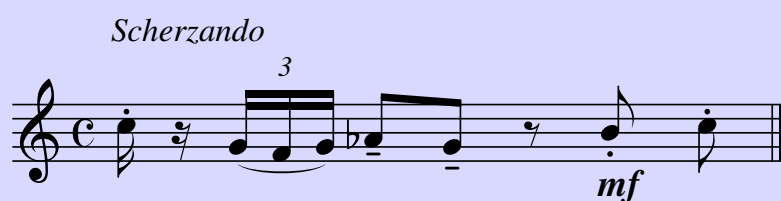


Guido Gonzato

Making Music with ABC PLUS



A guide to the notation and its applications

Making Music with ABC PLUS

Copyright © Guido Gonzato, PhD, 2003–2012

Latest update: March 30, 2012

Typeset with L^AT_EX, with the help of the [Jed editor](#) and L^AT_EX⁴JED.

This manual is released under the terms of the GNU Free Documentation License,

<http://www.gnu.org/licenses/fdl.html>

The latest version of this manual is available at:

<http://abcplus.sourceforge.net/#ABCGuide>

*To Annarosa,
Bruno,
Lorenzo*



Contents

About This Book	xiii
1 Music on the Computer with ABC PLUS	1
1.1 Introduction	1
1.1.1 Requirements	2
1.1.2 Software	2
1.1.3 Why ABC?	3
1.2 Getting Started	6
1.2.1 Installing the Programs	6
1.2.2 ABC in a Nutshell	7
1.2.3 Our First Score	8
2 Melody	13
2.1 Notes and Symbols	13
2.1.1 Pitch: <i>A-G a-g , '</i>	13
2.1.2 Note Length: <i>L:</i>	14
2.1.3 Rests and Spacing: <i>z Z x y</i>	15
2.1.4 Accidentals: <i>^ _ =</i>	16
2.1.5 Dotted Notes: <i>< ></i>	17
2.1.6 Ties, Slurs, Staccato: <i>- ()</i>	18
2.1.7 Tuplets: <i>(n</i>	18
2.1.8 Chords: <i>[]</i>	19
2.1.9 Lyrics: <i>W: w:</i>	20
2.1.10 Foreign Characters	22
Never Type Accented Letters!	22
2.1.11 Grace Notes: <i>~ {}</i>	23
2.1.12 Forcing Line Breaks: <i>!</i>	24
2.1.13 Avoiding Line Breaks: <i>\</i>	24
2.1.14 Inline Fields	25
2.2 Music Properties	26
2.2.1 Key signatures and Clefs: <i>K:</i>	26
Key Signatures	26

	All Seven Clefs	27
	Bass and Alto Clefs Compatibility Issues	29
2.2.2	Metre: <i>M:</i>	30
2.2.3	Bars and Repeats: / : []	30
2.2.4	Title, Composer, Tempo: <i>T:</i> <i>C:</i> <i>Q:</i>	32
2.2.5	Parts: <i>P:</i>	32
2.2.6	Accompaniment Chords: " "	33
2.2.7	Text Annotations: " ^ _ < > @ "	35
	Figured Bass	36
2.2.8	Ornaments: ! <i>symbol</i> !	36
2.2.9	Redefinable Symbols: <i>U:</i>	38
2.2.10	Information Fields	39
3	Harmony	41
3.1	Polyphony in ABC	41
3.1.1	Voices and Systems: <i>V:</i>	41
3.1.2	Positioning Voices: %% <i>staves</i> and %% <i>score</i>	44
3.1.3	Voice Splitting: &	48
3.1.4	Change of System	49
4	Page Layout	53
4.1	Formatting Parameters	53
4.1.1	Changing Parameters	56
4.1.2	The Grand Staff	57
4.1.3	Using Fonts and Text	57
4.1.4	Staff Breaks	60
4.1.5	Multi-column Output	61
4.1.6	Customising Titles	62
4.1.7	Headers and Footers	63
4.1.8	Inserting Graphics Files	64
4.2	Format files	64
4.3	Numbering Measures and Pages	65
4.3.1	Measure Control	66
4.4	Saving Space	66
4.5	Advanced Customisation (Experts Only!)	67
4.5.1	New PostScript Routines	67
4.5.2	Accompaniment Chords in Italian Notation	67
4.5.3	Defining New Symbols	68
4.5.4	Adding Fonts	70
4.5.5	Customising Tuplets	71

4.5.6	Tin Whistle Fingerings	72
5	Playing	75
5.1	MIDI Conversion	75
5.1.1	A Software MIDI Player: TiMidity++	76
5.1.2	Our First Midi	76
5.1.3	Example: How To Make a Ringtone	77
5.1.4	Supported Ornaments	78
5.1.5	%%MIDI Commands	78
5.1.6	Voices and Instruments	79
5.1.7	Accompaniment Chords	80
5.1.8	Customising Beats	82
5.1.9	Arpeggios	83
5.1.10	New Accompaniment Chords	83
5.1.11	Broken Rhythm	84
5.1.12	Drum Patterns	85
5.1.13	Percussion Instruments	86
5.1.14	Portamento	87
5.1.15	Drone	87
5.1.16	midi2abc	88
5.1.17	Global Settings	89
5.2	Differences and Incompatibilities	90
6	Converting	91
6.1	The abcpp Preprocessor	91
6.1.1	Basic Usage	91
6.1.2	Advanced Usage	92
6.2	abc2abc	94
7	Other Possibilities	97
7.1	Inserting Music in Other Programs	97
7.2	Inserting Music in L ^A T _E X	97
7.2.1	Using abc.sty	98
7.3	Converting Graphics to EPS	99
7.4	Parts Extraction	99
7.5	Limitations of abcm2ps	100
7.6	Final Comments	100
7.6.1	Please, Make a Donation...	100
7.6.2	In Loving Memory of Annarosa Del Piero, 1930–2000	101
A	Bits & Pieces	103

A.1	Web Links	103
A.2	ABC Fields	104
A.3	Glossary	104
A.4	Character Sets	105
A.5	Formatting Commands	105
A.5.1	Page Format	106
A.5.2	Text	106
A.5.3	Fonts	107
A.5.4	Spacing	108
A.5.5	Other Commands	110
A.6	abcMIDI commands	112
A.7	PostScript Fonts	115
A.8	MIDI Instruments	116
A.8.1	Standard instruments	116
A.8.2	Percussion Instruments	117

List of Tables

1.1	Comparison between note names in different notations.	12
2.1	How to obtain characters of foreign languages.	22
2.2	Correspondence between the key and the number of sharps or flats.	26
2.3	Modal scales.	27
2.4	Clefs and associated \mathbb{K} : fields.	28
2.5	Types of accompaniment chords.	34
2.6	Standard abbreviations for common symbols.	39
5.1	Standard notes and corresponding MIDI pitches.	88

List of Figures

1.1	Managing a tune collection with ABCexplorer.	3
1.2	Writing a tune with JEDABC and <code>runabc.tcl</code>	4
1.3	Writing a tune with the Folkinfo abc converter.	5
2.1	Standard expression symbols.	37
3.1	A piece where the system changes twice.	51
4.1	Ave Verum with formatting parameters.	55
4.2	Alternating text with music.	59
4.3	Using different fonts in strings	60
5.1	Converting a MIDI file to ABC with <code>runabc.tcl</code>	89

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 audio 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-)

Chapter 1

Music on the Computer with ABC PLUS

1.1 Introduction

IF you are a musician and if you 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, even for free.

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

Visual programs are easier for beginners and are probably more intuitive, but text-based notations make for faster transcription and have other advantages.

Many text-based notations have been invented. ABC, introduced by Chris Walshaw in 1991, is one of the best: being simple, easy to learn yet 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. The ABC home page is <http://abcnotation.com>.

ABC was later expanded to provide multiple voices (polyphony), page layout details, and MIDI commands. This was called ABC 2.1: its formal description is available at <http://abcnotation.com/wiki/abc:standard:v2.1>.

A few programs implement most ABC 2.1 features and provide some extensions. 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. And it's also free!

The ABC PLUS home page is <http://abcplus.sourceforge.net>.

A Few Words About “Standards”. While ABC is formally defined, the status of ABC PLUS is a bit fuzzy. It has no official definition, and it can be described as “whatever feature and/or extension to ABC 2.* is implemented by the best software available”. In other words, ABC 2.* is a theoretical standard, while ABC PLUS is what you can do with available programs. It should be emphasised that ABC PLUS is fully upwards compatible with ABC; in the future, these two “releases” of the same music notation language might even merge.

For the sake of simplicity, in the following I shall simply write “ABC” instead of “ABC PLUS”.

1.1.1 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. Expertise in the Windows or Unix command line interface is not required. It is required, however, that you can read music: the treble clef and two octaves starting at middle C should suffice.

1.1.2 Software

The state of the art of ABC is currently represented by two great little programs: the `abcm2ps` typesetter, and the MIDI creator `abc2midi`. Both are free software released under the GNU GPL license.

`abcm2ps` reads ABC 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, Acrobat PDF being the most important.

The author of `abcm2ps`, an organist and programmer called Jean-François Moine, releases “stable” and “development” versions of his program. The latter has many enhancements, but it may be buggy. As of this writing,¹ the stable release is 6.6.8, the previous stable release is 5.9.25, the development release is 7.0.3. This guide will only describe features of the former stable release 5.9.25; the new release introduces several changes (most notably, SVG output, UTF-8 support, and transposition) which will be described in future versions of this guide.

`abc2midi` converts ABC files to MIDI. It is part of the `abcMIDI` package, which include other utility applications. Its usage will be described in Part 5 (page 75) of this guide.

Since ABC 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 files. In general, any application that can be used to write text can also be used to write ABC files.

`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. Geeky stuff.

Scared off? Don’t worry: the command line can be avoided. There exist programs that gather all relevant pieces of software in a single integrated environment:

- ABCexplorer (Figure 1.1), a Windows application;
- EasyABC, multiplatform;

¹March 30, 2012.

- JEDABC (Figure 1.2), primarily designed for GNU/Linux but also available on Windows;
- `runabc.tcl`, multiplatform;
- The Folkinfo Abc Converter (Figure 1.3): a web interface to `abcm2ps` and `abcMIDI`. Highly recommended.

I feel that none of these programs are “perfect”; however, I would suggest that Windows users adopt ABCexplorer, while GNU/Linux and Mac OS X users may prefer JEDABC or `runabc.tcl` to other general-purpose editors like `vim` or `gedit`. The Folkinfo converter is probably the easiest solution, as you don’t have to install anything.

Many more applications are available and are listed on the ABC home page.

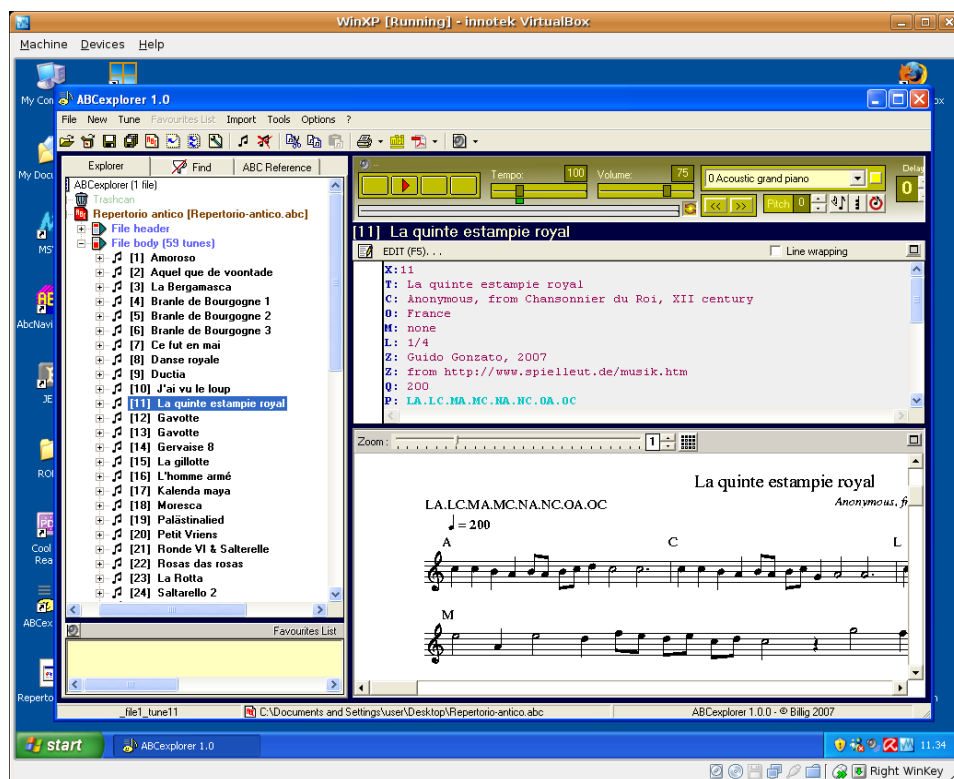


Figure 1.1: Managing a tune collection with ABCexplorer.

1.1.3 Why ABC?

I know from experience that graphical programs (nearly all are commercial software) are considered easier to use than non-graphical ones.² So, why should one learn ABC?

Well, compared to graphical programs ABC has many advantages:

smart: ABC allows for small and readable files, easy searching and indexing of tunebooks, easy creation of music archives, etc;

²erroneously, but only expert users realize it.

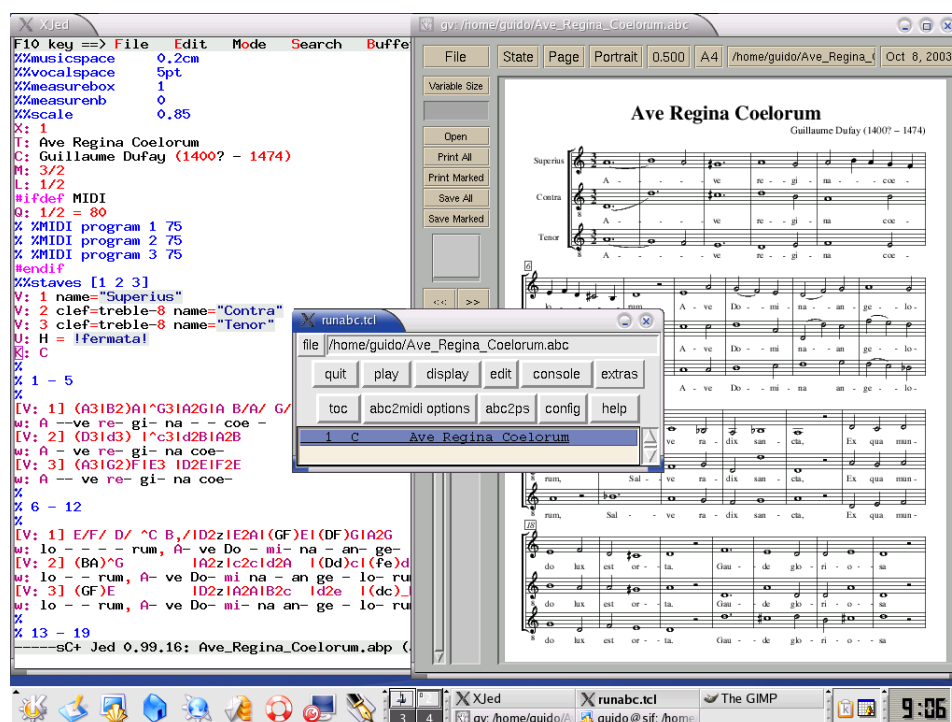


Figure 1.2: Writing a tune with JEDABC and runabc.tcl.

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

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

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

quality: using abcm2ps, ABC produces publication quality sheet music;

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

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

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;

flexibility: inserting music written in ABC in web pages or word processor files is very simple;

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

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

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

Needless to say, there are disadvantages as well:

Folkinfo - Abc Converter - Mozilla Firefox

File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto

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

Most Visited Getting Started Latest Headlines

Folkinfo - Abc Converter

folkinfo.org

Welcome to folkinfo

[forum](#) | [song db](#) | [abc converter](#) | [events](#) | [membership](#) | [login/out](#) | [personal](#) | [FAQ](#)

Forum Search Go

Quick Search: Search

[midi](#) | [pdf](#)

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' |

Messages: ☐ abcm2ps ☐ abc2midi
Switches: ☐ Notes & History
Transpose: No Transpose ☐ show transposed abc
Broken notation midi: 3:1 (abc standard ratio)
clear reset submit

Transposing help

From A to A transpose down 0 semitones or up 0 semitones

Completato [Contact us](#)

Figure 1.3: Writing a tune with the Folkinfo abc converter.

learning curve: while a graphical program may allow you to get started right away (at least in theory), ABC requires that you take your time and study the syntax before you can get started;

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

limitations: ABC can't do everything. Currently, it can't deal with some types of music, such as Gregorian chant 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 in action. The ABC home page has many links to ABC collections; on the ABC PLUS page you will find more complex choral scores.

1.2 Getting Started

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

1. using an editor, write the tune using the ABC syntax;
2. convert the tune using `abcm2ps`, creating a PostScript file;
3. view the PostScript file with Ghostscript or other viewer;
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!

1.2.1 Installing the Programs

Relevant web pages:

- 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 GSview:
<http://pages.cs.wisc.edu/~ghost>

A rather old but convenient version is Ghostscript 5.50, packaged with GSview 2.7. Unlike newer releases of GSview, this release has no nag screen, and is especially suitable for use with JEDABC. Get the installer `gsv27550.exe` from:

<ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/aladdin/gs550/>

European mirror:

<ftp://mirror.switch.ch/mirror/ghost/aladdin/gs550/>

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/GPL/>

and the latest GSview from:

<http://mirror.cs.wisc.edu/pub/mirrors/ghost/ghostgum/>

- ABCexplorer:
<http://abc.stalikez.info/abcex.php>
- EasyABC:
<http://www.nilsliberg.se/ksp/easyabc/>
- JEDABC:
<http://abcplus.sourceforge.net/#JedABC>
- runabc:
<http://ifdo.pugmarks.com/~seymour/runabc/runabc.html>
- The Folkinfo converter:
<http://www.folkinfo.org/songs/abcconvert.php>

Ready-to-run packages for Windows, GNU/Linux, and Mac OS X are available from the ABC PLUS web site. Besides, all GNU/Linux distributions include Ghostscript and the `gv`, `evince`, and `okular` viewers.

I assume that GNU/Linux and Mac OS X users will be able to install the programs. On Windows, the easiest and highly recommended solution is installing ABCexplorer, which includes all relevant programs; alternatively, download `abcm2ps` and `abc2midi`, extract the `.exe` files from the archives, then copy them to `C:\WINDOWS`. Ghostscript and GSview are installed like any standard Windows applications.

1.2.2 ABC in a Nutshell

ABC music is written in text files with extension `.abc` or `.abp`. The extension is not obligatory, but highly recommended.

ABC 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:`, `K:`, `T:`;

- 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, which is written in an ABC *file*, consists of two parts: a *header* and a *body*. 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 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 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 4.1.

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 bracket* {;
- Alt-125 to get a *closed curly bracket* };
- Alt-091 to get an *open square bracket* [;
- Alt-092 to get a *closed square bracket*].

1.2.3 Our First Score

When the programs are installed, we are ready to write our first score. As our first example, we shall write the C major scale and study the source carefully.

In the following, I'll describe the very basic commands: the command-line interface, plus an editor. You will see that it's not that difficult after all. Anyhow, remember that you can use the Folkinfo converter.

There's a very important detail to stress beforehand. Whenever you write an ABC file, you'll have to choose a file name. Any file name will do, *but*:



Very important!

do not use file names containing spaces!

That is: instead of, say, `My music file.abc`, you must write `My-music-file.abc` or `My-MusicFile.abc`. It's very important, because `abcm2ps` cannot easily use filenames containing spaces. Besides, file names containing spaces are not web-friendly either.

Ok? Good. Let's proceed. Commands you must enter are printed in **boldface**.

Windows. Start the `cmd` program. (Windows XP: from Start/Run..., insert `cmd`, then click on “Ok”. Windows 7: start the menu then insert `cmd` in “Search Programs and Files”.)

A black window will pop up showing a line that reads something like:

```
C:\Users\Guido>_
```

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

```
C:\Users\Guido>md abcmusic
```

and press the Enter key. This command creates a directory (folder), called `abcmusic`, where we will save our ABC files. Now, let's enter (double click) the folder:

```
C:\Users\Guido>cd abcmusic
C:\Users\Guido\abcmusic>_
```

Note that the command prompt changed to confirm that you've moved to that folder. Move to the `abcmusic` folder before writing tunes.

Let's now run the notepad editor. In our example, we'll use the file name `scale1.abc`.

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

Click 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:\Users\Guido\abcmusic>abcm2ps -c -O= scale1.abc
abcm2ps-5.9.25 (November 2, 2011)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 17899 bytes)
C:\Users\Guido\abcmusic>_
```

To display the PostScript file, launch GSview and open the `scale1.ps` file that you'll find in `C:\DocumentsandSettings\user\abcmusic`. The score will look like this:



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

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:\Users\user\abcmusic>abcm2ps -c -O= scale1.abc
abcm2ps-5.9.25 (November 2, 2011)
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, 17877 bytes)
C:\Users\user\abcmusic>_
```

Fix the error in the source, save it and convert it again.

GNU/Linux and Mac OS X. Open a terminal window (Ubuntu: Applications/Accessories; Mac OS X: Applications/Utilities). The first time, and only the first time you use the command prompt, type this command:

```
$ mkdir abcmusic
```

and press the Enter key. This command creates a directory (folder), called `abcmusic`, where we will save our ABC files. Now, let's enter (double click) the folder:

```
~$ cd abcmusic
~/abcmusic$ _
```

The command prompt should change to confirm that you've moved to that folder. Move to the `abcmusic` folder before writing tunes.

Now let's start an editor. In GNU/Linux systems, it's likely to be `gedit` or `kate` (if it's not, please find out). In Mac OS X, it's simply called `edit`. The file name is `scale1.abc`.

```
~/abcmusic$ gedit scale1.abc &
```

Don't forget the ampersand & at the end of the line. 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`:

```
~/abcmusic$ abcm2ps -c -O= scale1.abc
abcm2ps-5.9.25 (November 2, 2011)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 17899 bytes)
~/abcmusic$ _
```

To display the PostScript output, GNU/Linux users should use the `gv` or `evince` or `okular` viewer. Mac OS X has a Preview utility that will automatically open `scale1.ps` and convert it into PDF format.


```
~/abcmusic$ evince scale1.ps &
```

The score will look like this:



Finally, to convert the file into PDF type this command:

```
~/abcmusic$ ps2pdf scale1.ps
~/abcmusic$ _
```

The file `scale1.pdf` will be silently created.

Warning

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:

```
~/abcmusic$ abcm2ps -c -O= scale1.abc
abcm2ps-5.9.25 (November 2, 2011)
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, 17877 bytes)
~/abcmusic$ _
```

Fix the error in the source, save it and convert it again.

Let us now examine what we wrote in the source. It starts with two header fields: `X:`, index, and `K:`, key; both upper-case. 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.

Fields may 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 (ut) re mi fa sol la si (ti)”; if you live in such a country, you may want to consult Table 1.1 that compares notes written in English and in Italian notation. I call it “Italian notation” since it was introduced by medieval Italian composer Guido d’Arezzo.

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

Table 1.1: 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 starts the body of the tune, containing 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`, by default, 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.

Note

If you use “do re mi...”, the main hurdle is getting used to “C D E...” A trick I used was memorizing the notes as “doC”, “reD”, “miE”, “faF”, “solG”, “laA”, “siB” and the reverse.

Homework time. Try and write some ABC 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.



Chapter 2

Melody

2.1 Notes and Symbols

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

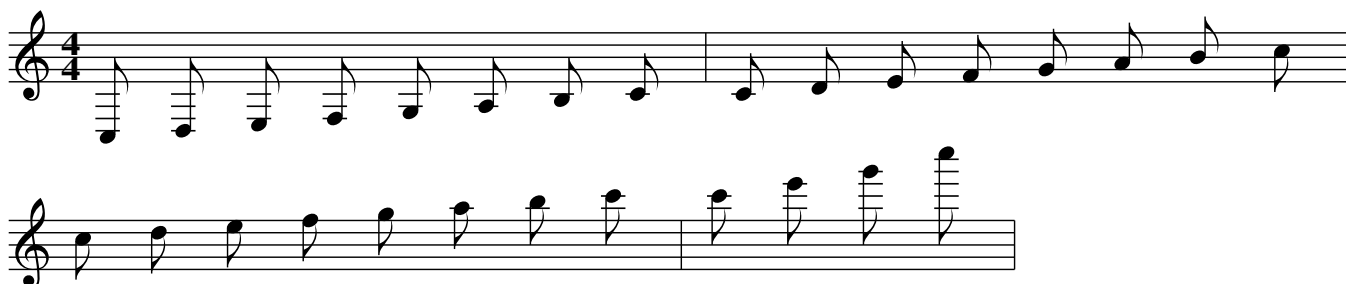
Note

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

2.1.1 Pitch: **A–G a–g , '**

The following source (`scale2.abc`) shows how to obtain notes under and above the staff, i.e. notes that sit on ledger lines; 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 2.2.1. `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' e' g' 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.¹

Note another important detail: we wrote two lines of ABC 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 2.1.13.

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.

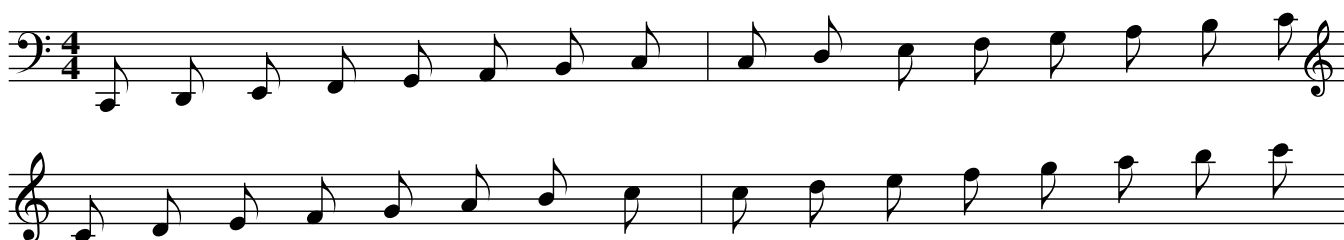
We'll now reformat the file using `-c`:

```
$ abcm2ps -O= -c scale2.abc
```



The bass clef is automatically set when we start a piece of music with low notes (F, and below). The treble clef is then set by `abcm2ps` for higher notes:

```
X: 1
M: 4/4
L: 1/8
K: 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 | c d e f g a b c' |
```



2.1.2 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.

¹Experts will notice some resemblance to the Helmholtz pitch notation, shifted two octaves.


```

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///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/yE/ F/yF/yF/yF/ yyy|xxxG|

```



`y` can be followed by a width in *points*; default is 20 points. This unit of measurement is widely used in typography; a “point” (PostScript point) is $\approx 0.353\text{mm}$.

Invisible rests are often used for transcribing piano music; examples will be shown in Section 3.1.2.

2.1.4 Accidentals: `^` `_` `=`

Sharp \sharp is denoted by a `^` before the note, flat \flat by `_` (underscore), and natural \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 $\sharp\sharp$ by `^^` and double flat $\flat\flat$ by `==`.

Quarter tone accidentals are also available. That is, $1/4$ and $3/4$ sharps and flats can be obtained adding `/` after `^` or `_`, while $3/4$ are obtained adding `3/2`:

```
X: 1
M: 4/4
L: 1/8
K: C
G ^/G ^G ^3/2G A4|A _/A _A _3/2A G4|G3 ^G A4|A3 _A G4|]
```



Note

Unlike normal accidentals, microtonal accidentals don't propagate across a whole measure. This is an important thing to remember when you make a MIDI file, to avoid unexpected results.

2.1.5 Dotted Notes: < >

Notes followed by a dot are printed in two ways. First, by 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/|
```



Note

Dotted notes are rendered by abc2midi in a peculiar way; please see Section 5.1.11.

2.1.6 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|]
```



Note

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: MIDI output (see Section 5.1) 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|]
```



2.1.7 Triplets: (n

Triplets, quadruplets, etc. are coded with an open left parenthesis, immediately followed by the number of notes, 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 $\langle n \rangle$ (see below)
- `(6` : 6 notes in the time of 2

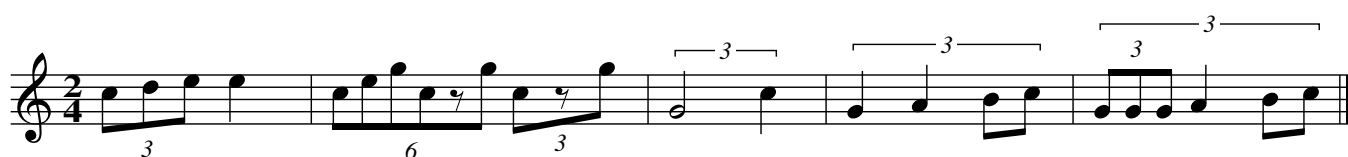
- (7 : 7 notes in the time of $\langle n \rangle$)
- (8 : 8 notes in the time of 3)
- (9 : 9 notes in the time of $\langle n \rangle$)

The notes in a tuplet must have the same length. If the tune metre is compound, that is 6/8, 9/8, 12/8 etc., $\langle n \rangle$ is 3; otherwise, it is 2.

More complex tuplets are coded using the extended notation $(\langle n \rangle : \langle t \rangle : \langle x \rangle)$, which means: put $\langle n \rangle$ notes into the time of $\langle t \rangle$ for the next $\langle x \rangle$ 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. To make things more interesting, nested tuplets are also possible! In fact, one of the notes can be replaced by a tuplet:

```
X: 1
M: 2/4
L: 1/8
K: C
(3cde e2 | (6cegczg (3czg |
(3:2:2G4c2 | (3:2:4G2A2Bc | (3:2:6(3GGA2Bc |]
```



To fine-grain the printing of tuplet indications, please see Section 4.5.5.

⚠ Warning

abcMIDI does not support nested tuplets, so you will not be able to hear them.

2.1.8 Chords: []

Chords are written enclosing the notes in square brackets; spaces between the brackets 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 |
```



⚠ Warning

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 3.1.

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 programs do not support this feature.

2.1.9 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. `<n> -` characters separated from the previous syllable skip `<n>` 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!

```

tan - ti au - gu - ri a te, _ tan - ti au - gu - ri a

te, tan - ti au - gu - ri fe - li - ci, tan - ti au - gu - ri a te!

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

Note

Believe or not, “Happy Birthday” is a copyrighted song—even if it existed over a century ago. Have a look on Wikipedia. You might want to write to the copyright holder to tell them what you think about it.

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.

```

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

```



Accidentals can be written in all strings, including `w:` lines, using special codes: `\201` prints \sharp , `\202` prints \flat , and `\203` prints \natural .



Very important!

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.

2.1.10 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. These sets of characters are called *backslash sequences*. It is easier to do than to explain: please see Table 2.1. Note the difference between the acute `'` (ASCII 39) and grave ``` (ASCII 96) accents.

A complete table of symbols is shown in Appendix A.4.

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

Table 2.1: How to obtain characters of foreign languages.

Never Type Accented Letters!

Let's suppose that your computer keyboard, like mine, provides accented letters. You may be tempted not to use backslash sequences. After all, the letters you need are there. Typing them directly is handy, so why bother?

Below, I'll tell you the reason why you should bother. But let me start with a strong statement:



Very important!

never ever write accented letters directly, *always* use backslash sequences instead!

The reason is somewhat technical. Different computers employ different character encodings: ISO 8859-1, UTF-8, and many others. A .abc file encoded in, say, UTF-8, could be unreadable on computers that use an ISO encoding; or vice versa.

More importantly, `abcm2ps` only understands ISO 8859. Let's suppose that the following source is encoded in UTF-8. Apparently it's OK, but look what happens when you feed it to `abcm2ps`:

```
X: 1
T: Wrong letters: à è ì ò ù
L: 1/4
K: C
CDEF|GABc|CEGc|cGEC|
```

Wrong letters: Ã Ä Å Æ Å² Å¹



(The T: field will be explained later on.) As you can see, the title is completely wrong! Now, a question for you: are you sure that your editor or ABC application is properly configured to encode its files in ISO 8859?

Unless you're a geek and you know what you're doing, the obvious conclusion is simple: to obtain foreign characters, always use backslash sequences. More information will be given in Appendix A.4.

Note

The latest stable version of `abcm2ps` supports UTF-8, so the problems with accented letters are solved. But beware: what encoding does your editor use? Can you change it?

2.1.11 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 brackets before the main note. To add a slash to a grace note (*acciaccatura*), the open curly bracket 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 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.³

³`abcm2ps` has special support for Great Highland Bagpipe music.

```
X: 1
L: 1/4
K: C
~c2 /dc c2d2c|d/c/d/c eded fefe f|
c/gfed/e/f/ f/e/gfedcd/c/|c G E cBAGFEDC|
```



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.

2.1.12 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.

2.1.13 Avoiding Line Breaks: \

Usually, $\langle n \rangle$ measures in a source line produce $\langle n \rangle$ 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. Do not use the `-c` switch:

```

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!

```

Brother John

Traditional

Are you slee - ping, Are you slee - ping, Bro - ther John! Bro - ther John!

Mor - ning bells are rin - - ging, Mor - ning bells are rin - - ging,

ding ding dong, ding ding dong!

2.1.14 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|

```

2.2 Music Properties

2.2.1 Key signatures and Clefs: κ :

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

Key Signatures

κ : 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 \sharp and `b` for \flat ; e.g. `Bb`, or `F#m`.

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 2.2 shows the keys that correspond to a specified number of sharps or flats.

Peculiar key signatures are κ :HP and κ :Hp, which are used for Great Highland Bagpipe music. κ :Hp marks the staff with $F\sharp$, $C\sharp$ and $G\flat$; both force note beams to go downwards.

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

Table 2.2: Correspondence between the key and the number of sharps or flats.

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 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 2.3 lists them all.

Explicit accidentals can also be specified by appending them to the key signature. For example, `K:D exp =c ^g` would set the key of D major but mark every `C` as natural, and every `G` as sharp.⁴ 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`.

Since many people don’t have formal music training, I suggest that a comment be added after the κ : field to indicate the number of accidentals: `K: AMix % 2 sharps`

⁴The keyword `exp` may be omitted, but `abc2midi` needs it.

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	B#Lyd	E#Loc
5 sharps	B	G#m	F#Mix	C#Dor	D#Phr	E#Lyd	A#Loc
4 sharps	E	C#m	BMix	F#Dor	G#Phr	A#Lyd	D#Loc
3 sharps	A	F#m	EMix	BDor	C#Phr	D#Lyd	G#Loc
2 sharps	D	Bm	AMix	EDor	F#Phr	G#Lyd	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 2.3: Modal scales.

All Seven Clefs

As we saw in Section 2.1.1, the clef is automatically set 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>] [staffscale=<number>]
```

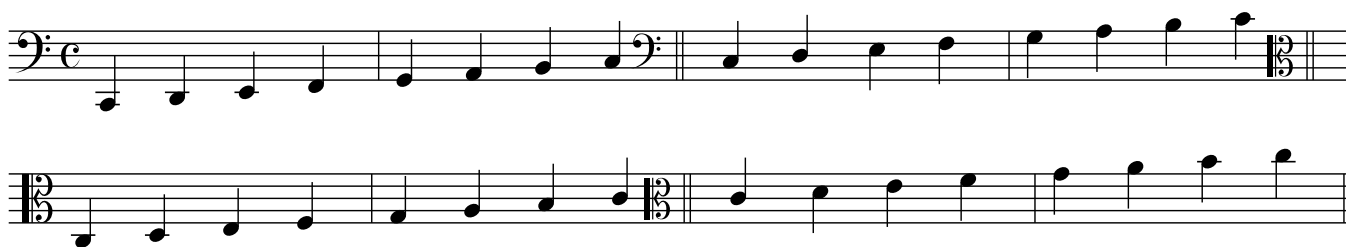
where:

- `clef=` may be omitted before the clef type when it's not none;
- clef types can be:
 - a note pitch: `G` or `g` (treble clef), `C` or `c` (alto clef), `F` or `f` (bass clef) are allowed. The specified note will be positioned on the relevant line, as explained below;
 - 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 on which the clef is drawn;
- `+8` and `-8` draws “8” above or below the staff;
- `middle=<pitch>` (or `m=<pitch>`) specifies the note on the third (middle) line of the staff;

The `clef=` field is very useful if you don't want to write notes in bass or alto clefs with all those commas. When you specify for instance `K: C clef=f`, the `f` note will be placed on the fourth line; with `K: C clef=c`, the `c` note will be placed on the third line.

The following example shows how the very same notes can be set in different clefs:

```
X: 1
M: C
L: 1/4
K: C clef=f
%
CDEF|GABc||
K: C clef=F
CDEF|GABc||
K: C clef=c
CDEF|GABc||
K: C clef=C
CDEF|GABc||
```



Bass and Alto Clefs Compatibility Issues

Previous versions of `abcm2ps` dealt with notes in bass and alto clefs in a peculiar way. `abcm2ps` could automatically transpose the music one or two octaves; besides, in bass clef the two notes `c` and `C`, could be equivalent, depending on the context.

This was quite confusing, and fortunately this scheme is now gone. However, problems may occur when you come across files that were written for the old scheme. You could see that some notes are printed one or two octaves higher than expected.

There are three ways to deal with these files:

- the long way: manually change every note to transpose it two octave lower, i.e. change `c` to `C`, ;
- the quick and dirty way: include the following command at the beginning of the file:
`%%abcm2pscompat 1`
- the quick and correct way: specify the key using the appropriate `clef=` field.

I'll throw in a real-life example. A viola player asked me to transpose some Irish music from treble to alto clef. The right way to do it was changing the `K:` field from `K: D` to `K: D alto`, and adding the `%%abcm2pscompat 1` line before the tunes.

Note

The `%%abc2pscompat` 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 can read the same source just ignoring the meta-comments.

We shall examine meta-comments in detail in Section 4.1.

2.2.2 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)` or `M:7/8 (3+2+2)`. There can be no space between digits and the `+` character;
- 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”). Ancient metre indications are supported: `M: c` `M: c.` `M: o` `M: o.`;
- as an explicit measure duration: `M:C|=2/1`;
- if there is no metre, use `M:none`.

Needless to say, the metre can change in the middle of a tune. 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 |
```

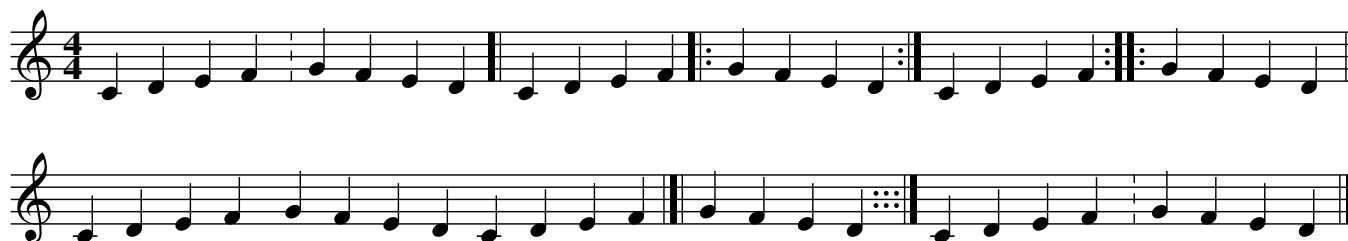


The recommended order of fields related to note length is `M:`, `L:`, and `Q:`.

2.2.3 Bars and Repeats: | / : []

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

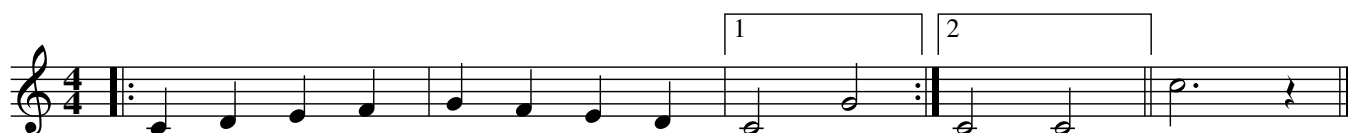
```
X: 1
L: 1/4
K: C
CDEF .| GFED [| CDEF |: GFED :|CDEF :: GFED || CDEF [|] GFED
CDEF [|] GFED :::|] CDEF : GFED [|]
```



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

To indicate that a section has two (or more) different repeats, use the symbols `[1, [2, ...` as in the following example. When repeats symbols are close to a bar, they can be shortened using `|1, |2, ...`. The end of the last repeat is set with `||`, or `|]` if at the end of the tune.

```
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 |]
```



Most commonly, the start repeat bar `|:` is left out. This is not a problem in printed music, but `abcmIDI` is a bit more restrictive and expects to find it. In Section 2.2.8 we will see how to include `|:` in the source, but hide it in printed output.

⚠ Warning

Unfortunately, repeats are a field where incompatibilities between `abcm2ps` and `abcmIDI` emerge:

- to repeat a run of notes three times, `|:::|` will not work. A simple workaround is: `|:|1-3 notes :|`.
- text indications in repeat bars (as "last time" above) are not recognised.

2.2.4 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: F
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!
```

Happy Birthday (Tanti auguri a te)

traditional
(transcription Guido Gonzato)

Allegro ♩ = 120

Hap - py birth - day to you, ___ Hap - py birth - day to

you, ___ hap - py birth - day dear fel - low, hap - py birth - day to you!

The text indication in the *Q*: field (“Allegro” in our example) may be omitted. In Section 4.1 we will learn how to change the title fonts.

2.2.5 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. In the header, this field specifies the order in which parts should be played; in the body, it marks the beginning of each part.

Part names must be upper-case letters, **A** . . . **Z**.

```
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|]
```

Song in three parts

AABBC



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:ABABABCD-CDCDABABABCD`. Try and count carefully!



Very important!

`P:` cannot be inserted within repeats (i.e. `|:...:|`), because `abc2midi` gets confused and produces wrong MIDI output.

2.2.6 Accompaniment Chords: " "

In many songbooks, accompaniment chords (say, for the guitar) are notated as “A”, “C7”, “Dm”, “F#” etc. above the staff. 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;⁵) the accidental is indicated with `#` for \sharp , `b` for \flat , or `=` for natural; the chord type is one of those listed in Table 2.5.

A slash `/` followed by a note `A...G` denotes either an optional bass note, or a *chord inversion*. Spaces between the chord and the following note are not allowed. Chord inversion and optional bass notes are rendered by `abc2midi`; please refer to Section 5.1.7 for explanations.

```
X: 1
T: Happy Birthday
T: (version with chords)
C: traditional
M: 3/4
Q: "Allegro" 1/4 = 120 % tempo
```

⁵lower case is possible, but not recommended; `abcMIDI` uses it

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

Table 2.5: Types of accompaniment chords.

```

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!

```

Happy Birthday (version with chords)

traditional

Allegro ♩ = 120

Chords: F, C, C, F, Bb, F, C, F

Lyrics: Hap - py birth - day to you, Hap - py birth - day to you, hap - py birth - day dear fel - low, hap - py birth - day to you!



Very important!

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 4.5.2.

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; C7" CCCC | ]
```



Although `\n` is allowed, one should avoid using it and always employ `;` instead, since `abc-2midi` only supports the latter.

2.2.7 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 must 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.

Very important!

A common mistake is writing text annotations leaving out one of the `^_<>@` characters. This “annotation” will be interpreted as an accompaniment chord, which will be almost certainly malformed and will wreak havoc in both printed and MIDI output.

To include accidentals in text annotations, use `\#`, `\b`, and `\=`; note the leading `\`. Multiple annotations are notated like multiple chords, as seen in Section 2.2.6.

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|
```

Dolcemente ♩ = 60

piano above left anywhere

below right

Figured Bass

Most symbols of figured bass notation can be written using `s:` lines, containing text annotations instead of symbols:

```
X: 1
T: Figured Bass
M: none
L: 1/4
K: C bass
%
C,4 C,4 C,4 C,4 C,4 C,4 C,4 C,4 C,4]
s: "_6;4" "_#" "_6;" "_6+" "_6-" "_-" "_\\6" "_/6" "_6 4"
```

Figured Bass

6 4 # $\flat 6$ 6+ 6- - \6 /6 6 4

Please note that all annotations have a leading `_` (underscore) character, so they are printed below the staff.

2.2.8 Ornaments: *!symbol!*

Ornaments are notated using the general form `!symbol!`: the ornament name (*ff*, *ppp*, *cresc*...) enclosed in exclamation marks. Ornaments, also known as “embellishments” or “decorations”, are written immediately before the note; two or more ornaments per note are possible. Most ornaments can also be applied to grace notes or measure bars.

```
X: 1
M: 4/4
L: 1/4
K: C
%
!p!GGG!>!!f!G!|!segno!GG!trill!G2!|!coda!GGG!fine!G|
```

p *f* *tr* FINE

As shown above, some symbols are printed above the staff by default, while others are printed under the staff. Forcing ornaments under or above the staff is possible; please refer to Section 4.1.

Supported ornaments are listed in Figure 2.1. You will probably notice that some symbols you need are missing. Don't worry: new symbols can be added, as we will see in Section 4.5.

Dynamics

!<(! !<(! !>(! !>(! !crescendo(! !crescendo(! !diminuendo(! !diminuendo(!

Tuning parts

!thumb! !D.C.! !D.S.! !coda! !fine! !segno!

Beams & measure bars

!beambr1! !beambr2! !beamon! !rbstop!

Others

!longphrase! !mediumphrase! !shortphrase! !/! !//! !///! !+! !>! !accent! !arpeggio! !breath! !dot! !downbow!

!upbow! !emphasis! !fermata! !invertedfermata! !gmark! !invisible! !mordent! !lowermordent!

!uppermordent! !open! !plus! !pralltriller! !roll! !tenuto! !trem1! !trem2! !trem3! !trem4!

!trill! !trill(! !trill)! !turn! !turnx! !invertedturn! !invertedturnx! !wedge!

Fingerings

!0! !1! !2! !3! !4! !5! !slide! !snap!

Figure 2.1: Standard expression symbols.

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

Rules for matching notes and symbols are those explained in Section 2.1.9. `s :` lines and note-linked symbols can be used at the same time.

```
X: 1
L: 1/4
K: C
C/D/ E/F/ G/A/ B/c/|c/B/ A/G/ F/E/ D/C/|e2!fermata!c2|z4||
s: !>! !>! * !ff! !p! !mp! * !ff! !>! !>! !>! !ff! * * * !ff! |
```



While most symbols just print something over a note and are self-explanatory, some of them act on multiple notes or other printed elements:

- symbols that include an open parenthesis (in their definition start a multi-note ornament, which is ended by the corresponding symbol with a closed parenthesis). For example, !<(! starts a crescendo hairpin, !<)! ends it. The same holds for !>(!, !>)!, and their long-name counterparts;
- !beamb1! and !beamb2! act on notes connected by beams, leaving only 1 or 2 beams;
- !beamon! prevents a beam from being broken across measures;
- !invisible! makes the next note or measure bar invisible;
- finally, !xstem! will be explained in Section 4.1.2.

The following tune shows a three-times repeat with a hidden start repeat bar:

```
X: 1
M: 4/4
L: 1/4
K: C
GGGG !invisible!|: |1-3 CDEF |G!rbstop!ABc:|
```



2.2.9 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 y, 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!:

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

Table 2.6: Standard abbreviations for common symbols.

```
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 2.6.

2.2.10 Information Fields

In Section 1.2.3 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)

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 4.1.1.

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 tune was found. Example: `S:from John Chambers' site`

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

All of these fields can be printed in titles by means of the `%%titleformat` command, or after the tunes by using `%%writehistory` and `%%info name`.



Chapter 3

Harmony

3.1 Polyphony in ABC

SO far, we only have seen melodies: music written for a single voice or instrument. This is all that ABC 1.6 is 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.

3.1.1 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 (e.g. woodwinds) or more than one (piano, organ). A set of staves related to instruments that play together in the piece is called a *system*.

Note

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

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|

```

Brother John

Traditional

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> [clef=] [clef type] [name=] [sname=] [merge] [stem=] [up | down | auto]
[gstem=] [up | down | auto] [dyn=] [up | down | auto] [lyrics=] [up | down | auto] [gchord=]
[up | down | auto] [scale=] [staffscale=] [stafflines=]

```

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 2.2.1.
- `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.
- `stem= up, down, or auto` forces the note stem direction. The `stem=` keyword may be omitted.
- `gstem=<up>, down or auto` forces the grace note stem direction.
- `dyn=<up>, down or auto` forces the placement of dynamic marks.
- `lyrics=<up>, down or auto` forces the placement of lyrics lines.

- `gchord=<up>` or `down` forces the placement of accompaniment chords.
- `scale=<n>` sets the scale of the voice. Default is 1, maximum value is 2, minimum is 0.5.
- `staffscale=<n>` sets the scale of the voice and the associated staff. Default is 1, maximum value is 3, minimum is 0.5.
- `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|
```

Brother John

Traditional

The musical score for 'Brother John' is presented in two systems. The first system consists of two staves: 'Soprano' and 'Contralto'. The second system consists of two staves: 'S' and 'VB'. The Soprano part (V: 1) features a melody of eighth and quarter notes. The Contralto part (V: 2) features a bass line with many rests. The key signature has three sharps (F#, C#, G#).

Fancy staves can be obtained using `staffscale` and `stafflines`. Percussion notes (cross-shaped note heads) are obtained using sharps:

```
X: 1
T: Special Staves
M: 4/4
```

```

L: 1/4
V: 1 stafflines=6 staffscale=0.7
V: 2 stafflines=4 scale=1.2
V: 3 stafflines=1 staffscale=1.4 perc
K: C
%
[V:1] "^6 staff lines, staffscale=0.7"CDEF|GABc|cBAG|FEDC|
[V:2] "^4 staff lines, staffscale=1, scale=1.2"CCCC|GGGG|EEEE|G2z2|
[V:3] "^1 staff line, staffscale=1.4"^c/^c/^c/^c/ ^c/^c/^c/^c/|z4| \
      ^c/^c/^c/^c/ ^c/^c/^c/^c/|cccc|

```

Special Staves

6 staff lines, staffscale=0.7

4 staff lines, staffscale=1, scale=1.2

1 staff line, staffscale=1.4

3.1.2 Positioning Voices: `%%staves` and `%%score`

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 either the `%%staves` or `%%score` command. They are basically equivalent, with one exception I shall explain later on.

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 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 brackets, their staves will be joined by a bracket. 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]`

The command `%%score` accepts the same parameters as `%%staves`, but measure bar indications work the opposite way: by default, they *do not* cross the staves. Therefore, `%%staves [S|A|T|B]` is the same as `%%score [SATB]`, while `%%staves [SATB]` is the same as `%%score [S|A|T|B]`.

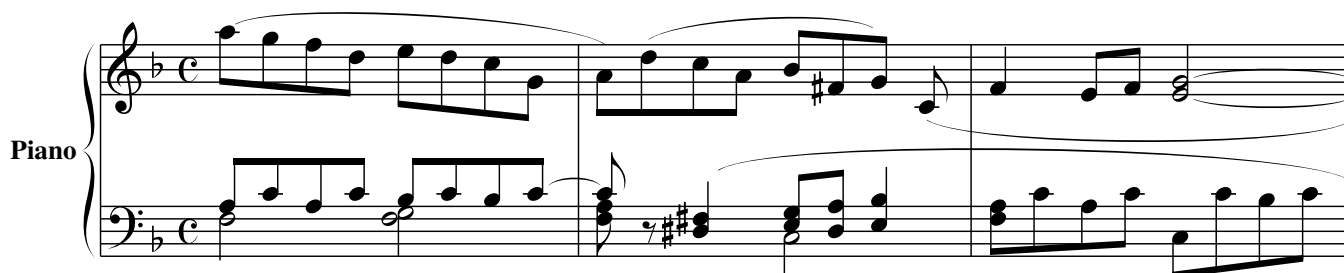
When two voices are printed on the same staff, the stem direction indicates the first voice (up) or the second (down).

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 2.1.3.

```
X: 1
T: Studio 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 |
[V: LH1] A,CA,C B,CB,C-|Cz ([^D,2^F,2][E,G,][D,A,][E,2B,2]|
[V: LH2] F,4 [F,4G,4] |[F,A,] x x2 C,4 |
%
[V: RH1] F2 EF [E4G4]- |
[V: LH1] [F,A,]CA,C C,CB,C|
[V: LH2] x4 x4 |
```

Studio Op. 10 – N. 3

F. Chopin



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] F,2F,A, | A,F,A,2- | A,4 | B,4 |
[V: MS2] D,4- | D,4- | D,4- | D,4 |
[V: B] z4 | z4 | D,2D,2 | D,2D,2 |
w: A- ve, A- ve,
[V: T] z4 | z4 | A,2A,2 | B,2B,2 |
[V: A] z4 | z4 | F2F2 | E2E2 |
[V: S] z4 | z4 | A2 (dF) | (A^G)=G2 |
w: A- ve, * A - ve,

```

Ave Verum

W. A. Mozart

Adagio

The musical score for 'Ave Verum' by W. A. Mozart is presented. It features five staves: Soprano, Alto, Tenore, Basso, and Organo. The key signature is D major (two sharps) and the time signature is 4/4. The tempo is marked 'Adagio'. The vocal parts (Soprano, Alto, Tenore, Basso) enter in the third measure with the lyrics 'A - - ve,'. The organ part provides harmonic support throughout. The score is written in reverse order for the voices, as noted in the text below.

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.

Note

The %%staves command is a strong point of the ABC 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:

```
%%topmargin 2cm
%%scale 0.7
%%leftmargin 1cm
%%rightmargin 1cm
%%squarebreve 1
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 C,4 F,2G,2F,2LC,2 D,2E,2
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] LF,4 z2 C,2F,2G,2F,2LD,2 F,2E,2D,2LB,,2 C,8 F,16[]
```

w: son Ky- ri - e e- lei ----son.

Missa Brevis

Andrea Gabrieli (1510? – 1586)

Kyrie

The musical score for the Kyrie from Missa Brevis by Andrea Gabrieli is presented in four staves. The lyrics are: Ky - ri - e e - lei - son. The score shows a complex polyphonic texture with multiple entries of the text. The notation includes various musical symbols such as notes, rests, and bar lines, along with the lyrics 'Ky - ri - e e - lei - son' repeated throughout.

3.1.3 Voice Splitting: &

In some pieces of music, a voice splits in two in some measures only. To avoid introducing an additional and 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|
```

The musical notation for the voice splitting example is shown in 4/4 time. The notes are C, E, G, and C, with the & symbol indicating a split between the first and second notes.

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

The above piece can be equivalently written as:

```

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 |

```

which is more verbose.

3.1.4 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 voices that are actually playing. Here is an example “Riu riu chiu”, a well-known 16th century villancico. The result is shown in Figure 3.1.

The `%%vskip 0` command is used to make a small vertical break between parts. `%%vskip` always inserts a break, even if one specifies 0.

Please note that strange `\241` in the title. It is the octal code of the `í` character in the ISO 8859-1 (Latin1) character set. We will cover this topic in Appendix A.4.

```

%%scale 0.6
%%barsperstaff 6
X: 1
T: Riu, riu, chiu, \241lla 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, \241lla guar-da ri-be-ra!
[V: 3] AAGA      |F2EG2GEF|D2D2|
w: Díos guar-dó el lo-bo de nue-stra cor-de-ra,
[V: 3] AAGA      |F2EG2GEF|D2D2|
w: Díos guar-dó el lo-bo de nue-stra cor-de-ra.
% CHANGE OF SYSTEM: everyone
%%vskip 0
%%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, \2411a guar-da ri-be-ra!
[V: 2] FFEC|D2EF2EDD |C2C2z2|
[V: 3] cccG|A2AA2ADD |E2E2z2|
w: Ri-u, ri-u, chi-u, \2411a guar-da ri-be-ra!
[V: 4] F,F,C,F, |D,2A,,D,2C,_B,,B,, |A,,2A,,2z2|
%
[V: 1] z4      |AAGA      |F2EF2FEE|D2D2z2|
w: Diós guar-dó el lo-bo de nue-stra cor-de-ra,
[V: 2] z2EE     |DCEC      |D2CD2DCC|D2D2z2|
w: Diós guar-dó el lob', el lo-bo de nue-stra cor-de-ra,
[V: 3] ccBc     |A2BA      |A2AA2AAA|A2A2z2|
w: Diós guar-dó el lo-bo, el lo-bo de nue-stra cor-de-ra,
[V: 4] A,A,G,A, |F,2E,F, |D,2A,,D,2D,A,,A,, |D,2D,2z2|
%
[V: 1] z4      |AAGA      |F2ED2DCC      |D2D2      |
w: Diós guar-dó el lo-bo de nue-stra cor-de-ra.
[V: 2] z2EE     |DCEC      |D2CA,2A,A,A, |A,2A,2|
w: Diós guar-dó el lob', el lo-bo de nue-stra cor-de-ra.
[V: 3] ccBc     |A2BA      |A2AF2FEE      |D2D2      |
w: Diós guar-dó el lo-bo, el lo-bo de nue-stra cor-de-ra.
[V: 4] A,A,G,A, |F,2E,F, |D,2A,,D,2D,A,,A,,      |D,2D,2      |
% CHANGE OF SYSTEM: men only
%%vskip 0
%%staves 3
[V: 3] AAGA|F2EG2GEF|D4|AAGA|
w: El lo-bo ra-bio-so la qui-so mor-der, Mas Diós 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.

```

There's an alternate way to write pieces with a variable number of voices. The command `%%staffnonote 0` hides the voices that don't contain notes:

```

X: 1
M: 4/4
L: 1/4
%%staves [S A T]
V: S name="Soprano" sname="S"
V: A name="Alto"      sname="A"
V: T name="Tenor"     sname="T"
K: G
%%staffnonote 0
[V: S] GABc|GABc|GABc|B3z|
[V: A] GGGG|BBBB|GGGG|D3z|
[V: T] GDGD|GDGD|GGDD|G3z|
%
[V: S] GABc|dcBA|GABc|d3z|
[V: A] z4|z4|z4|z4|

```


Riu, riu, chiu, ¡la guarda ribera!

Villancico (Spain, XVth century)

$\text{♩} = 240$

Tenor
Bass

8 Ri - u, ri - u, chi - u, ¡la guar-da ri - be - ra! Díos guar-dó el lo - bo de nue-stra cor - de - ra,

8 Díos guar - dó el lo - - bo de nue - stra cor - - de - - ra.

S
A
T
B

Ri - u, ri - u, chi - u, ¡la guar-da ri - be - ra! Díos guar-dó el lo - bo de nue-stra cor -

Díos guar - dó el lob', el lo - bo de nue-stra cor -

8 Ri - u, ri - u, chi - u, ¡la guar-da ri - be - ra! Díos guar-dó el lo - - bo, el lo - bo de nue-stra cor -

de - ra, Díos guar - dó el lo - bo de nue - stra cor - de - ra.

de - ra, Díos guar - dó el lob', el lo - bo de nue - stra cor - de - ra.

8 de - ra, Díos guar - dó el lo - - bo, el lo - bo de nue - stra cor - de - ra.

El lo - bo ra - bio - so la qui - so mor - der, Mas Díos po - de - ro - so la su - po de - fen - der;

8 qui so - le ha - ce que no pu - die - sce pe - car: ni aun o - ri - gi - nal e - sta Vir - gen no tu - vie - ra.

Figure 3.1: A piece where the system changes twice.

```

[V: T] z4 | z4 | z4 | z4 |
%
[V: S] GABc | GABc | GABc | B3z |
[V: A] GGGG | BBBB | GGGG | D3z |
[V: T] GDGD | GDGD | GGDD | G3z |

```

The musical score consists of four systems of staves. The first system has three staves labeled Soprano, Alto, and Tenor. The second system has one staff labeled S. The third system has three staves labeled S, A, and T. The fourth system has three staves labeled S, A, and T. All staves are in 4/4 time with a key signature of one sharp (F#). The notes are as follows:

System	Voice	Notes (Measure 1)	Notes (Measure 2)	Notes (Measure 3)	Notes (Measure 4)
System 1	Soprano	G4	A4	B4	C5
	Alto	E4	F4	G4	A4
	Tenor	D4	E4	F4	G4
System 2	S	G4	A4	B4	C5
System 3	S	G4	A4	B4	C5
	A	E4	F4	G4	A4
	T	D4	E4	F4	G4
System 4	S	G4	A4	B4	C5
	A	E4	F4	G4	A4
	T	D4	E4	F4	G4

As you can see, the source is greatly simplified. The disadvantage of this method is that you must write the source specifying an exact number of bars per line. (e.g. you can't use the `-c` switch).

♪ ♪ ♪ ♪ ♪ ♪ ♪ ♪

Chapter 4

Page Layout

4.1 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*. See Section 4.2.

Meta-comments (from now on, *commands*) are lines that start with `%%`, 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* 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).

I remind you that a “point” (PostScript point) is 0.3527 mm .

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 | A,2A,2 | B,2B,2 | A,2A,2 |
w: A- ve, a- ve, ve- rum
[V: 4] z4 | z4 | D,2D,2 | D,2D,2 | C,2C,2 |
[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] F,2F,A, | A,F,A,2- | A,4 | B,4 | A,4 |
[V: 8] D,4- | D,4- | D,4- | D,4 | C,4 |
% 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]	A, 2A, 2	A, 3A, A, A, A, A,	A, 3A,	A, 4
w: cor- pus na- tum de Ma- ri- a Vir- gi- ne,				
[V: 4]	D, 2D, 2	A,, 3A,, D, D, C, C,	D, 3D	A,, 4
[V: 5]	[EG] [DF] [DF] [FA]	AEEA [FA] [df] [eg]G	[E2G2] [D2F2]	[C4E4]
[V: 6]	x4	C2C2 DAAD	x4	x4
[V: 7]	A, 4	A, 4 A, 3A,	A, 4	x4
[V: 8]	D, 4	A,, 4 D, 2C, 2	D, D, F, D,	A,, A, E, C,

Ave Verum
per coro e organo

W. A. Mozart (1756–1791)

Adagio

Soprano
Alto

Tenore
Basso

Organo

cor - - pus na - - tum de Ma - ri - a Vir - - gi - ne,

cor - - pus na - - tum de Ma - ri - a Vir - - gi - ne,

Figure 4.1: Ave Verum with formatting parameters.

The result is shown in Figure 4.1. Quite a big change, isn't it? The difference should be clear. A complete list of available commands is presented in Appendix A.5.

4.1.1 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: vocalfont Times-Roman 12` is equivalent to `%%vocalfont Times-Roman 12`.

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

```
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- ël, ô dou- ce nuit! w: L'é- toile~est là qui nous con- duit.
%%vocalfont Times-Roman 12
w: Si - lent night! Ho - ly night! All is calm, _ all is bright.
```

Silent Night

F. Gruber

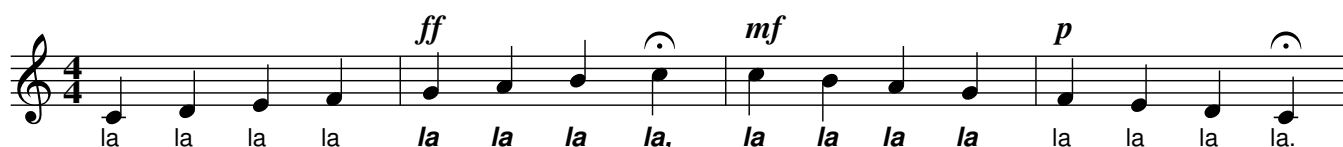
Andante tranquillo



A - stro del ciel, Par - gol di - vin, mi - te A - gnel - lo re - den - tor!
 Voi - ci No - ël, ô dou - ce nuit! L'é - toile est là qui nous con - duit.
 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 fonts will be explained in Section 4.1.3.

4.1.2 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
U: X = !xstem!
K:none
%%staves 1 2
V:1
cc//c//c//c//c2 | [CEGc]2[CEGc]2|
V:2 bass
XC,C,//XC,//C,//XC,//XC,2|X[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||
```



4.1.3 Using Fonts and Text

`abcm2ps` supports nearly all PostScript fonts, which are listed in Appendix A.7. Three fonts are especially important: Times, Helvetica, and Courier; all of them have 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. Text-related commands are:

- `%%font` declares a font;
- `%%textfont` sets a font;
- `%%begintext...%%endtext` define a range of text lines, which must start with `%%`;
- `%%center` centers some text;
- `%%vskip` inserts the specified vertical space;
- `%%sep` inserts a short separator.

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 4.2.

```
% 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|

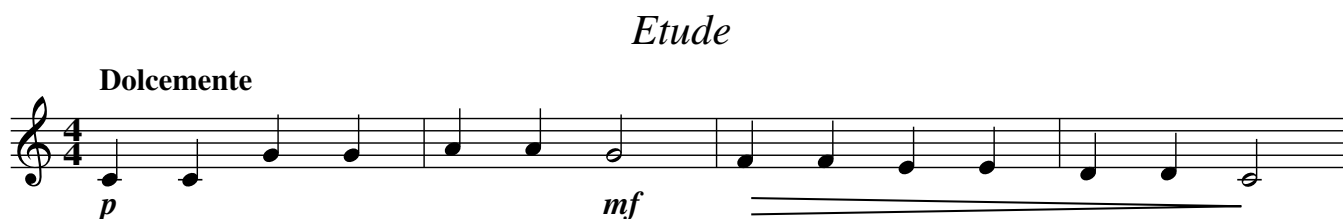
%%vskip 0.4cm
%%textfont AvantGarde-Book 14
%%begintext align
%%Now we'll have a look at something more lively. To start with, let's
%%switch fonts: from Bookman-Light to AvantGarde-Book. Here is the same
%%Etude with a few small variations to make it more interesting:
%%endtext
X: 2
T: Etude
T: second version
M: 4/4
L: 1/4
```



```
Q: "Adagio"
K: C
%
.CDCB,C.GAGFG|A>AG2|.FGFEF.EFEDE|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
```

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:



End of the example.

Figure 4.2: Alternating text with music.

Note

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 in points. 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,
%%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.



4.1.5 Multi-column Output

Text and music can be laid out in multiple columns. 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



4.1.6 Customising Titles

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

Complex titles are specified using `%%titleformat`, followed by a set of letters, digits, and commas. Letters refer to ABC fields, and may be anything in the range A...Z; 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, The
C: Anonymous
H: Written many many years ago...
L: 1/4
R: march
K: C
CDEF|GABc|CDEF|GABc|]
```

First Title

Second Title

The Last Title

march

Anonymous

Written many many years ago...



Note the strange third title. As you can see, it was rearranged to read “The Last Title”. Whenever `abcm2ps` finds that the last word of a title starts with a capital letter and it’s preceded by

a space and a comma, the word is moved to the head of the title. So it's possible to write titles like "Bay of Fundy, The", which sorts alphabetically in a more logical way.

To force the last word to its position (for example, "That's All, Folks"), write this line before the tune: `%%titletrim 0`.

4.1.7 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:

- `$d` prints date and time of the last modification of the current input file;
- `$D` prints the current date and time;
- `$F` prints the name of the current file;
- `$I<x>` prints the contents of a field (`<x>` is capital letter, from `A` to `Z`);
- `$P` prints the page number;
- `$P0` and `$P1` print the page number, but only if it is even or odd;
- `$T` prints the title of the current tune;
- `$V` prints `abcm2ps-` followed by the version number;
- `\n` starts a second line of text.

The three fields must be separated by a *tab character* (see Section 1.2.2.) 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"
```

Note

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

4.1.8 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
```

Testing the use of my logo



Beautiful music presented by...



`abcm2ps` will search for `logo.eps` in the directory containing the `.abc` sources or the format files (see below).

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 7.3.

4.2 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 a sample format file:

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

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
```

Binary packages for GNU/Linux usually store the format files in `/usr/share/abcm2ps`. The following command:

```
abcm2ps -V
```

will report the default format file directory.

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 4.5. Format files containing libraries of symbols can thus be employed when needed.

4.3 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 $\langle n \rangle$ measures, you use `%%measurenb $\langle n \rangle$` . 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 $\langle number \rangle$` 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.

4.3.1 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 2.1);
- when you want each staff to contain $\langle n \rangle$ measures, use the command `%%barsperstaff $\langle n \rangle$` in the source or the option `-B $\langle n \rangle$` in the `abcm2ps` command line;
- `%%alignbars $\langle int \rangle$` aligns the bars of the next $\langle int \rangle$ 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.

Warning

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.

4.4 Saving Space

Quite often, one wants to print 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 $\langle factor \rangle$` . 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 $\langle factor \rangle$` can be used to reduce the horizontal spacing between notes. Compression is maximum with $\langle factor \rangle = 1$, minimum with $\langle factor \rangle = 0$.
- 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 to consider the needs of those who don't have 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.

4.5 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.

4.5.1 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 {0.3 setlinewidth} bdef % default: 0.7
```

so the score looks better at high values of `%%scale`.

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

When writing several lines of PostScript code, using the sequence `%%beginps... %%endps` is recommended.

4.5.2 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 code¹ should be saved in a format file called `italian.fmt`:

```
% italian.fmt
% written by Christopher Lane
% for abcm2ps 5.9.*
% (adapted for more abcm2ps versions by Hudson Lacerda)

% -- latin guitar chords
beginps
/gcshow % string gcshow
-5 0 RM
dup 0 get
dup dup 65 ge exch 71 le and
65 sub [(La) (Si) (Do) (Re) (Mi) (Fa) (Sol)] exch get show
  currentfont /Encoding get exch get glyphshow
ifelse
dup length 1 sub 1 exch getinterval
%
dup mark exch (m) search
```

¹kindly provided by Christopher Lane.

```
(di) search cleartomark
length exch pop exch (aj) anchorsearch cleartomark
pop /tempstr 4 2 roll cleartomark
def tempstr exch (-) putinterval tempstr
ifelse
ifelse

cleartomark ifelse

currentdict /gchshow known
/gchshow load cshow %% abcm2ps older than abcm2ps-5.1.0

show %% abcm2ps-5.1.0 or newer
ifelse
!
endps

% End of file italian.fmt
```

If we add `-F italian` to the command line, the usual scale will be printed with Italian accompaniment chords:

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



4.5.3 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.
- $\langle ps \rangle$ is the name of the PostScript routine that draws the symbol. This may be a user-defined routine or one provided by `abcm2ps`;
 - $\langle h \rangle$ expression height in points;
 - $\langle wl \rangle$ and $\langle wr \rangle$ are not used;
 - finally, $\langle string \rangle$ 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 (f)     6 pf      20 0 0 (f)
%deco (ff)    6 pf      20 0 0 (ff)
X: 1
T: New symbols
K: C
!fp!CDEF GABc|CDEF !(f)!GABc|!(ff)!CDEF !ff!GABc|
```

New symbols

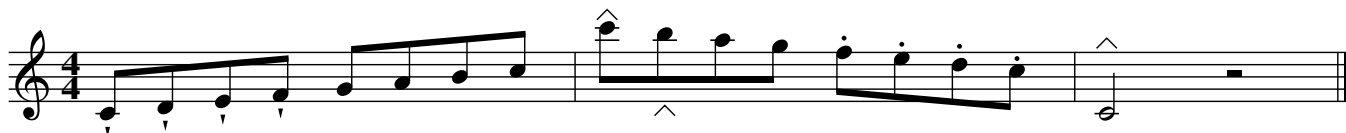


The `%%deco` line implements three new symbols: `!fp!`, `!(f)!`, and `!(ff)!`.

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|]
```



4.5.4 Adding Fonts

Standard Ghostscript fonts are usually enough for most users. However, if you wish to add a special touch to your scores, new fonts can be added. Recent versions of Ghostscript support both PostScript and TrueType fonts. For more details, please see Appendix A.5.3.

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 Fette 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:\ProgramFiles\gs\gs8.71\lib on Windows systems. (If needed, change the version number). On Ubuntu GNU/Linux and possibly other Unix variants, the directory is /usr/share/fonts/type1/gsfonts/. If you’re unsure, open a terminal window and type the following command

```
$ gs -h
```

to get a list of directories where Ghostscript searches for fonts.

Now edit Ghostscript’s font list. On Windows, this file is C:\ProgramFiles\gs\gs8.71\lib\Fontmap (create it if it doesn’t exist), on Ubuntu it is /var/lib/defoma/gs.d/dirs/fonts/Fontmap. Move to the bottom of the file and add this line:

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

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

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

We are now ready to use the new font in ABC files. First of all, declare it using the %%font command followed by the font name. Look at this:

```

%%font Fette-Haenel-Fraktur
%%titlefont Fette-Haenel-Fraktur 24
%%textfont Fette-Haenel-Fraktur 18
%%composerfont Fraktur 16 % alias
%%vocalfont Fraktur 12
X: 1
T: Test: Fette-Haenel-Fraktur font (Fraktur)
L: 1/4
K: C
%
CDEF|GABc|cBAG|FEDC|
w: Do Re Mi Fa... ||||
%%text ABCDEFGHIJKLMNOPQRSTUVWXYZ
%%text abcdefghijklmnopqrstuvwxyz 1234567890

```

Test: Haenel-Fraktur font (Fraktur)



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.

4.5.5 Customising Tuplets

The `%%tuplets` command allows for fine-grained tuplet indications. 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 $\langle n \rangle$ in the tuplet, 1 = no value at all, 2 = a ratio $\langle n \rangle : \langle t \rangle$.

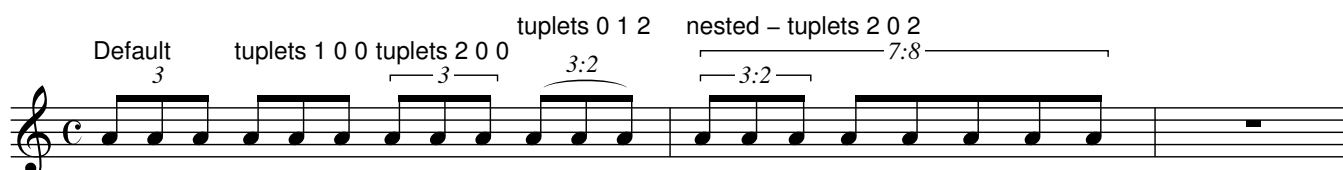
Common values are, for instance, `I:tuplets 2 0 2` and `I:tuplets 2 0 0`:

```

X: 1
T: Triplets
M: C
L: 1/4
K: none
%
"^Default"(3A/A/A/ "^triplets 1 0 0"[I:triplets 1 0 0](3A/A/A/ \
"^triplets 2 0 0"[I:triplets 2 0 0](3A/A/A/ \
"^triplets 0 1 2"[I:triplets 0 1 2](3A/A/A/|\
"^nested - triplets 2 0 2"[I:triplets 2 0 2](7:8:8(3A/A/A/ A/A/A/A/A/|z4]

```

Tuplets



4.5.6 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 six-holed diatonic fipple flute, 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!

The format file `flute.fmt` contains fingerings definitions for tin whistles in several keys and *galoubet*, a French three-hole fipple flute.

Specify the following parameters on the `abcm2ps` command line:

`-F flute.fmt -T <key>`

to obtain fingerings for the whistle in the key of `<key>` printed under the staff. The parameter `<key>` is a digit corresponding to a key as per the following table:

Digit	Key
1	D
2	C
3	E \flat
4	B \flat
5	F
6	G
7	A
8	C (galoubet)

Here is an example. The whistle in the key of D is the most commonly used; we'll specify the parameters -F flute.fmt -T1. An excerpt from "Atholl Highlanders":

```
X: 1
T: The Atholl Highlanders
M: 6/8
L: 1/8
R: jig
K: AMix
%
|:e2e ecA|ecA Bcd|e2e ecA|Bcd cBA|
e2e ecA|ecA Bcd|eae fed|cdB A3:|
```

The Atholl Highlanders

Chapter 5

Playing

5.1 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:



Very important!

while a score in PDF format will look the same on any computer, this 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 that is used to listen to it.

Having a MIDI version of your ABC music is convenient, because you get an immediate feedback of what you wrote. It's especially true for people who can't easily read music.

To this end, the free `abc2midi` program is helpful. Given a tune collection in a file called `file.abc`, `abc2midi` creates a MIDI file for each tune adding the index number of the `X:` field to each file name: `file1.mid`, `file2.mid`, ... `abc2midi` is a command-line program, but it's integrated in ABCexplorer and JEDABC.

`abc2midi` supports up to 16 voices ("channels", in MIDI lingo) that can play at the same time, as defined by the MIDI standard. Each channel can be associated one of the standard 128 MIDI instruments ("programs"), and their loudness ("velocity") can be controlled independently. Don't ask me about the strange terminology.

`abc2midi` is just one of the programs of the `abcMIDI` package:

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

Before we start, some information for GNU/Linux users is needed.

5.1.1 A Software MIDI Player: TiMidity++

Some advanced sound cards have a built-in MIDI synthesizer, but cheap cards usually do not provide this feature. In this case, some GNU/Linux users may need a software MIDI player. One of the best is Timidity++ (<http://timidity.sourceforge.net/>), usually provided by major GNU/Linux distributions.

Unfortunately, there's a catch. By default, TiMidity++ may not provide all 128 MIDI instruments, which are mapped to so-called *instrument patches*. Simply put, these are files that provide sounds. Some patches may be missing, which results in MIDI files that apparently do not play or miss some voices.

Replacing the missing instruments is easy. By default, the configuration files are:

```
/etc/timidity/timidity.cfg
/etc/timidity/freepats.cfg
```

The first file should contain a couple of lines that read:

```
# By default, try to use the instrument patches from freepats:
source /etc/timidity/freepats.cfg
```

To replace a missing instrument, edit the second file. Let's suppose you miss the instrument 41 (Viola) under the `bank 0` section. Add this line:

```
41      Tone_000/040_Violin.pat
```

which implements the Viola instrument using the same patch used for violin. Not perfect, but it works.

An alternate way to provide more instrument patches to TiMidity++ is by installing the package FluidSynth, <http://www.fluidsynth.org>, as I did on my Ubuntu machine.

5.1.2 Our First Midi

Do you remember the first scale and the command-line instructions we examined in Section 1.2.3? We wrote a very simple ABC source and turned it into PostScript using `abcm2ps` from the command line.

In a similar way, we can obtain our first MIDI file. Given the source file `scale1.abc` (or another source you want to convert), this command will create the corresponding MIDI file:

```
abc2midi scale1.abc
Warning in line 2 : No M: in header, using default
writing MIDI file scale11.mid
```

The new file `scale11.mid` will be written in the same folder as the source. Warning and/or error messages will be issued when minor and/or serious problems are found.

By default, the MIDI file name is composed using the source file name followed by a number, corresponding to the number in the `K:` field. Double-click on the MIDI file to start the default player.

5.1.3 Example: How To Make a Ringtone

MIDI files are not very impressive to listen to, but they can be very useful. For example, I made my own simple ringtone following a few simple steps; I used TiMidity++ to convert the MIDI file to WAV, then Lame (<http://lame.sourceforge.net/>) to encode the WAV file to MP3.

First, I wrote the ringtone (Ring.abc):

```
X:1
M:2/4
L:1/32
Q:1/4=240
K:Amaj
%%MIDI program 1 80
|:ac'ac'ac'ac'ac'ac'ac'ac'|ac'ac'ac'ac'ac'ac'ac'ac'|[L:1/2]z:|
```

then I ran `abc2midi` and obtained `Ring1.mid`. To convert the MIDI file to MP3, I used the following commands (boldface):

```
~$ timidity -A400 -Ow Ring1.mid
Playing Ring1.mid
MIDI file: Ring1.mid
Format: 0 Tracks: 1 Divisions: 480
Sequence: Trillo
Text:
Output Ring1.wav
Playing time: ~7 seconds
Notes cut: 0
Notes lost totally: 0
~$ lame Ring1.wav Ring1.mp3
LAME 3.98.4 32bits (http://www.mp3dev.org/)
CPU features: MMX (ASM used), SSE (ASM used), SSE2
Using polyphase lowpass filter, transition band: 16538 Hz - 17071 Hz
Encoding Ring1.wav to Ring1.mp3
Encoding as 44.1 kHz j-stereo MPEG-1 Layer III (11x) 128 kbps qval=3
  Frame          | CPU time/estim | REAL time/estim | play/CPU |    ETA
  206/206   (100%)|    0:00/    0:00|    0:00/    0:00|   24.460x|    0:00
-----
  kbps      LR   MS  %      long switch short %
  128.0     53.9 46.1      98.5   1.0   0.5
Writing LAME Tag...done
ReplayGain: -2.4dB
~$ _
```

Then I transferred `Ring1.mp3` to my smartphone.

Note

In the following sections, I will include some sound samples. In theory, MIDI files created by `abc2midi` should be playable on all computers, but unless you have the same instrument patches that I use (or compatible ones) there is a chance that you can't listen to them. Therefore, all MIDI files were converted to MP3 files with `Lame`, as shown above.

5.1.4 Supported Ornaments

Some of the ornaments described in Section 2.2.8 are supported by `abcMIDI` and produce audible output. These are:

```
!ppp! !pp! !p! !mp! !mf! !f! !ff! !fff! !breath!
!arpeggio! !crescendo(! !crescendo)! !<(! !<)!
!diminuendo(! !diminuendo)! !>(! !>)!
```

Default volume is equivalent to `!f!`.

5.1.5 %%MIDI Commands

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

`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.

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 using an inline `I` field: `[I:MIDI = <command> <parameters>]` (spaces are optional). Note the `=` character.

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 makes the source more readable.

5.1.6 Voices and Instruments

Let us consider “Ave Verum”, which we examined in Section 3.1.2. 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 the 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
%
V: S % enable the first track
%%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”).

Note the `V: S` line before the `%%MIDI program` lines. It should not be necessary, but it’s needed as a workaround for a bug in current `abc2midi`. Basically, it forces the instrument changes to occur in the first MIDI track.

Another way to assign an instrument to a voice is the following:

```
%%MIDI channel 1 % selects melody channel 1 (range: 1-16)
%%MIDI program 53 % assigns 53 to current channel
```

The numbers that identify each instrument are listed in Appendix A.8. Note that the numbers in the list range from 1 to 128, whereas `abc2midi` numbers them 0 to 127. If no MIDI program is specified, voices will be assigned by default the General MIDI instrument 1.

5.1.7 Accompaniment Chords

The accompaniment chords described in Section 2.2.6 are used by `abc2midi` to generate an accompaniment to the main melody. A lower case letter `a...g` can be used as note name; `abc2midi` will play the fundamental note instead of the chord.

If the optional note is present, as in `C/B` or `C/E`, two cases are possible. If the note does not belong to the chord as in the first example, it will be added below the fundamental note of the chord. If it does belong as in the second example, *chord inversion* will be applied and the note will become the lowest in the chord.

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 5.1.10.

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 `fzczzc`: fundamental, rest, chord, rest, fundamental, rest, chord, rest.

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

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:



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.

If you don't hear accompaniment chords, your tune might have a time signature for which a predefined `fcz` sequence is missing: 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. Here is the above 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 c2c2zc
%%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 file [folia.mp3](#).

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: Garôta De Ipanema
T: (The Girl From Ipanema)
C: Antônio 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 *P:A* field:

```

%%MIDI program 67          % Baritone Sax
%%MIDI gchord fzcffczc % bossa nova (approximate)
%%MIDI chordvol 30
%%MIDI bassvol 30
%%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.mp3](#).

5.1.8 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 are placed 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
E/F/|
% flattish
%%MIDI beat 105 95 80 1
"Em"GEE BEE |GEG BAG |"D"FDD ADD      |dcd "Bm"AGF|
"Em"GEE BEE |GEG B2c |"D"dcd "Bm"AGF|"Em"GEE E2 E/F/|
% more swing
%%MIDI beat 105 90 60 3
"Em"GEE BEE |GEG BAG |"D"FDD ADD      |dcd "Bm"AGF|
"Em"GEE BEE |GEG B2c |"D"dcd "Bm"AGF|"Em"GEE E2 B|
%%MIDI beat 105 95 80 1
"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 E2B|
%%MIDI beat 105 90 60 3
"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 E2z|]

```

that converts into this MIDI file: [swallowtail.mp3](#). Note how the different parts have less or more swing.

To remove temporarily the emphasis programmed with `%%MIDI beat`, use the command `%%MIDI nobeataccents`. To resume it, use `%%MIDI beataccents`.

5.1.9 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 `fcbg hijGHIJz`.

5.1.10 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 5.1.7, $\langle n1 \rangle$ is the chord fundamental 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.

5.1.11 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.mp3](#).

Note

Note that `abc2midi` plays the notes in a different way than they are notated. For instance, `A>B` should be played as $A^{3/2} B^{1/2}$ but is actually played as $A^{4/3} B^{2/3}$. That is, instead of a ratio 3:1 `abc2midi` employs a ratio 2:1. The rationale (pun unintended) is that 2:1 is the way hornpipes are approximately played.

This behaviour can be reversed to normal with the `%%MIDI ratio 3 1`.

5.1.12 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 A.8.2) 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.mp3](#)):

```
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 ccccccc
%%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]|
```

5.1.13 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 A.8.2, the following tune will have a melody played by piano, and low-tom, hi-hat and triangle accompaniment ([accomp.mp3](#)):

```
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/| % 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:

$\text{♩} = 120$

The musical score displays four staves, each representing a different percussion instrument. The top staff is for the Piano, showing a melody of eighth notes. The second staff is for the Low Tom, with notes on the first and third beats of each measure. The third staff is for the Open Hi Hat, also with notes on the first and third beats. The bottom staff is for the Open Triangle, featuring a continuous eighth-note pattern with 'x' marks on the second and fourth beats of each measure. The time signature is 4/4, and the tempo is marked as 120 beats per minute.

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

5.1.14 Portamento

A glide effect can be added using the `%%MIDI portamento [bass] [chord] <n>` command, where `<n>` is a number between 0 (no effect) and 63 (maximum effect).

The following is the riff from *Impressioni di Settembre*, a great song by Italian progressive rock band Premiata Forneria Marconi. The first repeat has no glide effect, while the second repeat mimics the original Moog glide ([impressioni.mp3](#)):

```
X: 1
T: Impressioni di Settembre (riff)
C: Premiata Forneria Marconi, 1972
M: 4/4
L: 1/8
Q: 1/4=140
K: Dm octave=-1
%
%%MIDI program 1 81
%%MIDI chordprog 80
%%MIDI gchord b4
%%MIDI chordvol 50
%
%%MIDI portamento 0
|:"Dm"D4-DE F2|G A2 c d2 f e-|"C"e d c2 g4-|g fed- dcg2|
"G"f e2 d2 c =B2-|=B A G3 BG A-|"Dm"A d- d6-|d8
%%MIDI portamento 60
:|
D8-|D8|]
```

`%%MIDI noportamento` turns off the effect.

5.1.15 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> <vel1> <vel2>` 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; `<vel1>` and `<vel2>` 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 5.1. 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 5.1: 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|/AG2 B/G/ B2 A|/AG2 FE D2 D|
/AG2 /CB/G/ /CB2 A/B/|d3-d2 B|
d2 B/G/ B2 A|G2 E /ED2 D|
/AG2 B/G/ B2 A|/AG3-G2 |]
%%MIDI droneoff
```

The resulting MIDI file is [amazinggrace.mp3](#).

5.1.16 midi2abc

This program converts a MIDI file to the corresponding ABC 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 file will contain only one measure per line.

The best way to get an ABC source from a MIDI file is using `runabc.tcl` as shown in Figure 5.1. 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 source.

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

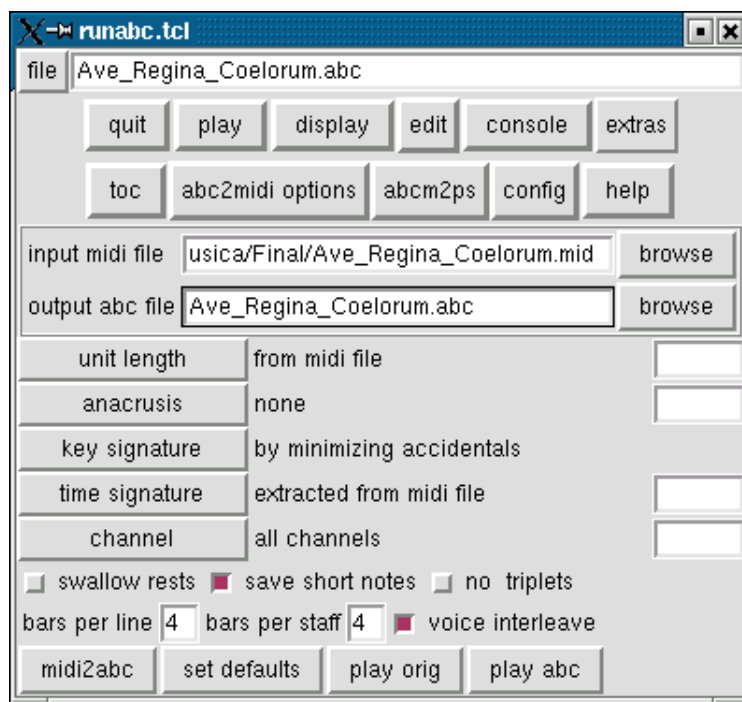


Figure 5.1: Converting a MIDI file to ABC with `runabc.tcl`.

hold true. Remember what I explained in Section 5.1. 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`.

5.1.17 Global Settings

In a tune collection (i.e. a file containing several tunes), the same MIDI settings could be shared by all or most tunes. Instead on inserting the same `%%MIDI` lines in each tune, it's possible to place them at the beginning of the file; the parameters will apply to all tunes.

These parameters can be

- `%%MIDI C`
- `%%MIDI nobarlines`
- `%%MIDI barlines`
- `%%MIDI fermatafixed`
- `%%MIDI fermataproportional`
- `%%MIDI ratio`
- `%%MIDI chordname`
- `%%MIDI deltaloudness`

Obviously, these parameters may be overridden by `%%MIDI` lines within each tune.

5.2 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 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.



Chapter 6

Converting

6.1 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 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*.

6.1.1 Basic Usage

Let us look at an example. We will write a portable ABC 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 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`.

6.1.2 Advanced Usage

We have seen in Figure 3.1 an example of ABC 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, \2411a guarda ribera!
C: Villancico (Spagna, sec. XVI)
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, \2411a 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: Diós guar-dó 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: Diós guar-dó el lo-bo de nue-stra cor-de-ra.
% WOMEN AND MEN
#endif 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: Diós guar-dó el lo-bo de nue-stra cor-de-ra,
[V: 2] z2EE    |DCEC|D2CD2DCC|D2D2z2|
w: Diós guar-dó el lob', el lo-bo de nue-stra cor-de-ra,
[V: 3] ccBc    |A2BA|A2AA2AAA|A2A2z2|
w: Diós guar-dó 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: Diós guar-dó el lo-bo de nue-stra cor-de-ra.
[V: 2] z2EE|DCEC|D2CA,2A,A,A,|A,2A,2z2|
w: Diós guar-dó el lob', el lo-bo de nue-stra cor-de-ra.
[V: 3] ccBc|A2BA|A2AF2FEE      |D2D2z2  |
w: Diós guar-dó 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 Diós po-de-
%
#ifdef MIDI
[V: 1] z8      |z4|z4  |z8      |
[V: 2] z8      |z4|z4  |z8      |
[V: 4] f2feggef|d4|aaga|f2fedefg|
#else
[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|
#else
[V: 3] A4|AAGA|F2FEGGEF|D2D2|
w: car: ni~aun o-ri-gi-nal e-sta Vir-gen no tu-vie-ra.

```

6.2 abc2abc

IT is part of the `abcMIDI` package. This command-line program is used to modify the ABC 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 $\langle x \rangle$ outputs only voice $\langle x \rangle$ of a polyphonic file.¹

-x $\langle n \rangle$ for a file with several pieces, renumbers the X: field starting with $\langle n \rangle$.

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:

```
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.



¹The program abc2prt (Section 7.4) works better.

Chapter 7

Other Possibilities

7.1 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 GSview. 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.

7.2 Inserting Music in L^AT_EX

This guide is written in L^AT_EX, which you may want to use instead of a word processor. To insert ABC music in L^AT_EX 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.

7.2.1 Using `abc.sty`

Prof. Enrico Gregorio of University of Verona, Italy, has written a package that enables the inclusion of ABC code in \LaTeX documents. The relevant archive, called `abc.zip`, can be freely obtained from CTAN mirrors. Please see Section A.1 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
\file{tune.abc}:

\medskip

\abcinput{tune}

\end{document}

```

7.3 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.

7.4 Parts Extraction

Another useful tool is `abc2prt`, which extract voices from polyphonic sources. You use `abc2prt` to create new ABC 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 4.1. All you have to do is run `abc2prt` this way:

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

A new ABC file called `aveverum-3.abc`, containing only voice 3, will be created. Needless to say, `abc2prt` is integrated in JEDABC.

7.5 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.

7.6 Final Comments

I started with ABC several years ago, and it's always been my tool of choice whenever I needed to print or listen to a piece of music. This manual started out as a list of notes I wrote for myself, and it slowly grew to its present form. Since the ABC software is free, I decided it was only fair to release this manual for free as well.

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 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!

7.6.1 Please, Make a Donation...

I say again, this manual is free. That said, if the results of my work on ABC 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.

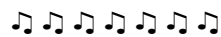
This is my home address: Guido Gonzato, Via Monte Ortigara 2/a, 37126 Verona, Italy.

Thank you so much!

7.6.2 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 threwed when she had a six-year-old kid listen to Rigoletto, so many years ago. Ciao, Annarosa.



Appendix A

Bits & Pieces

A.1 Web Links

The ABC PLUS home page:

<http://abcplus.sourceforge.net>

The original ABC page:

<http://www.abcnotation.com>

The ABC Project page:

<http://abc.sourceforge.net/>

The abcm2ps home page:

<http://moinejf.free.fr>

The abc2midi home page and user guide:

<http://ifdo.pugmarks.com/~seymour/runabc/top.html>

http://ifdo.pugmarks.com/~seymour/runabc/abcguides/abc2midi_guide.html

Hudson Lacerda's ABC page, including advanced customisation and PostScript new routines:

<http://hudsonlacerda.webs.com/>

The Folkinfo web-based ABC converter:

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

Abctab2ps is an almost complete implementation of ABC PLUS that supports tablatures for lute:

<http://www.lautengesellschaft.de/cdmm/>

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:

<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 L^AT_EX:

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

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

<http://ficking-music-archive.org/>

A.2 ABC 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
I:	body	Meta-command. I:MIDI program 2 32
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
s:	body	Decorations. s:!pp! * * !mf! * !ff!
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

A.3 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.

A.4 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 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	ı \241	ç \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

A.5 Formatting Commands

Command parameters will be specified as follows:

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

A.5.1 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 4.1.5 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>*.

A.5.2 Text

These commands are used to write text lines within a tune and between tunes. Fonts 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 <int>`: sets the default text option to be used between `%%begintext` and `%%endtext`, or in `%%EPS` files. The parameter can be a digit or a corresponding string: 0 (or `obeylines`), 1 (`justify`), 2 (`fill`), 3 (`center`), 4 (`skip`), 5 (`right`). If the option is 4 (`skip`), no text or EPS is output.

A.5.3 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 A.7. I remind you that indications for adding new fonts are given in Section 4.5.4.

`%%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 4.1.6 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.

A.5.4 Spacing

These commands specify spacing between score elements.

%%barsperstaff *<int>*: tries to typeset the score with *<int>* bars on each line.

%%breakoneoln *<logical>*: if true, treats an end of line as if it were a space (i.e. breaks note beams).

%%composerspace *<length>*: sets the vertical space before the composer to *<length>*.

%%gracespace *<float>* *<float>* *<float>*: defines the space before, within and after grace notes.

%%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.

%%musicospace *<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⟩* [*“f”*]: sets a *⟨length⟩*-long break (gap) in the current staff. If the letter “f” is present, the staff break is forced even if it occurs at the beginning or end of a line.

%%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. *⟨h⟩* may be a negative value.

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

A.5.5 Other Commands

Miscellaneous commands are grouped in this section.

%%abc2pscompat *<logical>* : if true, reverts to the old method of dealing with notes in bass and other clefs. Please see Section 2.2.1.

%%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 true, prints clefs and possibly clef changes when no clef is defined in K: or V:.

%%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 4.5.

%%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.
- %%info** *<logical>*: if true, displays the rhythm and the origin on the same line, plus the A: field.
- %%infoname** *<string>* *<string>*: defines the fields to be printed when %%writehistory is true.
- %%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.
- %%score** *<string>*: like %%staves, but with a different behaviour regarding bars. See Section 3.1.2 for details.
- %%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.

%%shiftunisson *<logical>*: if true, shifts note heads that belong to different voices that are in unison. It applies to dotted notes and notes shorter than minim.

%%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.

%%staffnonote *<logical>*: if false, staves with no notes are not printed.

%%staves *<string>*: defines how staves are to be printed. See Section 3.1.2 for details.

%%stemheight *<float>*: sets the stem height to *<float>*.

%%tablature: TO BE WRITTEN

%%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.

%%titletrim *<logical>*: if true, move the last word of a title to the head if it starts with a capital letter and it’s preceded by a space and a comma.

%%tuplets *<int1>* *<int2>* *<int3>*: defines how tuplets are to be drawn. See Section 4.5.5 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.

A.6 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: turns off %%nobarlines.

%%MIDI bassprog *<int>*: sets the MIDI instrument for the bass notes in accompaniment chords 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 $k = 0, 1, 2, \dots, x-1$ within each bar. If k is a multiple of *<int4>*, then the note is “strong”.
- %%MIDI beataccents**: reverts to normally emphasised notes. See also %%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. This number should normally be a multiple of 12.
- %%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 redefines existing ones as was seen in Section 5.1.10.
- %%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 drumbars** *<int>*: specifies the number of bars over which a drum pattern string is spread. Default is 1.
- %%MIDI drum** *<str>* *<int1 int2 int3 int4 int5 int6 int7 int8>*: generates a drum accompaniment pattern, as described in Section 5.1.12.
- %%MIDI drummap** *<str>* *<int>*: associates the note *<str>* (in ABC notation) to the a percussion instrument, as listed in Section A.8.2.

%%MIDI drumoff turns drum accompaniment off.

%%MIDI drumon turns drum accompaniment on.

%%MIDI fermatafixed: expands a `!fermata!` by one unit length; that is, HC3 becomes C4.

%%MIDI fermataproportional: doubles the length of a note preceded by `!fermata!`; that is, HC3 becomes C6. `abc2midi` does this by default.

%%MIDI gchordbars *<str>*: spreads the gchord string over *<n>* consecutive bars of equal length. The gchord string should be evenly divisible by *<n>* or else the gchords will not work properly.

%%MIDI gchord *<str>*: sets up how guitar chords are generated. Please see Section 5.1.7.

%%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 the `abcMIDI` documentation.

%%MIDI nobarlines: normally, an accidental applied to a note also applies to other equal notes until the next bar. By using this command, the accidental will apply to the following note only.

%%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 [*bass*] [*chord*] *<int>*: turns on the portamento controller (glide effect) on the current channel (or to bass/chord channel) and set it to *<int>*. 0 turns off the effect.

%%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 $\langle int1 int2 \rangle$: 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 $\langle int1 \rangle$: transposes relatively to a prior `%%transpose` command by $\langle int1 \rangle$ semitones; the total transposition will be $\langle int1 + int2 \rangle$ semitones.

%%MIDI snt $\langle int \rangle \langle float \rangle$: alters the standard MIDI pitch of a note $\langle 0-127 \rangle$; e.g. `%%MIDI snt 60 60.5` alters the pitch of C.

%%MIDI temperament $\langle int1 \rangle \langle int2 \rangle$: TO BE WRITTEN

%%MIDI temperamentlinear $\langle float1 float2 \rangle$: changes the temperament of the scale. $\langle float1 \rangle$ specifies the size of an octave in cents of a semitone, or 1/1200 of an octave. $\langle float2 \rangle$ specifies in the size of a fifth (normally 700 cents).

%%MIDI temperamentnormal: restores normal temperament.

%%MIDI transpose $\langle int1 \rangle$: transposes the output by $\langle int1 \rangle$ semitones. $\langle int1 \rangle$ may be positive or negative.

%%MIDI trim $\langle int1 \rangle \langle int2 \rangle$: controls the articulation of notes and chords by placing silent gaps between the notes. The length of these gaps is determined by $\langle int1 \rangle / \langle int2 \rangle$ 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 $\langle int1 \rangle / \langle int2 \rangle$ be a fraction close to zero. Trimming is disabled inside slurs as indicated by parentheses. Trimming is disabled by setting $\langle int1 \rangle$ to 0.

A.7 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–Demibold*****Bookman–Light*****Bookman–Lightbold***

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***

Συμβολ

ZapfChancery–MediumItalic

A.8 MIDI Instruments

A.8.1 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

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

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

- 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

- 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

- 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

A.8.2 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			