:** guitar chords font.

`%%headerfont` ⟨***string***⟩**:** font of `%%header` lines.

`%%historyfont` ⟨***string***⟩**:** font of `H:` field.

`%%infofont` ⟨***string***⟩**:** text font in `I:` fields.

`%%measurefont` ⟨***string***⟩ ***[box]*****:** text font of measure numbers. If the word `box` is present, a box is drawn around the measure number.

`%%partsfont` ⟨***string***⟩**:** `P:` fields font.

`%%repeatfont` ⟨***string***⟩**:** font of repeat numbers or text.

`%%setfont-`⟨***int***⟩ ⟨***string***⟩ ⟨***int***⟩**:** sets an alternate font for strings. In most strings, the current font may be changed by `$n` (n = 1, 2, 3, 4). `$0` resets the font to the default value.

`%%subtitlefont` ⟨***string***⟩**:** font of the second `T:` field.

`%%tempofont` ⟨***string***⟩**:** tempo font.

`%%textfont` ⟨***string***⟩**:** text font in `%%text` lines.

`%%titlecaps` ⟨***logical***⟩**:** if true, writes the title in capital letters.

`%%titlefont` ⟨***string***⟩**:** font of the first `T:` field.

`%%titleformat` ⟨***string***⟩**:** defines the format of the tune title. This format overrides `%%titleleft`, `%%infoline`, and `%%composerspace`. See Section 5.6 for examples.

`%%titleleft` ⟨***logical***⟩**:** if true, writes the title left-aligned instead of centered.

`%%voicefont` ⟨***string***⟩**:** font of voice names.

`%%vocalfont` ⟨***string***⟩**:** font of the text in `w:` lines.

`%%wordsfont` ⟨***string***⟩**:** font of the text in `W:` lines.

E.4 Spacing

These commands specify spacing between score elements.

%%barsperstaff ⟨***int***⟩**:** try to typeset the score with ⟨*int*⟩ bars on each line.

%%composerspace ⟨***length***⟩**:** sets the vertical space before the composer to ⟨*length*⟩.

%%infospace ⟨***length***⟩**:** sets the vertical space before the infoline to ⟨*length*⟩.

%%lineskipfac ⟨***float***⟩**:** sets the factor for spacing between lines of text to ⟨*float*⟩.

%%maxshrink ⟨***float***⟩**:** sets how much to compress horizontally when staff breaks are chosen automatically. ⟨*float*⟩ must be between 0 (don't shrink) and 1 (full shrink).

%%maxstaffsep ⟨***length***⟩**:** sets the maximum vertical space between staves.

%%maxsysstaffsep ⟨***length***⟩**:** sets the maximum vertical space between systems.

%%musicspace ⟨***length***⟩**:** sets the vertical space before the first staff to ⟨*length*⟩.

%%newpage: sets a page break.

%%notespacingfactor ⟨***float***⟩**:** sets the proportional spacing of notes. The default value is 1.414 ($\sqrt{2}$); 1 makes all notes equally spaced.

%%parskipfac ⟨***float***⟩**:** sets the factor for spacing between parts to ⟨*float*⟩.

%%partsspace ⟨***length***⟩**:** sets the vertical space before a new part to ⟨*length*⟩.

%%scale ⟨***float***⟩**:** sets the music scale factor to ⟨*float*⟩.

%%sep: prints a centered separator (a short line).

%%sep ⟨***length1***⟩ ⟨***length2***⟩ ⟨***length3***⟩**:** prints a separator of length ⟨*length3*⟩, with spacing ⟨*length1*⟩ above and ⟨*length2*⟩ below.

%%slurheight ⟨***float***⟩**:** sets the slur height factor; lesser than 1 flattens the slur, greater than 1 expands it.

%%staffbreak ⟨***length***⟩**:** sets a ⟨*length*⟩-long break (gap) in the current staff.

%%staffsep ⟨***length***⟩**:** sets the vertical space between different systems to ⟨*length*⟩.

%%stretchlast ⟨***logical***⟩**:** stretches the last staff of the tune when underfull.

%%stretchstaff ⟨***logical***⟩**:** stretches underfull staves across page.

%%subtitlespace ⟨***length***⟩**:** sets the vertical space before the subtitle to ⟨*length*⟩.

%%sysstaffsep ⟨***length***⟩**:** sets the vertical space between staves in the same system to ⟨*length*⟩.

%%textspace ⟨***length***⟩**:** sets the vertical space before texts to ⟨*length*⟩.

%%titlespace ⟨***length***⟩**:** sets the vertical space before the title to ⟨*length*⟩.

%%topspace ⟨***length***⟩**:** sets the vertical space at the top of a tune to ⟨*length*⟩. Note that a tune may begin with `%%text` commands before the title.

%%vocalspace ⟨***length***⟩**:** sets the vertical space before the lyrics under staves to ⟨*length*⟩.

%%vskip ⟨***h***⟩**:** adds ⟨*h*⟩ vertical space.

%%wordsspace ⟨***length***⟩**:** sets the vertical space before the lyrics at end of the tune to ⟨*length*⟩.

E.5 Other Commands

Miscellaneous commands are grouped in this section.

%%alignbars ⟨***int***⟩**:** aligns the bars of the next ⟨*int*⟩ lines of music. It only works on single-voice tunes.

%%aligncomposer ⟨***int***⟩**:** specifies where to print the composer field. A negative value means 'on the left', 0 means 'centre', and a positive value means 'on the right'.

%%autoclef ⟨***logical***⟩**:** if false, prevents the automatic change of clef when notes are too low or too high.

%%barnumbers ⟨***int***⟩**:** same as `%%measurenb`, see below.

%%beginps–%%endps: start/end of a POSTSCRIPT sequence.

%%bstemdown ⟨***logical***⟩**:** if true, the stem of the note on the middle of the staff goes downwards. Otherwise, it goes upwards or downwards according to the previous note.

%%comball ⟨***logical***⟩**:** if true and `%%combinevoices` is set, combines voices in all cases.

%%combinevoices ⟨***logical***⟩**:** if true, notes of same duration that belong to voices of the same staff are combined producing chords. It does not apply when note pitches are in unison, inverted or differ by a second.

%%continueall ⟨***logical***⟩**:** ignores the line breaks in tune if true. It's the equivalent of the `-c` command line flag.

%%contbarnb ⟨***logical***⟩**:** if true, the bar number of the second repeat(s) is reset to the number of the first repeat. If false, bars are sequentially numbered.

%%dateformat ⟨***string***⟩**:** defines the format of date and time. Default is `%b %e, %Y %H:%M`. The fields specify, respectively: abbreviated month name (Jan–Dec), day of month (1–31), year, hour (0–23), minute (0–59).

%%deco ⟨***string1***⟩ ⟨***int1***⟩ ⟨***string2***⟩ ⟨***int2***⟩ ⟨***int3***⟩ ⟨***int4***⟩ ⟨***string3***⟩**:** adds a new decoration. Details are explained in Section 9.

%%dynalign ⟨***logical***⟩**:** if true, horizontally aligns the dynamic marks.

%%EPS ⟨***string3***⟩**:** includes an external EPS file in the score.

%%encoding ⟨***int***⟩**:** (for expert users) sets the language encoding to ISO-Latin⟨*int*⟩, which may range from 0 to 6. The value 0 is the same as 1, but the POSTSCRIPT encoding table is not output.

%%exprabove ⟨***logical***⟩**:** draws the expression decorations above the staff. If neither `exprabove` nor `exprbelow` are true, the expression decorations are drawn above the staff if there are lyrics on the staff, below otherwise. `exprabove` takes precedence over `exprbelow`.

%%exprbelow ⟨***logical***⟩**:** draws the expression decorations below the staff.

%%flatbeams ⟨***logical***⟩**:** if true, forces flat beams in bagpipe tunes (`K:HP`).

%%freegchord ⟨***logical***⟩**:** if true, prevents the characters `#`, `b` and `=` from being displayed as sharp, flat and natural sign in guitar chords. When this flag is set, displaying accidental may be forced escaping the characters (e.g. `\#`.)

%%format ⟨***string***⟩**:** reads the format file specified as parameter.

%%gchordbox ⟨***logical***⟩**:** draws a box around accompaniment chords.

%%graceslurs ⟨***logical***⟩**:** draws slurs on grace notes.

%%hyphencont ⟨***logical***⟩**:** if true and if lyrics under the staff end with a hyphen, puts a hyphen in the next line.

%%infoline ⟨***logical***⟩**:** displays the rhythm and the origin on the same line.

%%measurenb ⟨***int***⟩**:** draws the measure number every ⟨*int*⟩ bars.

%%measurebox ⟨***logical***⟩**:** draws a box around measure numbers.

%%measurefirst ⟨***int***⟩**:** starts numbering the measures from ⟨*int*⟩.

%%musiconly ⟨***logical***⟩**:** if true, doesn't output the lyrics.

%%oneperpage ⟨***logical***⟩**:** outputs one tune per page.

%%partsbox ⟨***logical***⟩**:** draws a box around part names.

%%postscript ⟨***string***⟩**:** a series of these commands lets the user add a new POSTSCRIPT routine, or change an existing one.

%%printparts ⟨***logical***⟩**:** prints the part indications (`P:`).

%%printtempo ⟨***logical***⟩**:** prints the tempo indications (`Q:`).

%%pslevel ⟨***int***⟩**:** specifies the POSTSCRIPT level (1, 2, or 3).

%%repbra ⟨***logical***⟩**:** if false, prevents displaying repeat brackets for the current voice.

%%setbarnb ⟨***int***⟩**:** sets the number of the next measure.

%%setdefl ⟨***logical***⟩**:** if true, outputs some indications about the note/chord and/or decorations for customization purposes. These indications are stored in the PostScript variable 'defl'.

%%shifthnote ⟨***logical***⟩**:** in multivoice tunes, when voices go to unison with a white and a black note (say, a half and a quarter note), only the black note head is printed. When this flag is set, a note shift is done and both are printed.

%%splittune ⟨***logical***⟩**:** if true, splits tunes that do not fit in a single page.

%%squarebreve ⟨***logical***⟩**:** displays 'brevis' notes in square format.

%%straightflags ⟨***logical***⟩**:** prints straight flags on stems in bagpipe tunes.

%%staff ⟨***int***⟩**:** prints the next symbols of the current voice on the ⟨*int*⟩-th staff.

%%staves ⟨***string***⟩**:** defines how staves are to be printed. See Section 4.2 for details.

%%stemheight ⟨***float***⟩**:** sets the stem height to ⟨*float*⟩.

%%timewarn ⟨***logical***⟩**:** if true, if a time signature occurs at the beginning of a music line, a cautionary time signature is added at the end of the previous line.

%%tuplets ⟨***int1***⟩ ⟨***int2***⟩ ⟨***int3***⟩**:** defines how tuplets are to be drawn. See Section 9.5 for details.

%%vocalabove ⟨***logical***⟩**:** draws the vocals above the staff.

%%withxrefs ⟨***logical***⟩**:** prints the `X:` number in the title.

%%writehistory ⟨***logical***⟩**:** outputs notes, history, etc.

♫ ♫ ♫ ♫ ♫ ♫ ♫

F abcMIDI commands

Some of these commands will only make sense to advanced users who have some experience with MIDI files. In a few cases, explanations are taken from the `abcguide.txt` file included in the `abcMIDI` archive.

`%%MIDI barlines`: deactivates `%%nobarlines`.

`%%MIDI bassprog` ⟨*int*⟩: sets the MIDI instrument for the bass notes to ⟨*int*⟩ (0–127).

`%%MIDI bassvol` ⟨*int*⟩: sets the velocity (i.e., volume) of the bass notes to ⟨*int*⟩ (0–127).

`%%MIDI beat` ⟨*int1*⟩ ⟨*int2*⟩ ⟨*int3*⟩ ⟨*int4*⟩: controls the volumes of the notes in a measure. The first note in a bar has volume ⟨*int1*⟩; other 'strong' notes have volume ⟨*int2*⟩ and all the rest have volume ⟨*int3*⟩. These values must be in the range 0–127. The parameter ⟨*int4*⟩ determines which notes are 'strong'. If the time signature is x/y, then each note is given a position number k = 0, 1, 2... x-1 within each bar. If k is a multiple of ⟨*int4*⟩, then the note is 'strong'.

`%%MIDI beataccents`: reverts to normally emphasised notes. See `%%MIDI nobeataccents`.

`%%MIDI beatmod` ⟨*int*⟩: increments the velocities as defined by `%%MIDI beat`

`%%MIDI beatstring` ⟨*string*⟩: similar to `%%MIDI beat`, but indicated with an `fmp` string.

`%%MIDI c` ⟨*int*⟩: specifies the MIDI pitch which corresponds to `c`. The default is 60.

`%%MIDI channel` ⟨*int*⟩: selects the melody channel ⟨*int*⟩ (1–16).

`%%MIDI chordattack` ⟨*int*⟩: delays the start of chord notes by ⟨*int*⟩ MIDI units.

`%%MIDI chordname` ⟨*string int1 int2 int3 int4 int5 int6*⟩: defines new chords or re-defines existing ones as was seen in Section 11.7.

`%%MIDI chordprog` ⟨*int*⟩: sets the MIDI instrument for accompaniment chords to ⟨*int*⟩ (0–127).

`%%MIDI chordvol` ⟨*int*⟩: sets the volume (velocity) of the chord notes to ⟨*int*⟩ (0–127).

`%%MIDI control` ⟨*bass/chord*⟩ ⟨*int1 int2*⟩: generates a MIDI control event. If `%%control` is followed by ⟨*bass*⟩ or ⟨*chord*⟩, the event apply to the bass or chord channel, otherwise it will be applied to the melody channel. ⟨*int1*⟩ is the MIDI control number (0–127) and ⟨*int2*⟩ the value (0–127).

`%%MIDI deltaloudness` ⟨*int*⟩: by default, `!crescendo!` and `!dimuendo!` modify the beat variables ⟨*vol1*⟩ ⟨*vol2*⟩ ⟨*vol3*⟩ 15 volume units. This command allows the user to change this default.

`%%MIDI drone` ⟨*int1 int2 int3 int4 int5*⟩: specifies a two-note drone accompaniment. ⟨*int1*⟩ is the drone MIDI instrument, ⟨*int2*⟩ the MIDI pitch 1, ⟨*int3*⟩ the MIDI pitch 2, ⟨*int4*⟩ the MIDI volume 1, ⟨*int5*⟩ the MIDI volume 2. Default values are 70 45 33 80 80.

`%%MIDI droneoff`: turns the drone accompaniment off.

`%%MIDI droneon`: turns the drone accompaniment on.

`%%MIDI drum` ⟨*str*⟩ ⟨*int1 int2 int3 int4 int5 int6 int7 int8*⟩: generates a drum accompaniment pattern, as described in Section 11.9.

`%%MIDI drummap` ⟨*str*⟩ ⟨*int*⟩: associates the note ⟨*str*⟩ (in ABC notation) to the a percussion instrument, as listed in Section H.2.

`%%MIDI drumon` turns drum accompaniment on.

%%MIDI drumoff turns drum accompaniment off.

%%MIDI fermatafixed: expands a `!fermata!` by one unit length; that is, `GC3` becomes `C4`.

%%MIDI fermataproportional: doubles the length of a note preceded by `!fermata!`; that is, `GC3` becomes `C6`. `abc2midi` does this by default.

%%MIDI gchord: sets up how guitar chords are generated. Please see Section 11.4.

%%MIDI gchordoff: turns guitar chords off.

%%MIDI gchordon: turns guitar chords on.

%%MIDI grace ⟨***float***⟩**:** sets the fraction of the next note that grace notes will take up. ⟨*float*⟩ must be a fraction such as `1/6`.

%%MIDI gracedivider ⟨***int***⟩**:** sets the grace note length as 1/⟨*int*⟩th of the following note.

%%MIDI makechordchannels ⟨***int***⟩**:** this is a very complex command used in chords containing microtones. Please consult `abcMIDI` documentation.

%%MIDI nobarlines: makes a note's accidental apply only to the following note and not to all notes until the end of the measure.

%%MIDI nobeataccents: forces the ⟨*int2*⟩ volume (see `%%MIDI beat`) for each note in a bar, regardless of their position.

%%MIDI noportamento: turns off the portamento controller on the current channel.

%%MIDI pitchbend ⟨***bass/chord***⟩ ⟨***int1 int2***⟩**:** generates a pitchbend event on the current channel, or on the bass or chord channel as specified. The value given by the following two bytes indicates the pitch change. This option is not well documented.

%%MIDI portamento ⟨***int***⟩**:** turns on the portamento controller on the current channel and set it to ⟨*int*⟩. Experts only.

%%MIDI program ***[int1]*** ⟨***int2***⟩**:** selects the program (instrument) ⟨*int2*⟩ (0–127) for channel ⟨*int1*⟩. If this is not specified, the instrument will apply to the current channel.

%%MIDI randomchordattack: delays the start of chord notes by a random number of MIDI units.

%%MIDI ratio ⟨***int1 int2***⟩**:** sets the ratio of note lengths in broken rhythm. Normally `c>c` will make the first note three times as long as the second; this ratio can be changed with `%%ratio 2 1`.

%%MIDI rtranspose ⟨***int1***⟩**:** transposes relatively to a prior `%%transpose` command by ⟨*int1*⟩ semitones; the total transposition will be ⟨*int1* + *int2*⟩ semitones.

%%MIDI temperamentlinear ⟨***float1 float2***⟩**:** changes the temperament of the scale. ⟨*float1*⟩ specifies the size of an octave in cents of a semitone, or 1/1200 of an octave. ⟨*float2*⟩ specifies in the size of a fifth (normally 700 cents).

%%MIDI temperamentnormal: restores normal temperament.

%%MIDI transpose ⟨***int1***⟩**:** transposes the output by ⟨*int1*⟩ semitones. ⟨*int1*⟩ may be positive or negative.

%%MIDI trim ⟨*int1*⟩ ⟨*int2*⟩**:** controls the articulation of notes and chords by placing silent gaps between the notes. The length of these gaps is determined by ⟨*int1*⟩/⟨*int2*⟩ and the unit length is specified by the `L:` command. These gaps are produced by shortening the notes by the same amount. If the note is already shorter than the specified gap, then the gap is set to half the length of the note. It is recommended that ⟨*int1*⟩/⟨*int2*⟩ be a fraction close to zero. Trimming is disabled inside slurs as indicated by parentheses. Trimming is disabled by setting ⟨*int1*⟩ to 0.

♫ ♫ ♫ ♫ ♫ ♫ ♫

G PostScript Fonts

There are 35 standard POSTSCRIPT fonts. They are all listed below, with the exception of ZapfDingbats which is not supported by `abcm2ps`.

Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Courier
Courier-Oblique
Courier-Bold
Courier-BoldOblique
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Helvetica
Helvetica-Oblique
Helvetica-Bold
Helvetica-BoldOblique
Helvetica-Narrow
Helvetica-Narrow-Oblique
Helvetica-Narrow-Bold
Helvetica-Narrow-BoldOblique
Palatino-Roman
Palatino-Italic
Palatino-Bold
Palatino-BoldItalic
NewCenturySchlbk-Roman
NewCenturySchlbk-Italic
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
Times-Roman
Times-Italic
Times-Bold
Times-BoldItalic
Σψμβολ
ZapfChancery-MediumItalic

♫ ♫ ♫ ♫ ♫ ♫ ♫

H MIDI Instruments

H.1 Standard instruments

The following is a complete list of the General MIDI standard instruments, subdivided according to instrument family. Remember, when using `abc2midi` the instrument number must be decreased by 1.

Piano
1. Acoustic Grand
2. Bright Acoustic
3. Electric Grand
4. Honky-Tonk
5. Electric Piano 1
6. Electric Piano 2
7. Harpsichord
8. Clavinet

Guitar
25. Nylon String Guitar
26. Steel String Guitar
27. Electric Jazz Guitar
28. Electric Clean Guitar
29. Electric Muted Guitar
30. Overdriven Guitar
31. Distortion Guitar
32. Guitar Harmonics

Ensemble
49. String Ensemble 1
50. String Ensemble 2
51. SynthStrings 1
52. SynthStrings 2
53. Choir Aahs
54. Voice Oohs
55. Synth Voice
56. Orchestra Hit

Pipe
73. Piccolo
74. Flute
75. Recorder
76. Pan Flute
77. Blown Bottle
78. Skakuhachi
79. Whistle
80. Ocarina

Synth Effects
97. FX 1 (rain)
98. FX 2 (soundtrack)
99. FX 3 (crystal)
100. FX 4 (atmosphere)
101. FX 5 (brightness)
102. FX 6 (goblins)
103. FX 7 (echoes)
104. FX 8 (sci-fi)

Sounds Effects
121. Guitar Fret Noise
122. Breath Noise
123. Seashore
124. Bird Tweet
125. Telephone Ring
126. Helicopter
127. Applause
128. Gunshot

Chromatic Percussion
9. Celesta
10. Glockenspiel
11. Music Box
12. Vibraphone
13. Marimba
14. Xylophone
15. Tubular Bells
16. Dulcimer

Bass
33. Acoustic Bass
34. Electric Bass(finger)
35. Electric Bass(pick)
36. Fretless Bass
37. Slap Bass 1
38. Slap Bass 2
39. Synth Bass 1
40. Synth Bass 2

Brass
57. Trumpet
58. Trombone
59. Tuba
60. Muted Trumpet
61. French Horn
62. Brass Section
63. SynthBrass 1
64. SynthBrass 2

Synth Lead
81. Lead 1 (square)
82. Lead 2 (sawtooth)
83. Lead 3 (calliope)
84. Lead 4 (chiff)
85. Lead 5 (charang)
86. Lead 6 (voice)
87. Lead 7 (fifths)
88. Lead 8 (bass+lead)

Ethnic
105. Sitar
106. Banjo
107. Shamisen
108. Koto
109. Kalimba
110. Bagpipe
111. Fiddle
112. Shanai

Organ
17. Drawbar Organ
18. Percussive Organ
19. Rock Organ
20. Church Organ
21. Reed Organ
22. Accordion
23. Harmonica
24. Tango Accordion

Solo Strings
41. Violin
42. Viola
43. Cello
44. Contrabass
45. Tremolo Strings
46. Pizzicato Strings
47. Orchestral Strings
48. Timpani

Reed
65. Soprano Sax
66. Alto Sax
67. Tenor Sax
68. Baritone Sax
69. Oboe
70. English Horn
71. Bassoon
72. Clarinet

Synth Pad
89. Pad 1 (new age)
90. Pad 2 (warm)
91. Pad 3 (polysynth)
92. Pad 4 (choir)
93. Pad 5 (bowed)
94. Pad 6 (metallic)
95. Pad 7 (halo)
96. Pad 8 (sweep)

Percussive
113. Tinkle Bell
114. Agogo
115. Steel Drums
116. Woodblock
117. Taiko Drum
118. Melodic Tom
119. Synth Drum
120. Reverse Cymbal

H.2 Percussion Instruments

These instruments can be used with `%%MIDI drum`, or using the corresponding note in the MIDI channel 10.

35.	`B,,,`	Acoustic Bass Drum	**36**.	`C,,`	Bass Drum 1	**37**.	`^C,,`	Side Stick
38.	`D,,`	Acoustic Snare	**39**.	`^D,,`	Hand Clap	**40**.	`E,,`	Electric Snare
41.	`F,,`	Low Floor Tom	**42**.	`^F,,`	Closed Hi Hat	**43**.	`G,,`	High Floor Tom
44.	`^G,,`	Pedal Hi-Hat	**45**.	`A,,`	Low Tom	**46**.	`^A,,`	Open Hi-Hat
47.	`B,,`	Low-Mid Tom	**48**.	`C,`	Hi Mid Tom	**49**.	`^C,`	Crash Cymbal 1
50.	`D,`	High Tom	**51**.	`^D,`	Ride Cymbal 1	**52**.	`E,`	Chinese Cymbal
53.	`F,`	Ride Bell	**54**.	`^F,`	Tambourine	**55**.	`G,`	Splash Cymbal
56.	`^G,`	Cowbell	**57**.	`A,`	Crash Cymbal 2	**58**.	`^A,`	Vibraslap
59.	`B,`	Ride Cymbal 2	**60**.	`C`	Hi Bongo	**61**.	`^C`	Low Bongo
62.	`D`	Mute Hi Conga	**63**.	`^D`	Open Hi Conga	**64**.	`E`	Low Conga
65.	`F`	High Timbale	**66**.	`^F`	Low Timbale	**67**.	`G`	High Agogo
68.	`^G`	Low Agogo	**69**.	`A`	Cabasa	**70**.	`^A`	Maracas
71.	`B`	Short Whistle	**72**.	`c`	Long Whistle	**73**.	`^c`	Short Guiro
74.	`d`	Long Guiro	**75**.	`^d`	Claves	**76**.	`e`	Hi Wood Block
77.	`f`	Low Wood Block	**78**.	`^f`	Mute Cuica	**79**.	`g`	Open Cuica
80.	`^g`	Mute Triangle	**81**.	`a`	Open Triangle			

♫ ♫ ♫ ♫ ♫ ♫ ♫