# Making Music with Abc 2

*A practical guide to the Abc notation*

Guido Gonzato

# Making Music with

# ABC 2

The latest version of this manual is available at:
http://abcplus.sourceforge.net/#ABCGuide

*To Annarosa,*
*Bruno,*
*Lorenzo*
♡

# Contents

x

# List of Tables

# List of Figures

# About This Guide

THIS manual explains how to make beautiful sheet music and MIDI files using a computer, some free and open source software, and the ABC 2 music notation. It is aimed at musicians with some computer expertise who don't want to spend a lot of money on commercial music software. Music teachers, students, amateur and professional musicians may benefit greatly from this guide and from the software it describes.

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 open source, and can be freely copied and shared.

I hope you will find my work useful and enjoyable.

Cheers,



Guido Gonzato, PhD =8−)

# Chapter 1

# Music on the Computer with Abc 2

## 1.1 Introduction

M<small>USICIANS</small> who can also use a computer are very lucky people. First of all, because they are musicians; secondly, because the computer is an excellent tool for writing and managing music. Lots of programs are available.

Most music notation programs have a visual (or WYSIWYG) 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 1993, is one of the best available: 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 is a major release of the Abc notation, and has been called Abc 2: its formal description is available at http://abcnotation.com/wiki/abc:standard:v2.1.

A few programs implement most Abc 2 features and provide some extensions, which in turn may become part of the Abc 2 standard in the future. The purpose of this guide is to introduce the reader to Abc 2 and the most important features of its related programs. Ideally, people who could profit from Abc 2 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;

- classically trained musicians who would like to use Abc 2 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 even teach music, you have found the right tool. As an additional bonus, all programs are free and open source.

The Abc 2 home page is http://abcplus.sourceforge.net.

**A Few Words About "Standards".**   Although Abc 2 is formally defined, what you get using software differs slightly from the theoretical standard. This guide will concentrate on practical implementations of Abc 2, not on theoretical features.

To avoid confusion, deprecated syntax variations of old Abc releases will not be explained, or even mentioned unless strictly necessary. The following chapters will cover most features and options of the best available programs.

For the sake of simplicity, in the following I shall simply write "Abc" instead of "Abc 2".

### 1.1.1   Requirements

I assume that you have a PC with Windows, macOS, GNU/Linux or other Unix variants, and that you are reasonably familiar with computers. An Android tablet, or even a phone, is a viable alternative.

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

First of all, I would like to stress that Abc *is not tied to a specific program.* Abc is a musical notation that can be turned to printed music, on paper or web pages, and to sound files by several applications, each of which is just an Abc implementation.

The state of the art of Abc implementation is currently represented by a few great little programs: the `abcm2ps` and `abc2svg` scorewriters, the MIDI creator `abc2midi`, and the Abc/MusicXML translators `abc2xml.py` and `xml2abc.py`. All these programs are free an open source software, released under the GNU GPL license.

`abcm2ps`, written by Jean-François Moine, reads Abc files and converts them to PostScript. This is a file format closely related to PDF, and it can be viewed and printed with another free program: Ghostscript. This application converts PostScript files into several formats, PDF being the most important.

A twin program to `abcm2ps` is `abc2svg`, which is basically `abcm2ps` rewritten in Javascript. This program can be integrated in web pages to provide an Abc editor, viewer, and player that works in any web browser.

`abc2midi` converts Abc files to MIDI. It is part of the `abcMIDI` package, written by James Allwright and currently maintained by Seymour Shlien. `abcMIDI` includes other utilities that convert Abc and MIDI files in several ways.

As their names suggest, the `abc2xml.py` and `xml2abc.py` translators, written by Willem Vree, convert music written from Abc to MusicXML and the reverse. MusicXML is an open format for exchanging digital sheet music, and is supported by many professional applications.

Please note that `abcm2ps`, `abc2svg`, `abcMIDI`, and the Abc/MusicXML tools *are not completely compatible with each other*: each program supports extensions to the Abc standard and/or lacks some features. Usually, this is not a problem; besides, one should expect minor differences, as applications target different musical aspects. For instance, music elements that only make sense in a printed score (e.g. a text annotation) are not supposed to produce audible output. In the following chapters, these differences will be highlighted.

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 specialised facilities for editing ABC files. In general, any application that can be used to write text can also be used to write ABC files.

It's important to stress that `abcm2ps`, `abc2svg`, `abc2midi`, and the `abc2xml.py`/`xml-2abc.py` translators are *command-line driven* programs: in short, you cannot start them double-clicking on their icons, or selecting them from a menu. To use these programs, you will have to open a command shell (Windows Command Prompt, Unix or Android terminal) and type commands.

People with limited computer expertise need not worry. First of all, using the programs in the command line is trivial. Besides, there are applications, and even web pages, that gather all relevant pieces of software in a single integrated environment. For example, EasyABC (Figure 1.1) is a free and multiplatform application that displays and plays the music using the command-line programs internally. More fine software options are listed at the ABC 2 and ABC home pages.
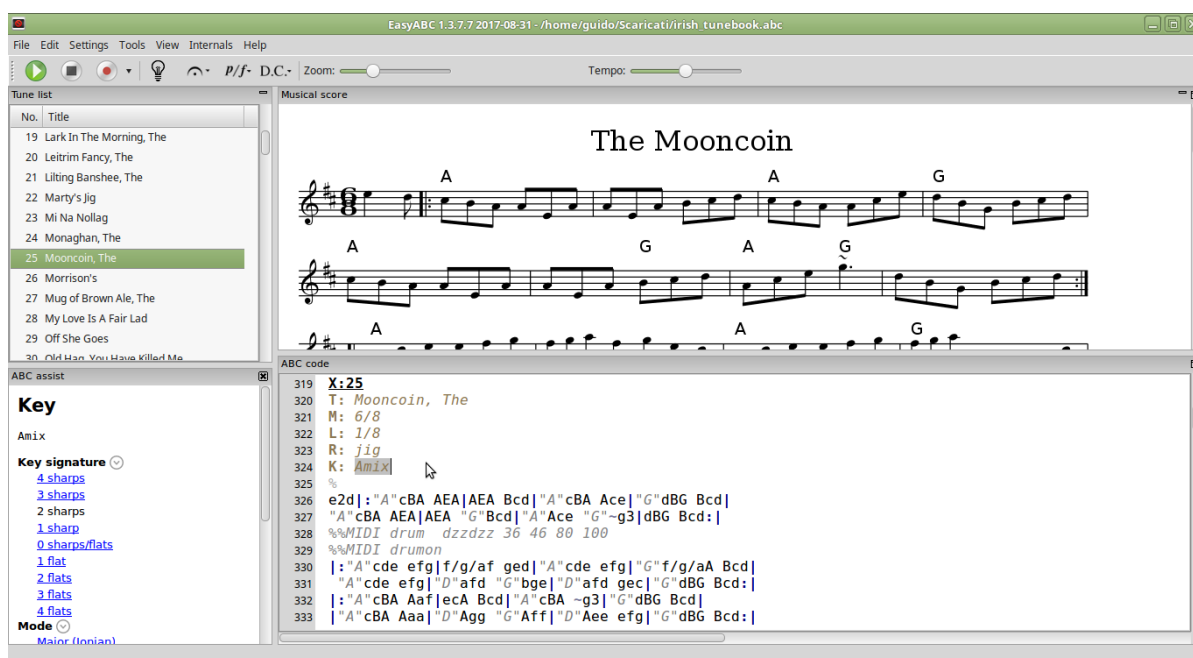


Figure 1.1: Managing a tune collection with EasyABC.

### 1.1.3 Why ABC?

I know from experience that graphical programs, most of which are commercial software, are perceived to be easier to use than non-graphical ones. This is a common misconception, but only expert users realize it. So, why should one learn ABC?

Well, compared to graphical programs and even other text-based notations ABC has many advantages:

**text only:** the importance of this feature can't be emphasised enough. Being simple text, ABC music can be read, written and stored by any computer system on the Earth; it can be used by visually impaired people, emailed, texted by phone, scribbled down on beermats, etc.

**readability:** as stated at the Abc home page:

> One of the most important aims of abc notation, and one that distinguishes it from most computer-based musical languages is that it can also be read easily by humans. In other words, with a little practice, it is possible to play a tune directly from the abc notation without having to process and print it out. The resulting compactness—each tune is about the same as a paragraph of text—and clarity makes it very easy to notate tunes.

**features:** Abc supports nearly all features of Western music including multiple voices and staves, symbols, dynamics, lyrics, text annotations, and much more;

**indexing:** Abc allows for easy creation, searching and indexing of tune books and music archives, many of which are freely available on the web;

**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` or `abc2svg`, 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;

**openness:** Abc is an open standard and it has no owner, so it can be freely implemented by everyone royalty-free;

**low resources:** Abc programs are very light on resources and can run on old computers, or even tablets and smartphones;

**portability:** music is created as SVG or PDF and MIDI files instead of proprietary file formats. Open formats let you 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 documents is very simple;

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

**learning value:** if you teach music, Abc in an invaluable tool that facilitates the learning of music theory.

Needless to say, there are disadvantages as well:

**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 Gregorian chant, contemporary music notation, and non-European music;

**compatibility:** some Abc programs provide extensions to the standard (e.g. tablatures and guitar diagrams) that other programs can't deal with.

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 2 page you will find more complex choral scores.

## 1.2 Getting Started

In order to write music with Abc and the command-line programs, you follow these steps:

1. using an editor, write the tune using the Abc syntax;

2. convert the tune with `abcm2ps`, creating a PostScript file;

3. view the PostScript file and convert it to PDF;

4. optionally, create a MIDI file with `abc2midi`.

Using `abc2svg`, writing music in Abc is even simpler: there is nothing to install, and all you need is a web browser.

If the music you write is free from copyright, please consider sharing it on the web for others to enjoy.

### 1.2.1 Getting the Programs

Relevant web pages:

- the `abcm2ps` scorewriter:
  http://moinejf.free.fr
  http://abcplus.sourceforge.net/#abcm2ps

- the `abc2svg` online scorewriter:
  http://moinejf.free.fr/js/edit-1.xhtml
  http://moinejf.free.fr/js/

- `abcMIDI`:
  http://ifdo.ca/~seymour/runabc/top.html
  http://abcplus.sourceforge.net/#abcMIDI

- the Abc/MusicXML translators:
  https://wim.vree.org/svgParse/abc2xml.html
  https://wim.vree.org/svgParse/xml2abc.html

- Ghostscript (all platforms except Android):
  http://www.ghostscript.com/
  As of this writing, the latest version is 9.26; you may want to download the free, GNU Affero licensed version.

- Ghostscript for macOS:
  http://pages.uoregon.edu/koch/Ghostscript-9.26.pkg

- Sumatra PDF, a fast PDF and PostScript viewer (Windows):
  https://www.sumatrapdfreader.org

- GSView, another PostScript viewer (Windows):
  http://www.gsview.com

Ready-to-run packages for Windows, GNU/Linux, macOS, and Android are available from the ABC 2 web site. Besides, all GNU/Linux distributions include Ghostscript and a document viewer that reads PostScript. SVG output can be viewed with any web browser.

At the moment, Android has no Ghostscript support and only SVG output is usable.

### 1.2.2   Which Program?

The choice of the ABC program that better suits you depends on your needs. In fact, each program has advantages and disadvantages, and gives its best in different cases. There is no such thing as "the best" ABC program.

**abcm2ps**

abcm2ps typesets music in PostScript, which is the best option for printing.

abcm2ps can typeset files containing thousands of tunes without even a hiccup, and it's incredibly fast. It gives the user full control of page layout and output options; many ABC applications use abcm2ps internally to display the music. Unfortunately, abcm2ps is no longer being developed, as its author is concentrating on abc2svg; new ABC features will not be implemented. In its current state, though, abcm2ps is a very capable scorewriter.

If you want to manage big tune collections, or if you need precise page layout customisation, then abcm2ps is the best program for you.

**abc2svg**

abc2svg typesets music in SVG/XHTML, which is meant to be displayed in web pages.

abc2svg can be integrated in web-based ABC editors that make it very easy to use. In addition to displaying the score, web-based abc2svg can also play the music and highlight the notes being played. abc2svg is updated frequently, and includes the latest ABC improvements and extensions. Besides, its output looks slightly better than that of abcm2ps.

The main disadvantage of abc2svg is its lack of control on printed output; printing, in fact, is delegated to the browser. Furthermore, when it is integrated in web-based editors abc2svg cannot handle large tune collections, which can make it very slow or even hang it completely.

If you are a beginner, or if you need to display music in web pages, or if you want to use the latest ABC features, then abc2svg is the best program for you.

**abcMIDI**

The abcMIDI suite focuses on MIDI output, and it also performs manipulations such as transposition and reformatting. Like abcm2ps, many ABC applications use abc2midi internally to play the music.

You will need `abcMIDI` programs if you want MIDI output.

**Abc/MusicXML Translators**

The Abc/MusicXML translators are the bridge between Abc and other music applications.

These tools are essential if you want to typeset and play Abc music with MusicXML-enabled programs. Some of these are renowned professional applications, like Sibelius® and Finale®, which may produce better output than `abcm2ps/abc2svg` and `abc2midi`. In addition, the Abc/MusicXML translators provide useful extensions to Abc such as tablatures and percussion notes.

Unfortunately, the conversion from/to MusicXML and Abc cannot be perfect, and some features may be lost. In particular, Abc tune collections cannot be converted, because the MusicXML file format has currently no notion of tune collection.

Use the Abc/MusicXML translators if you want to insert Abc music in MusicXML applications, or if you want to use Abc extensions.

**Other Programs**

As already explained, there are many other Abc applications. A number of them, like the aforementioned EasyABC, use `abcm2ps` and `abc2midi` internally to display and play the music. These applications are quite popular and are very easy to use, but there's a catch.

Users often ignore or forget that the real work is done under the hood by `abcm2ps` and `abc2midi`, which are bundled with the application. Often, these are old versions that should be upgraded to fix errors and/or add new features. Further, the application that employs `abcm2ps` and `abc2midi` may introduce its own errors. If you use such an application, take all this into account.

Please remember: *the most reliable way to make music in Abc is by using the latest versions of the command line programs, directly in the command line.* It's not difficult: command line usage will be explained in Section .

### 1.2.3 Abc in a Nutshell

Abc music is written in text files with extension `.abc` or `.abp`. The extension is not obligatory, but it's 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 called *decorations*, 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 *directives* like `%%titlefont` or `%%MIDI program 19`.

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

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

An Abc file may contain several pieces, divided by one or more blank lines. Each piece has its own header and body. Some fields and directives can also appear in the body of a piece; others may be written at the top of a file, so they apply to all pieces. The file containing Abc music is also called the *source* of the pieces.

Strictly speaking, directives for low-level details are not part of the notation. In fact, Abc was designed for a high-level description of musical pieces, with no special instructions for typesetting or sound output. That said, Abc 2 does provide directives for specifying such details; these will be examined in Chapter 4 (p. 57).

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 a *grave accent* ʻ.

- `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* ].

GNU/Linux and macOS users will have to find out how to type these characters.

### 1.2.4   Our First Abc File

To get our feet wet, we are going to use `abc2svg` to write, display and play the C major scale. The recommended browser is Chrome or its open source version Chromium, because it supports a handy "Save as PDF" print option.

There's a very important detail to stress before we begin. Whenever you write an Abc file, you'll have to choose a file name. Any file name will do, but:

> 🛑 **Avoid mistakes!**
> *do not use file names containing spaces!*

That is: instead of, say, `my music file.abc`, you must use `my_music_file.abc` or `My-MusicFile.abc`. This is very important, because command line programs cannot easily use file names containing spaces. Besides, file names containing spaces are not web-friendly either.

Let's start. Connect to the Internet, start your web browser and open the `abc2svg` page, http://moinejf.free.fr/js/edit-1.xhtml. The screen is split in two; the left-hand side is a large

text input area with a menu on the top; the right-hand side is blank, apart from the "(void)" string. This is the area where the score will be displayed.

Click on the text input area and type (or copy and paste) 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' |
```

When you are done, wait two seconds and you should see this score:



To save the source for later use, select File/Save file and input the file name `scale1.abc`. To convert the music to PDF, open the browser menu and select Print.... From the following dialog, change the Destination to Save as PDF, then click on Save and insert the name of the PDF file.

Now, let us make a mistake on purpose: insert the # character instead of the first bar |. An error message will be printed in red near the menu; click on it and an explanation will be displayed:

```
noname.abc:3:17 Error: Bad character '#'
```

Fix the error in the source, wait two seconds, and the error message will disappear.

**Using `abc2svg` Offline**

A portable, offline version of the `abc2svg` web editor is available from the ABC 2 page. It can be installed on any operating system, and it only requires a web browser to work; it could even be installed on a pendrive.

You may want to install it if you want to manage ABC music without an Internet connection.
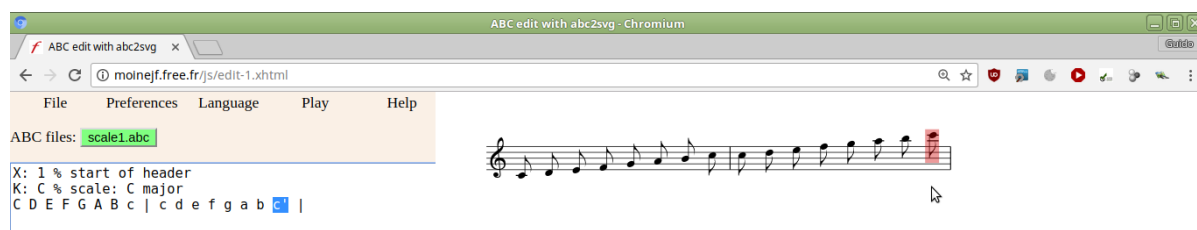


Figure 1.2: Writing a tune in a web browser using `abc2svg`.

### 1.2.5 Using the Command Line

Now we will use the command line interface to turn the C major scale to PDF and MIDI; I assume that `abcm2ps` and `abcMIDI` are installed.

Commands you must enter are printed in **boldface**. The following paragraphs will provide operating-system specific details; explanations for all users will follow.

**Windows**

Start the cmd program. (Windows XP: from Start/Run…, insert cmd, then click on "Ok". Windows 7 and later: 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>_
```

Let's check whether abcm2ps and abcMIDI were installed correctly. Type the following:

```
C:\Users\Guido>abcm2ps -V
abcm2ps-8.14.2 (2018-12-18)
Options: A4_FORMAT PANGO
Default format directory: /usr/share/abcm2ps
C:\Users\Guido>abc2midi -ver
abc2midi 4.07 May 02 2018 abc2midi
C:\Users\Guido>_
```

If you don't see the above output messages, there's an installation problem you will have to fix; please refer to the programs' documentation. Otherwise, you can proceed.

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 (that is, a "folder"), called abcmusic, where we will save our ABC files. Now, let's enter the directory:

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

This command is equivalent to a double click on the folder icon. Note that the command prompt changed to confirm that you've moved to that folder. Always 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
```

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

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

Save it, then switch back to the command prompt.

We are ready to make the PostScript output using abcm2ps:

```
C:\Users\Guido\abcmusic>abcm2ps -c -O= scale1.abc
abcm2ps-8.14.2 (2018-12-18)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 19965 bytes)
C:\Users\Guido\abcmusic>_
```

A bit of explanation may be helpful. `-c -O=` are options that tell abcm2ps to perform automatic line breaks and to give the PostScript output file the same name as the input file, but with a `.ps` extension.

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



To convert the score to PDF, select File/Save As… from Sumatra's menu, then select PDF documents in Save as Type.

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-8.14.2 (2018-12-18)
File scale1.abc
scale1.abc:3:16: error: 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, 19943 bytes)
C:\Users\user\abcmusic>_
```

Fix the error in the source, save it and convert it again. Finally, let's make a MIDI file:

```
C:\Users\user\abcmusic>abc2midi scale1.abc
4.07 May 02 2018 abc2midi
Warning in line-char 2-0 : No M: in header, using default
writing MIDI file scale11.mid
C:\Users\user\abcmusic>_
```

Double click on the MIDI file to listen to it.

**GNU/Linux, macOS**

Open a terminal window (Ubuntu: Applications/Accessories; Mac OS X: Applications/Utilities).

Let's check whether abcm2ps and abcMIDI were installed correctly. Type the following:

```
$ abcm2ps -V
abcm2ps-8.14.2 (2018-12-18)
Options: A4_FORMAT PANGO
Default format directory: /usr/share/abcm2ps
$ abc2midi -ver
abc2midi 4.07 May 02 2018 abc2midi
$ _
```

If you don't see the above output messages, there's an installation problem you will have to fix; please refer to the programs' documentation. Otherwise, you can proceed.

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 (that is, a "folder"), called abcmusic, where we will save our Abc files. Now, let's enter the directory:

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

This command is equivalent to a double click on the folder icon. The command prompt should change to confirm that you've moved to that folder. Always move to the abcmusic folder before writing tunes.

Now let's start an editor. In GNU/Linux systems, the default editor should be one of the following: xed, gedit, pluma, kate, mousepad, leafpad. If it is not, please find out. In macOS, the default editor is TextEdit; from Preferences, select Plain Text in the Format section. The file name we are about to create is scale1.abc.

```
~/abcmusic$ xed 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-8.14.2 (2018-12-18)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 19965 bytes)
~/abcmusic$ _
```

A bit of explanation may be helpful. -c -O= are options that tell abcm2ps to perform automatic line breaks and to give the PostScript output file the same name as the input file, but with a .ps extension.

To display the PostScript output, GNU/Linux users have a wide choice of viewers; use xdg-open if you're in doubt. macOS has a Preview utility that will automatically open scale1.ps and convert it into PDF format; from the command line, use open.

```
~/abcmusic$ xdg-open 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.

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-8.14.2 (2018-12-18)
File scale1.abc
scale1.abc:3:16: error: 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, 19943 bytes)
~/abcmusic$ _
```

Fix the error in the source, save it and convert it again. Finally, let's make a MIDI file:

```
~/abcmusic$ abc2midi scale1.abc
4.07 May 02 2018 abc2midi
Warning in line-char 2-0 : No M: in header, using default
writing MIDI file scale11.mid
~/abcmusic$ _
```

Double click on the MIDI file to listen to it.

**Android (expert users only!)**

To use Abc software on Android, you will have to install a terminal emulator beforehand; please refer to the instructions available at the Abc 2 home page. Don't forget to install a text editor too.

Android users will follow almost the same procedure as GNU/Linux users. The difference is the lack of Ghostscript for the Android platform: you will have to use SVG output instead. By the way, this is a viable solution for all users, not only Android users.

Once your Abc source is ready, use this command line:

```
~/abcmusic$ abcm2ps -c -O= -X scale1.abc
abcm2ps-8.14.2 (2018-12-18)
File scale1.abc
Output written on scale1.xhtml (1 page, 1 title, 5160 bytes)
~/abcmusic$ _
```

Note the `-X` switch: it tells abcm2ps to produce SVG output embedded in a file called `scale1.xhtml`. To view this file, just open it with a web browser.

Finally, let's make a MIDI file:

```
~/abcmusic$ abc2midi scale1.abc
4.07 May 02 2018 abc2midi
Warning in line-char 2-0 : No M: in header, using default
writing MIDI file scale11.mid
~/abcmusic$ _
```

Tap on the MIDI file to listen to it.

**Explanations, for everybody**

We started our session by running the two programs. In particular, `abcm2ps` reported its version number, release date, supported options, and default format directory. A4_FORMAT means that the European A4 paper format is supported; PANGO, when present, means that `abcm2ps` employs the external Pango library (http://www.pango.org) for the rendering of text; the default format directory may contain additional files that contain formatting parameters (see Section 4.1.1, p. 60).

Let's now examine the source thoroughly:

```
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' |
```

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. However, field lines must not begin with spaces, and spaces are not allowed between the field letter and the `:` character.

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 Latin notation.

| **English note** | C | D | E | F | G | A | B |
|---|---|---|---|---|---|---|---|
| **Latin note** | Do | Re | Mi | Fa | Sol | La | Si |

Table 1.1: Note names in English and Latin notation.

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 a single quote, which denotes an octave higher. Note that `abcm‐2ps`, 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.

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 strongly suggest that you do your exercises at the end of each of the following sections.

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.

♫ ♫ ♫ ♫ ♫

# 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, tuplets, chords, grace notes, and expression symbols. But before we begin, we have an important topic to examine.

### 2.1.1 Manual or Automatic Formatting?

How should we write the music? Should we take care of the exact number of measures in each line of music, or should we leave this job to the scorewriter?

To some extent, it depends on the type of music we want to write. Simple tunes of fixed length, 16 or 32 bars for instance, are probably better written specifying four measures per line. Most traditional music (jigs, reels, bourrées, etc.) falls into this category.

For longer pieces, such as choral or classical music, the best option is usually *automatic formatting*. In fact, it is usually better to concentrate on writing the music, and leave the typesetting job to `abcm2ps` or `abc2svg`. After all, a scorewriter is a specialised piece of software that knows how to place music elements optimally on the page. As a matter of fact, this is the way we work with word processors (or LaTeX, in my case): we let the program rearrange text lines automatically. Inserting every line break manually would be a pointless waste of time.

In other cases, it could be desirable to set the number of measures per line manually; Abc provides several ways to do so. However, in this manual all scores will be typeset using `abcm2ps` and the `-c` (automatic line breaks) option, unless otherwise specified.

### 2.1.2 Note 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 (p. 23).

`K:C treble`, besides specifying the C major scale, forces the treble clef. In this example it must be specified because there are several notes much below the staff; `abcm2ps` or `abc2svg` would set the bass clef by default. Also, note that each note is separated by a space, but there is no space between notes and commas or single quotes. Finally, note that the last measure is not complete (it lacks four eights); this is not a problem.

If you use `abcm2ps`, convert the source without the `-c` option:

```
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'' |
```
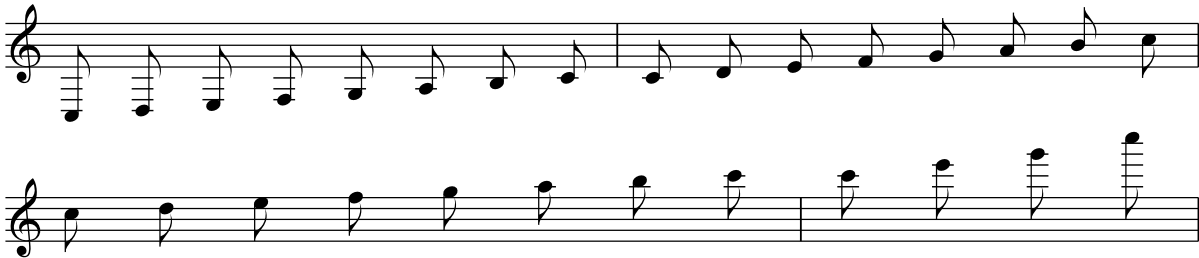


The rule is: if a note is followed by one or more commas, it goes down one or more octaves; if it is followed by one or more single quotes, it goes up one or more octaves. Experts will notice some resemblance to the Helmholtz pitch notation, shifted two octaves upwards.

It is important to note that 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.* However, automatic line breaks may be preferable. In the above example, for instance, note spacing is too wide; a single line would probably look better.

We can instruct abcm2ps and abc2svg to print the score with automatic formatting. This is done inserting the following line:

```
%%linebreak <none>
```

at the top of the source. This line is one of the many directives available that let us fine-tune the score; we will study them in Chapter 4, p. 57.

Converting the new source:

```
%%linebreak <none>
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'' |
```

we obtain this score:



which looks a bit compressed, but is definitely better than before.

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

```
%%linebreak <none>
X: 1
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'|
```



To sum up, these are the note names in the bass and treble clef:



## 2.1.3 Note Length: *L:*

Unless otherwise indicated, note length is automatically set depending on the tune metre.

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

The L: field is used to modify the default note length, specifying a value as in L:1/4. (To change the metre, we use the M: field; see Section 2.2.2, p. 28.)

To double, triple etc. the length of a note, we write the number 2, 3, etc. immediately after the note. To divide the value of a note by 2, 4 etc., we write /2, /4, /8... or, equivalently, /, //, ///... Spaces between the note and the digit or the slash are not allowed.

```
X: 1
L: 1/4
K: C
% 16/4  8/4  4/4  2/4   2/4   1/4  1/4 1/4 1/8 1/8
  C16 | C8 | C4 | D2    D2 | E0   E    E   F/   F/|
% 3/4   1/16  1/16  1/32  1/32  1/64  1/64  1/64  1/64
  G3    G//   G/4   G/8   G/8   G/16  G/16  G/16  G/16|
```
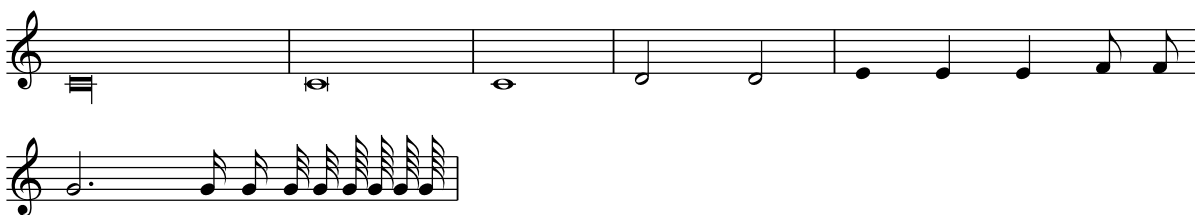
Note that abcm2ps and abc2svg support notes that are longer than a whole note! The first note in the above example is called a *longa*, and its length equals four whole notes. The second note is a *brevis*, equal to two whole notes. The spacing between notes is proportional to their length. (We shall see in Section 4.1 (p. 57) how to make the spacing of notes constant instead of proportional.)

Another weird note is the first E in measure 5. The trailing 0 denotes a stemless quarter note, which is sometimes found in early music.

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

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



### 2.1.4   Rests and Spacing: *z  Z  x  y*

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

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

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



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

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



The y character 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 ≈ 0.353 mm.

Invisible rests are often used for transcribing piano music; examples will be shown in Section 3.1.2 (p. 48).

### 2.1.5   Accidentals: ˆ _ =

Sharp ♯ is denoted by a caret ˆ before the note, flat ♭ by an underscore _ and natural ♮ by an equal sign =. Spaces between the accidental and the note are not allowed. Double accidentals are indicated doubling the special character: double sharp by ˆˆ and double flat by __:

```
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 |
C ^^C __C z |
```



Quarter tone accidentals (microtones) are also available. That is, 1/4 and 3/4 sharps and flats can be obtained adding / after the caret or the underscore, while 3/4 are obtained adding 3/2:

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



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

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



### 2.1.7  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|]
```



> 🛑 **Avoid mistakes!**
>
> Although ties and slurs are graphically very similar, they have a completely different musical meaning. Do not use ties to connect notes of different pitch: MIDI output (see Section 6.1, p. 91) would be wrong!

Dashed slurs and ties are indicated by a dot before the open parentheses or hyphen character. 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|]
```



## 2.1.8 Tuplets: (*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
L: 1/8
K: C
(3cde e2 | (6cegczg (3czg |
(3:2:2G4c2 | (3:2:4G2A2Bc | (3:2:6(3GGGA2Bc |]
```
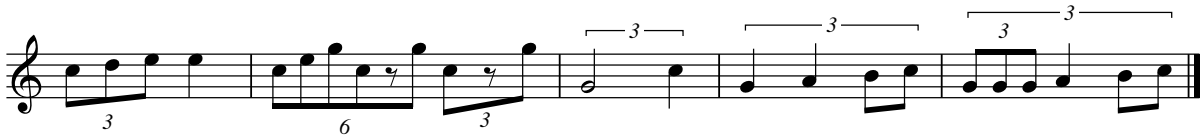
To fine-grain the printing of tuplet indications, please see Section 4.2.11 (p. 71).

> ☞ **Warning**
>
> abcMIDI and the Abc/MusicXML translators do not support nested tuplets.

### 2.1.9  Chords: *[]*

Chords are written enclosing notes in square brackets; spaces between brackets and notes are not allowed. A chord behaves as a single note when it comes to adding dots, slurs, etc. That is, a chord can be preceded by a dot for staccato, or by a symbol, and so on. The same holds for note length: instead of specifying the length of each note, we can simply add a number after the closing bracket.

```
X: 1
L: 1/4
K: C
CE [C2G] c | .[CEGc] ([C/E/G/c/][E/G/c/e/]) [C2D2G2c2] |
[CEGc] > [CEGc] [CEGc] > [CEGc]| [C2-E2-G2-][CEG]2 |
```

   Note the chord in the first measure. While abcm2ps and abc2svg print chords containing unequal notes as they are, abcMIDI extends shorter notes to match the longer ones.

> 🛑 **Avoid mistakes!**
>
> Do not mistake chords for something completely different! If you want to obtain something like this:
>
> you should be aware that these are not chords, but different *voices* on the same staff. Voices will be the subject of Section 3.1 (p. 45).

### 2.1.10  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:; it is always 1/8 for a single grace note, 1/16 for two or more, and 1/32 in bagpipe tunes. (Yup, abcm2ps and abc2svg support keys and directives for Great Highland Bagpipe music.)

Chords can also be specified instead of a single grace note. However, abc2midi can't play these "grace chords" correctly.

```
X: 1
L: 1/4
K: C
~c2 {/d}c {c2d2}c|{d/c/d/}c {ede}c {fef}c c|
c/{[ceg][gc]}c/c/c/ c/c/{gfedc}c/c/| c2 c {cBAGFED}C|
```
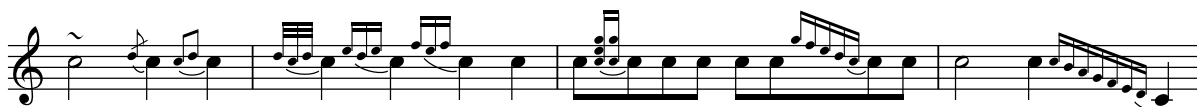


To remove the slur joining grace notes to the main note, we use the -G option of abcm2ps, or a formatting parameter that we'll see later.

### 2.1.11 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:

```
X: 1
L: 1/4
K: C
CDEF|GABc|[r:this is a note][K:D][L:1/8] DEFG ABcd|
```



The new r: field, as you might have guessed, is used to include remarks within the music.

## 2.2 Music Properties

### 2.2.1 Key signatures and Clefs: *K:*

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

**Key Signatures**

`K:` must be followed by a note name in upper case, followed by `m` or `min` when the mode is minor. Accidentals are written as `#` for ♯ and `b` for ♭; e.g. `Bb`, or `F#m`. When there are no accidentals, the keyword `none` may also be used.

I remind you the simple rule to find out the major key in accordance with the number of sharps or flats: *one tone* higher than the last sharp note, or *one fourth* below the last flat note.

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

Western classical music only uses major and minor modes, but other modes 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 2.1 lists them all.

| Accidental | Major (Ionian) | Minor (Aeolian) | Mixolydian | Dorian | Phrygian | Lydian | Locrian |
|---|---|---|---|---|---|---|---|
| 7 sharps | C# | A#m | G#Mix | D#Dor | E#Phr | F#Lyd | B#Loc |
| 6 sharps | F# | D#m | C#Mix | G#Dor | A#Phr | BLyd | E#Loc |
| 5 sharps | B | G#m | F#Mix | C#Dor | D#Phr | ELyd | A#Loc |
| 4 sharps | E | C#m | BMix | F#Dor | G#Phr | ALyd | D#Loc |
| 3 sharps | A | F#m | EMix | BDor | C#Phr | DLyd | G#Loc |
| 2 sharps | D | Bm | AMix | EDor | F#Phr | GLyd | C#Loc |
| 1 sharp | G | Em | DMix | ADor | BPhr | CLyd | F#Loc |
| none | 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 2.1: Modal scales.

*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. (The keyword `exp` may be omitted, but since `abc2midi` needs it, don't leave it out if you need MIDI output.) 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 `K:` field to indicate the number of accidentals: `K:AMix % 2 sharps`

**Clefs**

As we saw in Section 2.1.2 (p. 15), the clef is automatically set by `abcm2ps` and `abc2svg` depending on the notes pitch. For example, if we start a tune with notes much below the staff

(notes with commas), abcm2ps or abc2svg will select the bass clef. However, we can set the clef with the K: field at the start of the tune; we can also opt to have no clef at all.

The complete syntax of the K: field is:

K: ⟨*key*⟩ *[clef=*⟨*clef type*⟩*] [line number] [octave=*⟨*number*⟩*] [transpose=*⟨*number*⟩*] [+8] [-8] [ˆ8] [_8] [stafflines=*⟨*number*⟩*] [staffscale=*⟨*number*⟩*] [cue=*⟨*on/off*⟩*]*

Parameters in ⟨*angular brackets*⟩ are required, while those in *[square brackets]* are optional. The only required parameter is the key specified after the K: field. All others are optional:

- *[clef=]* may be omitted before the clef type when it is not none;

- ⟨*clef type*⟩ can be:

    - the ABc name of a note: 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 keyword none indicates that there is no clef;

    - the keyword perc (or, equivalently, the letter P) indicates a clef for percussion instruments.

- *[line number]* indicates the staff line on which the clef is drawn;

- *[octave=*⟨*number*⟩*]* transposes the music by ⟨*number*⟩ octaves;

- *[transpose=*⟨*number*⟩*]* (or *[t=*⟨*number*⟩*]*) currently does nothing, but is supported by abcMIDI for MIDI output transposition;

- *[+8]* and *[-8]* print 8 above or below the clef;

- *[ˆ8]* and *[_8]* print 8 above or below the clef, and perform octave transposition;

- *[stafflines=*⟨*number*⟩*]* sets the number of lines of the associated staff;

- *[staffscale=*⟨*number*⟩*]* sets the staff scale. Default is 1, maximum value is 3, minimum is 0.5;

- *[cue=*⟨*on/off*⟩*]* sets the music scale to 0.7 (on) or 1.0 (off). The staff scale is unchanged.

To sum up, available clefs and the respective fields are listed in Table 2.2 and in the following example:

```
X: 1
L: 1/4
K: C clef=none
CEGc | [K:C treble] CEGc |[K:Cm bass]C,E,G,C |
w: none | treble | bass |
[K:C bass3]C,E,G,C | [K:Cm alto4]CEGc| [K:C alto]CEGc |
w: baritone | tenor | alto |
[K:Cm alto2]CEGc | [K:C alto1]CEGc | [K:perc] cdef |]
```

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

Table 2.2: Clefs and associated K: fields.

```
w: mezzosoprano | soprano | percussions |
```



Note the difference between the +8/-8 and ^/_8 in clefs indications. The latter prints 8 above or below the clef, and also transposes the notes one octave lower or higher:

```
X: 1
L: 1/4
K: C
[K:C treble+8] CEGc | [K:C treble-8] CEGc |
w: +octave\ indication | \-octave\ indication |
[K:C treble^8] CEGc | [K:C treble_8] CEGc |
w: +octave\ and\ transposition * | \-octave\ and\ transposition * |
```

The \ characters in the example above are used to insert the next character literally, avoiding its special meaning; for example, \- will not produce a new syllable. (More details in Section 2.2.4, p. 30.)

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

```
X: 1
L: 1/4
K: C treble
%
CDEF|GABc||
K:C clef=bass octave=-2
CDEF|GABc||
K:C clef=alto octave=-1
CDEF|GABc||
K:C clef=G octave=1
CDEF|GABc||
```



**Changing the Clef: I:clef**

When we want to change the clef but not the key signature, we can use a shortcut. Instead of writing:

```
K:C clef=bass
```

we can write the following field:

```
I:clef bass
```

which can also used inline. Note that we did not write =. Clefs types are the ones listed in Table 2.2 (p. 26).

**Bass and Alto Clefs Compatibility Issues**

Old 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 method is now disallowed. However, problems may occur when we come across files that were written for the old methond. We could see that some notes are printed one or two octaves higher than expected.

There are two recommended ways to deal with these files:

- manually transpose every note to the right octave, i.e. change c to C,;

- add a clef specifier and/or an octave modifier to the K: field.

The first option is formally more correct; the second one is more practical.

The most common case is given by notes written for the bass clef that are printed two octaves higher. Fix them by changing:

```
K: C clef=bass
```

to:

```
K: C clef=bass octave=-2
```

The same method can be used to transpose music for a different clef. For instance, let's suppose we wanted to transpose some music from the treble to the alto clef; that is, we transpose music from violin to viola. We can do that by changing the K: field, from

```
K:D
```

to:

```
K:D clef=alto octave=-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 also supported: M:o (perfect minor), M:o. (perfect major), M:c (imperfect minor), M:c. (imperfect major);

- 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, we insert an inline M: field in the body:

```
X: 1
M: C
K: C
L: 1/4
C D E F |[M: 3/4] G A B |[M: 2/4] c c|
[M:7/8 (3/8 2/8 2/8)] [L:1/8] CDE FG AB |c2 z4 z|
```

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

### 2.2.3 Bars, Repeats, Endings: | / : [ ]

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 |]
```



Note that `[|]` doesn't print anything: it is an *invisible bar*, and it can be used as a place-holder for a decoration. The same can be accomplished using `[]`. Also, note that `:` is the same as `.|`.

To indicate that a section has two (or more) different endings, 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` and `abc2svg` also support 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 2.2.11 (p. 41) we will see how to include `|:` in the source, but hide it in printed output.

> ☞ **Warning**
>
> Unfortunately, repeats are one of the features that make `abcm2ps/abc2svg` and `abc-MIDI` not compatible with each other. In the latter,
>
> - 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   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 we add lines that start with the `W:` (upper case) field, followed by the lyrics.

Stanzas are separated by an empty `W:` line, i.e. not followed by any words. Stanzas are automatically arranged in columns so as to take up less space; this can be avoided using `W:\` instead of `W:`.

```
X: 1
L: 1/4
K: C
CDEF|GFED|C4|
%
W: La la la la la la la la laaaaa
W:\ % keep on the same column
W: Na na na na na na na na naaaaa
W: % print on a new column
W: La la la la la la la la laaaaa
```



La la la la la la la la laaaaa                          La la la la la la la la laaaaa

Na na na na na na na na naaaaa

As shown by the source above, an Abc tune may also consist of `W:` lines only, even without notes to provide the melody! As long as we provide `X:` and `K:`, in fact, a tune is defined.

Note-aligned lyrics are obviously more complex to write. Immediately after a line of music, we write one or more lines that start with the `w:` (lower case) field, followed by the lyrics split in syllables. Obviously, rests are not matched by lyrics.

Alignment rules between notes and syllables are:

- the – character (hyphen) separates syllables of a word. If it is separated from the previous syllable by a space, a note is skipped. $\langle n \rangle$ – characters separated from the previous syllable skip $\langle n \rangle$ notes. Spaces after the – have no effect, and can be used to improve 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 (insert \* to print a literal * character);

- ~ (tilde) joins two syllables under a note;

- \- inserts a – character.

If a w: line grows too long, it can be broken and continued on the following line, which will begin with the +: field. These two lines will match the notes on the previous melody line.

The following tune is "Happy Birthday" with Italian lyrics:

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



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

Beware of the difference between – (hyphen) and _ (underscore). Usually, – is used to skip syllables *within* a word, while _ is used *at the end* of a word. Let's see a Gregorian chant example, the Kyrie from Missa de Angelis:

```
X: 1
L: 1/8
K: D
%
z D FGA A2 BAG A2 dBAG AB A2  AFED GFE E D2|
w: Ky-ri --e, _____ \* e ----- le-i-son.
```
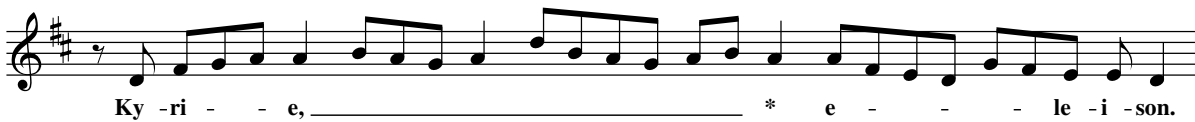
If a `w:` line contains digits, these will not be aligned with the notes but moved a bit to the left; this feature is used to enumerate subsequent `w:` lines. The digit must be followed by a `~` and joined with the following syllable. If we want to align digits with notes (i.e. for fingerings), all we have to do is insert a `~` character just before the number.

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



Take special care to write a number of syllables that matches the number of notes! Mismatch between notes and syllables is one of the most common causes of error. Besides, please bear in mind that lyrics may alter the spacing between notes. This effect is especially pronounced when a whole word, or a very long syllable, is forced under a note.

## 2.2.5   Title, Composer, Tempo: `T:` `C:` `Q:`

Our scores still miss a few details. In the next example we introduce the `T:` (title, subtitle), `C:` (composer) and `Q:` (tempo) fields:
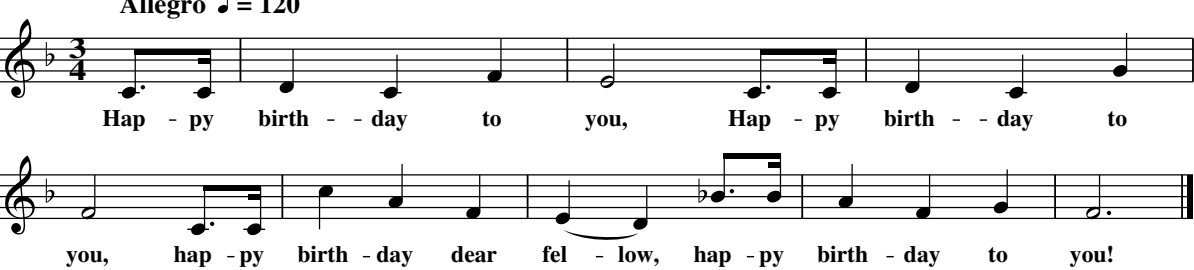
```
X: 1
T: Happy Birthday To You   % title
T: (Good Morning To All)   % subtitle
C: Patty Hill, Mildred J. Hill % composer
M: 3/4
Q: "Allegro " 1/4 = 120 % tempo
K: F
C>C | D2C2F2 | E4 C>C | D2C2G2 | F4 C>C |
w: Hap-py birth-day to you, Hap-py birth-day to you, hap-py
c2A2F2 | (E2D2) _B>B | A2F2G2 | F6 |]
w: birth-day dear fel-low, hap-py birth-day to you!
```

Happy Birthday To You

(Good Morning To All)

*Patty Hill, Mildred J. Hill*



The text indication in the `Q:` field ("Allegro" in our example) may be omitted. Besides, `Q:` also accepts references to previous tempo indications, as in `Q: 3/8 = 1/4`.

In Section 4.1 (p. 57) we will learn how to change the title fonts.

### 2.2.6 Foreign Characters

Writing titles or lyrics in English is straightforward, but if we want to write text in European languages like Italian, German, Hungarian etc., we will need accented characters that don't appear on most keyboards. How can you type "Crêuza de mä"? The problem is even more complicated if we want to write lyrics in non-latin alphabets, such as Russian or Greek for instance.

Many characters of foreign languages can be inserted by using special character sequences that begin with a backslash \, followed by a special character, then followed by the character to be modifiedaltered. For example, to get the "a grave" character à we type \`a. These peculiar strings are called *backslash sequences*, and are listed in Table 2.3. Note the difference between the acute ' (ASCII 39) and grave ` (ASCII 96) accents.

| Accent | Special character | Followed by |
|---|---|---|
| grave | \` | AEIOUaeiou |
| acute | \' | AEIOUYaeiouySZszRLCNrlcn |
| circumflex | \^ | AEIOUaeiouHJhjCGScgs |
| cedilla | \, | CcSsTtRLGrlgNKnk |
| umlaut | \" | AEIOUYaeiouy |
| tilde | \~ | ANOanoIiUu |
| ring | \o | AaUu |
| macron/stroke | \= | ADEHIOTUadehiotu |
| slash | \/ | OoDdLl |
| ogonek | \; | AEIUaeiu |
| caron | \v | LSTZlstzCEDNRcednr |
| breve | \u | AaEeGgIiOoUu |
| long Hungarian umlaut | \: | OUou |
| dot | \. | ZzIiCcGgEe |
| ligatures | \ae \oe \ss \ng | n/a |

Table 2.3: How to obtain characters of foreign languages.

Typing accented letters using backslash sequences is the best option if you don't need other foreign characters. Besides, your Abc source will be encoded in 7–bit pure ASCII, which is guaranteed to be readable by every digital device on the planet.

However, there are many more foreign characters you might need, not to mention whole foreign alphabets. In this case, you will need to write your sources using a *character encoding* called UTF-8. Simply put, UTF-8 is a computer standard that makes all possible characters and symbols available to applications.

UTF-8 is fully supported by abcm2ps (with Pango support), abc2svg, and the Abc/Music-XML translators. If you're sure that your text editor saves its files in UTF-8 encoding, you can use accented letters and symbols freely; possibly, by copy & paste.

### 2.2.7   UTF-8 Characters and Symbols

The site http://www.utf8-chartable.de lists all UTF-8 characters, grouped in many different *blocks*. Each character is identified by a number called *Unicode code point*; for example, the code of character © in block Basic Latin (U+0000–U+007F) is U+00A9.

When compiled with Pango support, abcm2ps can print all UTF-8 characters. There are three ways to insert UTF-8 characters in an Abc source:

1. directly, possibly pasting it from a web page. I suggest that you use the virtual keyboards at http://www.typeit.org/.

2. using the *Javascript syntax*: replace the leading U+ with \u, followed by the character code (case insensitive). For example, character U+00A9 can be entered as \u00A9. This method is limited to the first 65535 (U+FFFF) UTF-8 characters.

3. using the *XML character entity reference*, or *XML syntax* for short: replace the leading U+ with &#x, followed by the character code (case insensitive), then by a semicolon ; . For example, character U+00A9 can be entered as &#x00A9;

Copying and pasting UTF-8 characters in the source is the best method; if you can't, you should use the XML syntax:

```
X: 1
T: XML Characters
K: C
%
%  Yes, no music!
%
W: A few strange characters in XML:
W:\
W: Grinning face: &#x1F600; (U+1F600)
W: Ace of hearts: &#x1F0B1; (U+1F0B1)
W: G clef: &#x1D11E; (U+1D11E)
W: Porrectus: &#x1D1D9; (U+1D1D9)
W: Aleph: &#x05D0; (U+05D0)
W: Omega: &#x03A9; (U+03A9)
W: Question mark as UTF-8: \u003f (U+003F)
W:\
```

# XML Characters

A few strange characters in XML:

Grinning face: 😀 (U+1F600)
Ace of hearts: 🂱 (U+1F0B1)
G clef: 𝄞 (U+1D11E)
Porrectus: 𝇙 (U+1D1D9)
Aleph: א (U+05D0)
Omega: Ω (U+03A9)
Question mark as UTF-8: ? (U+003F)

If your version of `abcm2ps` lacks Pango support, you can still use a pretty large number of foreign characters. In fact, `abcm2ps` recognises a set of *octal codes* in the form `\nnn`, where nnn are three digits. Inserting an octal code in the source we get the corresponding character; codes are listed below. For the curious: these are a subset of the Basic Latin block (`U+0000`–`U+007F`), with 5 musical symbols thrown in. Octal codes, however, are not recognised by abc2svg and by the Abc/MusicXML translators.

| | | | | |
|---|---|---|---|---|
| \201: ♯ | \202: ♭ | \203: ♮ | \204: 𝄪 | \205: 𝄫 |
| \241: ¡ | \242: ¢ | \243: £ | \244: ¤ | \245: ¥ |
| \246: ¦ | \247: § | \250: ¨ | \251: © | \252: ª |
| \253: « | \254: ¬ | \256: ® | \257: ¯ | \260: ° |
| \261: ± | \262: ² | \263: ³ | \264: ´ | \265: µ |
| \266: ¶ | \267: · | \270: ¸ | \271: ¹ | \272: º |
| \273: » | \274: ¼ | \275: ½ | \276: ¾ | \277: ¿ |
| \300: À | \301: Á | \302: Â | \303: Ã | \304: Ä |
| \305: Å | \306: Æ | \307: Ç | \310: È | \311: É |
| \312: Ê | \313: Ë | \314: Ì | \315: Í | \316: Î |
| \317: Ï | \320: Ð | \321: Ñ | \322: Ò | \323: Ó |
| \324: Ô | \325: Õ | \326: Ö | \327: × | \330: Ø |
| \331: Ù | \332: Ú | \333: Û | \334: Ü | \335: Ý |
| \336: Þ | \337: ß | \340: à | \341: á | \342: â |
| \343: ã | \344: ä | \345: å | \346: æ | \347: ç |
| \350: è | \351: é | \352: ê | \353: ë | \354: ì |
| \355: í | \356: î | \357: ï | \360: ð | \361: ñ |
| \362: ò | \363: ó | \364: ô | \365: õ | \366: ö |
| \367: ÷ | \370: ø | \371: ù | \372: ú | \373: û |
| \374: ü | \375: ý | \376: þ | \377: ÿ | |

Special characters and symbols can obviously be used in all string instances (titles, composer name, annotations, etc.) as we will see in the following sections.

For your convenience, this table contains many characters used in several European languages that you can copy and paste in your sources:

```
Grave:          À È Ì Ò Ù à è ì ò ù
Acute:          Á É Í Ó Ú Ý á é í ó ú ý
Acute:          Ś Ź ś ź Ŕ Ĺ Ć Ń ŕ ĺ ć ń
Circumflex:     Â Ê Î Ô Û â ê î ô û
Circumflex:     Ĥ Ĵ ĥ ĵ Ĉ Ĝ Ŝ ĉ ĝ ŝ
Cedilla:        Ç ç Ş ş Ţ ţ Ŗ Ļ Ģ ŗ ļ ġ Ņ Ķ ņ ķ
Umlaut:         Ä Ë Ï Ö Ü Ÿ ä ë ï ö ü ÿ
Tilde:          Ã Ñ Õ ã ñ õ Ĩ ĩ Ũ ũ
Ring:           Å å Ů ů
Macron:         Ā Ð Ē Ħ Ī Ō Ŧ Ū ā đ ē ħ ī ō ŧ ū
Slash:          Ø ø Ð đ Ł ł
Ogonek:         Ą Ę Į Ų ą ę į ų
Caron:          Ľ Š Ť Ž ľ š ť ž Č Ě Ď Ň Ř č ě ď ň ř
Breve:          Ă ă Ĕ ĕ Ğ ğ Ĭ ĭ Ŏ ŏ Ŭ ŭ
Hungumlaut:     Ő Ű ő ű
Dot:            Ż ż İ ı Ċ ċ Ġ ġ Ė ė
Other:          Æ æ ð ŋ Œ œ ß Þ þ
```

> ✍ **Note**
>
> Nowadays the UTF-8 standard is ubiquitous, but a few years ago other encodings were quite common. In particular, many European languages employed the Latin ISO-8859 encoding, and several Abc files available for download are encoded with it. These files should be converted to UTF-8 to be displayed and converted correctly.
>
> The online converter at https://www.charset.org/utf8-to-latin-converter lets you convert from/to ISO-8859-1 and UTF-8. Advanced users will want to use the `iconv` command-line program. This command converts a file encoded in ISO-8859-1 to its equivalent encoded in UTF-8:
>
> ```
> $ iconv -f ISO_8859-1 -t UTF-8 file_8859.abc > file_utf.abc
> ```

### 2.2.8  Parts: *P:*

Some tunes are made of different parts, possibly repeated in several ways. To specify the order in which parts are played, we use the P: field. 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 uppercase single 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|]
```

| Type | Meaning |
|---|---|
| `m` or `min` | minor |
| `maj` | major |
| `dim` | diminished |
| `+` or `aug` | augmented |
| `sus` | suspended |
| 7, 9, … | seventh, ninth, etc. |

Table 2.4: Types of accompaniment chords.

## 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:ABAB-ABCDCDCDABABABCDCDCD`. Try and count carefully!

> 🛑 **Avoid mistakes!**
>
> `P:` fields cannot be inserted within repeats (i.e. `|:...:|`), because `abc2midi` gets confused and produces wrong MIDI output.

### 2.2.9 Accompaniment Chords: `""`

In many songbooks, accompaniment chords (say, for the guitar) are notated as "A", "C7", "Dm", "F♯" etc. above the staff. These chords are notated in Abc by writing the chord name between double quotes `"` immediately before the note, using this format:

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

The note is A…G (uppercase only), the accidental is indicated with # for ♯, b for ♭, or = for natural; the chord type is one of those listed in Table 2.4. 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 6.1.9 (p. 97) for explanations.

```
X: 1
T: Happy Birthday with chords
M: 3/4
```

```
Q: "Allegro" 1/4 = 120 % tempo
K: F
C> C|"F"D2C2F2|"C"E3 z C> C|"C"D2C2G2|"F"F3 z C> C|
w: Hap-py birth-day to you, Hap-py birth-day to you, hap-py
"F"c2A2F2|"Bb"E2D z B> B|"F"A2F2"C"G2|"F"F6|]
w: birth-day dear fel-low, hap-py birth-day to you!
```

Happy Birthday with chords



## 🛑 Avoid mistakes!

If you need to write accompaniment chords using notes with Italian names, 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 7.4 (p. 112).

Multiple chords per note are possible. They can be notated writing two or more consecutive chords before the same note, or using the character ; to separate the chords:

```
X: 1
L: 1/4
K: C
"C""G"CCCC|"G""G7"GGGG|"C;C7"CCCC|]
```



## 2.2.10   Text Annotations: `"^_ <>@"`

Text annotations can be added in different ways. The first method is to write the annotation as an accompaniment chord, enclosing the text between double quotes but preceding it by a special character. Another method is to use the Q: field, which 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:

- ˆ 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 with the origin at the centre of the note head (the lowest note, if in a chord), with an offset of X horizontal and Y vertical points.

When we use @, it is useful to know that the distance between staff lines is 5 pt.

To include accidentals in text annotations, use \#, \b, and \=; note the leading \. Multiple annotations are notated like multiple chords, as seen in Section 2.2.9 (p. 37).

Let's see an example that uses all methods:

```
X: 1
Q: "Dolcemente" 1/4=60
L: 1/4
K: C
"_below"CEG"^above"c |
"<left"a'">right"A,"^around the note" "<(" ">)"A2 |
[Q: "sostenuto"] "_one" "_two" "_three" CEG"@-15,8.5 anywhere"c|
```



> 🛑 **Avoid mistakes!**
>
> A common mistake is writing text annotations leaving out one of the ˆ_ <>@ characters. This "annotation" will be misinterpreted as an accompaniment chord, which will be almost certainly malformed and will wreak havoc in both printed and MIDI output.

### Courtesy Accidentals

Courtesy and editorial accidentals are simply a special case of text annotation; the only problem is how to insert the accidental we need.

Since accidentals and many more musical symbols are defined as UTF-8 characters, we can insert them in XML syntax (see Section 2.2.7, p. 34) if abcm2ps has Pango support. In this case, we can use all musical characters defined in block U+1D100–U+1D1FF. Accidentals codes are:

- &#x266f; for sharp;

- `&#x266d;` for flat;

- `&#x266e;` for natural;

- `&#x1d12a;` for double sharp;

- `&#x1d12b;` for double flat.

We can also use a simpler notation that works even without Pango support:

- strings `"\b"` `"\#"` `"\="` print ♭ ♯ ♮ above the staff. Other characters in the string are not allowed;

- codes `\001 \002 \003 \004 \005` print sharp, flat, natural, double sharp, and double flat;

- codes `\201–\205` are equivalent to `\001–\005`.

```
X: 1
M: none
L: 1/4
K: C
%
"\b"G "\#"G "\="G
"^(\001)"G "^(\002)"G "^(\003)"G "^(\004)"G "^(\005)"G|]
w: \201 \202 \203 \204 \205
```



### Figured Bass

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

```
X: 1
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" "_\001" "_\0026;\002" "_6+" "_6-" "_-" "_\\6" "_/6" "_6 4"
```
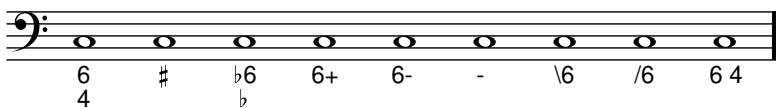


As you can see, all annotations have a leading ‿ (underscore) to print them below the staff.

### 2.2.11  Decorations: `!symbol!`

Decorations, also known as "embellishments" or "ornaments", are notated using the general form !symbol!: the decoration name, like **ff**, **ppp**, *cresc.…*, enclosed in exclamation marks. Decorations are written immediately before the note; two or more decorations per note are possible. Most decorations can also be applied to grace notes, rests, invisible rests, and measure bars.

```
X: 1
L: 1/4
K: C
!p!GG!>!!f!GG!segno!|GG!trill!G2!coda!|G4!fine!|
```
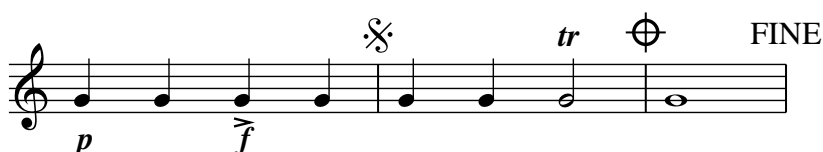


As shown above, some symbols are printed above the staff by default, while others are printed under the staff. Manual positioning is possible, as explained in Section 4.2.9 (p. 68).

Supported decorations are listed in Figure 2.1. You may notice that some symbols you need are missing. Don't worry: new symbols can be easily added, as we will see in Chapter 7 (p. 107).

Some symbols have synonims:

- !>!: !accent! or !emphasis!

- !+!: !plus!

- !mordent!: !lowermordent!

- !uppermordent!: !pralltriller!

- !<(! and !<)!: !crescendo(! !crescendo)!

- !>(! and !>)!: !diminuendo(! !diminuendo)!

- !rbstop!: !rbend!

If the tune contains many decorations, an alternative notation may be handy. After a line of music, we write a line that starts with the s: field. This line will contain only symbols.

Rules for matching notes and symbols are the same as those explained in Section 2.2.4 (p. 30). 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! |
```

## Decorations on single notes



!^! !>! !/! !//! !///! !+! !0! !1! !2! !3! !4! !5! !breath! !coda! !D.C.!

!dot! !downbow! !D.S.! !f! !fermata! !ff! !fff! !ffff! !fine! !gmark!

!invertedfermata! !invertedturn! !invertedturnx! !invisible! !marcato!

!mf! !mordent! !mp! !open! !p! !ped! !ped-up! !pp! !ppp! !pppp!

!roll! !segno! !sfz! !slide! !snap! !stemless! !tenuto! !thumb! !trill! !turn!

!turnx! !upbow! !uppermordent! !wedge!

## Decorations on multiple notes

!~(! !~)! !<(! !<)! !>(! !>)! !-(! !-)! !8va(! !8va)! !8vb(! !8vb)!

!beam-accel! !beam-rall! !trem1! !trem2! !trem3! !trem4! !trill(! !trill)!

!longphrase! !mediumphrase! !shortphrase!

## Special decorations

!arpeggio! !beambr1! !beambr2! !beamon! !rbstop! !xstem!

Figure 2.1: Standard decorations.

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 decoration, which is ended by the respective symbol with a closed parenthesis ). For example, `!<(!` starts a crescendo hairpin, `!<)!` ends it. The same holds for `!>(!`, `!>)!`, and their long-name counterparts;

- `!beambr1!` and `!beambr2!` act on notes connected by beams, leaving only 1 or 2 beams;

- `!beamon!` prevents a beam from being broken across measures;

- `!rbstop!` stops a repeat bracket;

- `!invisible!` makes the next note or measure bar invisible;

- finally, `!xstem!` is used in piano music, and it will be explained in Section 4.2.7 (p. 66).

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

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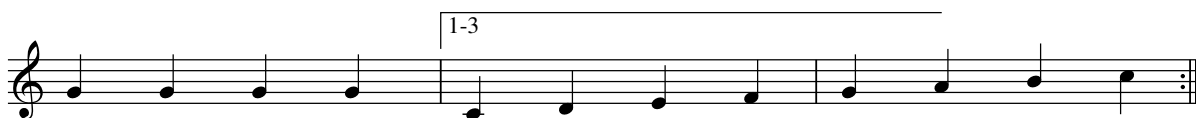


Piano symbols for "ottava alta" and "ottava bassa" indications automatically transpose the printed notes one octave lower or higher:

```
X: 1
L: 1/4
M: C
K: C
%
CEGc|!8va(! CEGc !8va)! |
cGEC|!8vb(! cGEC !8vb)! |
```



## 2.2.12   Redefinable Symbols: *U:*

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

The field is followed by an uppercase letter from H to Y or by a lowercase 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!`:

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

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

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

Letters u v T H L M P S O are predefined abbreviations for common symbols:

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

♫ ♫ ♫ ♫ ♫

# Chapter 3

# Harmony

## 3.1 Polyphony in Aʙᴄ

Iɴ previous sections we dealt with simple melodies, i.e. music written for a single voice or instrument. This is what old version 1.6 of Aʙᴄ could do, and it is actually a lot; many folk musicians do not need anything more.

We now turn our attention to Aʙᴄ 2 and its extensions for polyphonic music, using choral and piano pieces for our examples.

### 3.1.1 Voices and Systems: *V:*

Let's 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 a piece is called a *system*.

> ✎ **Note**
>
> abcm2ps allows to typeset music for up to 32 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: F
M: 4/4
V:1
FGAF|FGAF|ABcz|ABcz|c/d/c/B/ AF|c/d/c/B/ AF|
V:2
z4 |z4 |FGAF|FGAF|ABcz |ABcz |
V:1
```

```
GCFz |GCFz |z4 |z4 |
V:2
c/d/c/B/ AF|c/d/c/B/ AF|GCFz|GCFz|
```

## Brother John

This score was written alternating the lines of voices 1 and 2, as in real sheet music. We could write all of the music of voice 1, then all of voice 2: the resulting score would be printed correctly, but the source would be much less readable.

> ✎ **Note**
>
> When writing multiple voices:
>
> • the P: field must be indicated in the top voice only;
>
> • slurs and decorations on multiple notes must start and stop in the same voice.

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

V: ⟨*voice name*⟩ *[clef=* ⟨*clef type*⟩*] [name=* ⟨*string*⟩*] [sname =*⟨*string*⟩*] [merge] [stem=* ⟨*up* | *down* | *auto*⟩*] [gstem=* ⟨*up* | *down* | *auto*⟩*] [dyn=* ⟨*up* | *down* | *auto*⟩*] [lyrics=* ⟨*up* | *down* | *auto*⟩*] [gchord=* ⟨*up* | *down* | *auto*⟩*] [scale=* ⟨*n*⟩*] [staffscale=* ⟨*n*⟩*] [stafflines=* ⟨*n*⟩*] [cue=* ⟨*on* | *off*⟩*]*

The voice name may be a digit or a string (e.g. "Tenor"). Possible parameters are:

• `clef=` specifies the clef of the voice; we use the same parameters examined in Section 2.2.1 (p. 23);

• `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,

- gstem=⟨*up*⟩, ⟨*down*⟩ or ⟨*auto*⟩ forces the grace note stem direction;

- dyn=⟨*up*⟩, ⟨*down*⟩ or ⟨*auto*⟩ forces the placement of dynamics 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;

- cue=⟨*on*⟩ or ⟨*off*⟩ sets the current voice as "cue" (i.e. a voice for hints, printed smaller) or normal voice.

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="C"
K: F
%
[V:1] FGAF|FGAF|ABcz|ABcz|c/d/c/B/ AF|c/d/c/B/ AF|
[V:2] z4  |z4   |FGAF|FGAF|ABcz        |ABcz         |
%
[V:1] GCFz         |GCFz          |z4   |z4   |
[V:2] c/d/c/B/ AF|c/d/c/B/ AF|GCFz|GCFz|
```



Brother John

*Traditional*

Fancy staves can be obtained using staffscale and stafflines. Percussion notes are notated using sharps for x-shaped note heads, and flats for circle-x-shaped note heads:

```
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



> ☞  **Warning**
>
> Percussion notes are not supported in `abc2svg`. To be more precise, `abc2svg` imple-
> ments percussion notes using an extension called *note mapping* that will be explained
> in Section 7.2 (p. 109).

### 3.1.2  Positioning Voices: *%%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
laid out on the score, we use the %%score directive.

   %%score must be followed by voice names, optionally enclosed by a pair of delimiters:
[], {}, and (). As other fields in the header, %%score must appear before K:. In the tune
body, if there are voices 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:
  %%score 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: %%score [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. For example: `%%score {MS MD}`

- if two or more voices are enclosed between parentheses, they will be printed on the same staff. For example: `%%score [(SA) (TB)]`

- to add measure bars that cross the staves, use the character | between voice names: `%%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).

In addition to `%%score`, voices and staves can also be specified using the alternate directive `%%staves`. These two directives use the same parameters, but measure bar indications work the opposite way. For instance, `%%staves [S A T B]` is equivalent to `%%score [S|A|T|B]`, and it means that bar lines are drawn. As you can see, using `%%score` is more intuitive.

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 2.1.4 (p. 18).

```
X: 1
T: Studio Op. 10 - N. 3
C: F. Chopin
M: C
%%score {RH1 | (LH1 LH2)}
V: RH1 clef=treble name="Piano"
V: LH1 clef=bass
V: LH2 clef=bass
K: F
% measures 1 - 2
[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                     |
% 3
[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*
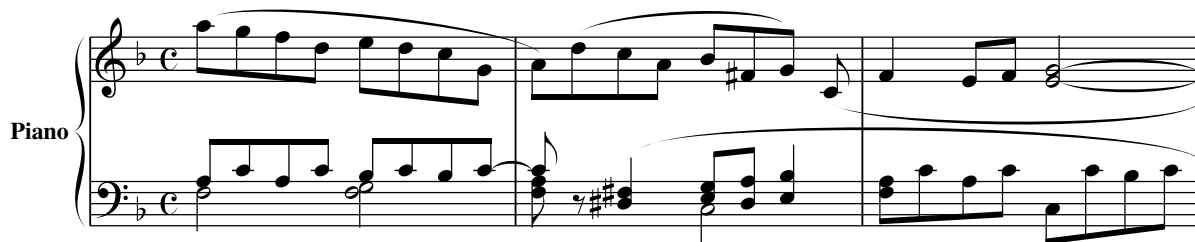


Let's now try a more complex piece. These are the first four measures of Mozart's famous "Ave Verum Corpus", for organ and SATB:

```
X: 1
T: Ave Verum Corpus
C: W. A. Mozart
M: 4/4
L: 1/4
Q: "Adagio"
%%score [(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
%
% measures 1 - 4
%
[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 Corpus

W. A. Mozart



Note that the voices were intentionally written in reverse order, but %%score rearranged them as expected. Normally, for clarity sake voices should be written in the same order as specified in %%score.

Sometimes, a piece has a *master* voice and *cue* voices; the former is the main voice in the

piece, while cue voices are used, for instance, as hint for variations. `%%score` and `%%staves` accept the symbol + before a voice name, which becomes the master voice. The `cue=on` option is then used to set the cue voice(s):

```
X: 1
M: C
L: 1/4
K: C
%%score (1 +2)
V:1 cue=on
cdef | Z2 | g/f/e/d/ c2|]
V:2
C4 | CEGc | cGEC | C4 |]
```

As a last example, let's see 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 case we use `M:none`. The length of each measure is different for each voice, consequently the `!longphrase!` symbol replaces measure bars. We also want "cut time" tempo indicated. This is how the piece is written:

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



Unfortunately, this piece cannot be rendered correctly by the current version of abc2svg.

### 3.1.3   Printing Reductions

Just by changing the %%score directive, we can obtain *reductions* or part extractions from the same source. In fact, %%score tells the scorewriter to print only the voices that are specified. We are free to omit voices in %%score, even if we wrote the corresponding music lines.

The next example is an excerpt from Händel's "Giulio Cesare" that specifies V: fields for two violins, a solo voice, and bass. We also write three pairs of T: and %%score lines, but keep two of them commented out. Only the pair that is uncommented will print an additional subtitle and will specify a score layout.

```
X: 1
T: Giulio Cesare in Egitto
T: Sesto: Svegliatevi nel core, furie d'un alma offesa (excerpt)
C: G.F. Händel
L: 1/8
```

```
M: C
T: Full score
%%score [V1 V2 Sesto Bassi]
%  T: Vocal part and keyboard reduction
%  %%score Sesto {V1 Bassi}
%  T: First violin only
%  %%score V1
V: V1    clef=treble % name="Violino I"
V: V2    clef=treble % name="Violino II"
V: Sesto clef=treble % name=Sesto
V: Bassi clef=bass   % name=Bassi
K: EbMaj
%
V:V1
z2 z G ce/d/ cc  |cC e2-ed/e/ fA    |AG z c !trill!c3/2=B//c// dG |
V:V2
z2 z G ce/d/ cc  |cC e2-ed/e/ fA    |AG z c !trill!c3/2=B//c// dG |
V:Sesto
z8               |z8                |z8                           |
V:Bassi
C,2 z G, CE/D/ CC|CC, z G, A,F,B,B,,|E,E,,, z C, E,C,G,E,         |
```

Note that only these lines:

```
T: Full score
%%score [V1 V2 Sesto Bassi]
```

are uncommented. The resulting full score is:

### Giulio Cesare in Egitto
Sesto: Svegliatevi nel core, furie d'un alma offesa (excerpt)
Full score

*G.F. Händel*



If we change these lines:

```
%  T: Full score
%  %%score [V1 V2 Sesto Bassi]
%  T: Vocal part and keyboard reduction
%  %%score Sesto {V1 Bassi}
```

```
T: First violin only
%%score V1
```

then we tell the scorewriter to ignore all voices but V1. The resulting score, predictably, will contain only one staff:

<div align="center">

## Giulio Cesare in Egitto

Sesto: Svegliatevi nel core, furie d'un alma offesa (excerpt)

First violin only

</div>

<div align="right">

*G.F. Händel*

</div>



Similarly, we could have obtained a solo voice and piano reduction leaving these lines uncommented:

```
%  T: Full score
%  %%score [V1 V2 Sesto Bassi]
T: Vocal part and keyboard reduction
%%score Sesto {V1 Bassi}
%  T: First violin only
%  %%score V1
```

### 3.1.4  Voice Overlay: &

In some pieces of music, isolated notes (courtesy notes) or short runs of notes belonging to an extra voice are occasionally needed.

To avoid introducing an additional voice that would be mostly void of notes, we can use the & symbol. When it's written within a measure, it starts a new voice called *overlay*. The notes that follow, up to the next bar line, belong to the overlay voice. Up to four overlay voices may be added; each starts with an &.

```
X:1
M:4/4
L:1/4
K:C
%
cdef & xxCD|gfed & EDCB,|c2G2 & C2x2|c2 z2 & E2x2 & G2x2|
```
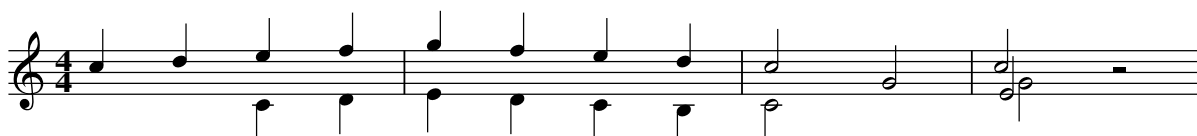


The above piece can be equivalently written as:

```
X:2
M:4/4
```

```
L:1/4
K:C
%%score (1 2 3)
%
[V:1] cdef|gfed |c2G2|c2 z2|
[V:2] xxCD|EDCB,|C2x2|E2 x2|
[V:3] x4  |x4   |x4  |G2 x2|
```

which is more verbose.

### 3.1.5 Writing Clean Sources

Polyphonic scores can grow quite crowded: multiple systems, multiple voices, lyrics, symbol lines, text annotations. It is important to maintain an Abc source clean and easy to read.

One of the main advantages of the Abc notation is its readability. A well written Abc source is easy to understand, debug, maintain, and extend. You will do yourself a favour if try and keep your Abc sources as clean as possible.

You may want to follow these rules of thumb:

- write relatively short music lines (less than 80 characters, 66 would be optimal);

- write the voices in the same order as they would appear on paper;

- insert comments to indicate the measure numbers, and to explain complex passages if needed;

- divide systems with a short comment line;

- bars within the same system should be aligned, if possible.

In general, the layout of the source should resemble that of the printed score as much as possible. The following source is a model you should follow.

```
X: 1
T: Son questi i crespi crini
C: Claudio Monteverdi (1567-1643)
M: C
L: 1/4
Q: "Andante mosso" 1/4 = 200
V: 1 clef=treble name="Soprano"sname="A"
V: 2 clef=treble name="Alto"   sname="T"
V: 3 clef=bass   name="Bass"   sname="B"
U: i = !invisible!
K: Eb
%
% 1 - 4
%
[V: 1] i|: z4   | z4      | f2ec          | _ddcc                |
w: Son que-sti~i cre-spi cri-ni~e
[V: 2] i|: c2BG | AAGc    | (F/G/A/B/)c=A| B2AA                  |
w: Son que-sti~i cre-spi cri-ni~e que ---- sto~il vi-so e
[V: 3] i| :z4   | F,2E,C, | _D,D,C,F,     | (B,,/C,/_D,/E,/)F,F,|
```

```
w: Son que-sti~i cre-spi cri-ni~e que ---- sto~il
%
% 5 - 9
%
[V: 1] cAB2            | cAAA         | c3B      | G2HGz :: e4    |
w: que-sto~il vi-so ond' io ri-man-go~uc-ci-so. Deh,
[V: 2] AAG2           | AFFF        | A3F      | =E2HEz:: c4    |
w: que-sto~il vi-so ond' io ri-man-go~uc-ci-so. Deh,
[V: 3] (A,G,/F,/E,2) | A,,_D,D,D, | A,,3B,, | C,2HC,z ::A,,4|
w: vi --- so ond' io ti-man-go~uc-ci-so. Deh,
%
% 10 - 12
%
[V: 1] f_dec        | B2c2       | zAGF          |
w: dim-me-lo ben mi-o, che que-sto
[V: 2] ABGA         | G2AA       | GF=EF         |
w: dim-me-lo ben mi-o, che que-sto sol de-
[V: 3] _D,B,,C,>D, | E,2A,,F,, | =E,,F,,C,_D, |
w: dim-me-lo ben mi-o, che que-sto sol de-
%
% 13 - 15
%
[V: 1] =EFG2            |1 F2z2   :|2 F8  |]
w: sol de-si-o. _
[V: 2] (GF3/2=E//D//E)|1 F2z2   :|2 F8  |]
w: si ---- o. _
[V: 3] C,4             |1 F,,2z2 :|2 F,,8|]
w: si-o. _
```

♫ ♫ ♫ ♫ ♫

# Chapter 4

# Formatting with `abcm2ps`

## 4.1 Formatting Parameters

WE HAVE learnt how to write music in ABC, from simple single-voice tunes to complex polyphonic scores. In theory, we are done: in fact, the old ABC notation only describes high-level musical elements, and it's completely output-agnostic.

In practice, ABC 2 allows for the implementation of low-level details such as formatting parameters, MIDI instruments, and also extensions like tablatures, chord diagrams, and much more. These features are obtained writing lines that start with %%; these special lines are known as *pseudo-comments* or *directives*. Since they start with %, directives can be treated as comments by programs that don't implement them.

We have already dealt with directives; for example, when we learnt %%score. There are several types of directives: some specify the page layout, others the fonts, the spacing, many types of customisation, and so on. We will see in Chapter 6 (p. 91) that abc2midi provides its own directives for MIDI output.

Directives can be written in the source, or in external ABC files known as *header files*, or in external files known as *format files*. abcm2ps and abc2svg support over 160 directives that are listed in alphabetical order and briefly described at this page: http://moinejf.free.fr/abcm2ps-doc/index.html

> ✎ **Note**
>
> Alas, this is the point where ABC applications start to become incompatible with one another. In particular:
>
> - The ABC/MusicXML translators only support a small subset of the directives supported by abcm2ps, abc2svg, and abcMIDI;
>
> - abc2svg supports nearly all of abcm2ps' directives, but is not 100% compatible with it. Conversely, some directives are only available in abc2svg;
>
> - other ABC applications may ignore directives altogether, or support their own.
>
> As a result, *this chapter is specific to `abcm2ps`* and, mostly, to abc2svg. Useful directives that are supported by other programs will be described in the following chapters.
> You can skip this chapter if you plan to convert your ABC music to MusicXML, and do the final formatting with another application.

Many directives accept a *parameter* of one of these types:

- a *unit of length* in centimeters (cm), inches (in), or PostScript points (pt): for instance, 30pt, 1cm, 0.3in. A point equals $0.352\bar{7}$ mm.

- a *logical value* that can be specified using the words true, yes, 1 if positive; false, no, 0 if negative;

- a *string*, like Times-Roman 24;

- a *number*, either integer or float (that is, with decimals).

Directives may have a different *scope*. That is, they may affect the appearance and/or the behaviour of different elements:

- the whole Abc file: this is called *global* scope;

- the current *page*;

- the current *tune*;

- the current *voice*;

- *immediate* (the directive will immediately apply to the next music elements);

- *generation* (the last directive of this scope will apply to the next music elements);

- *restart* (the directive acts as if a new tune were started.)

Furthermore, some directives have a default value; for instance, page width and height are 8.5 and 11 inches (ANSI letter paper size) by default.

abcm2ps and abc2svg have sensible default values, and usually make beautiful scores. In general, you should refrain from messing about and changing too many directives. Unless you really know what you are doing, chances are you'll make the layout worse than the default. I suggest that you only customise the following:

```
% PAGE LAYOUT
%
%%pagewidth      21cm
%%pageheight     29.7cm
%%topmargin      1.5cm
%%botmargin      1.5cm
%%leftmargin     1.5cm
%%rightmargin    1.5cm
%%topspace       0cm
%%footer " $T, p. $P "
%
% FONTS
%
%%titlefont      Times-Bold 28
%%subtitlefont   Times-Bold 22
%%composerfont   Times-Roman 16
%%vocalfont      Times-Roman 15
```

```
%%annotationfont Times-BoldItalic 14
%%footerfont     Times-Roman 12
%
% MISC
%
%%measurebox     yes       % measure numbers in a box
%%measurenb      0         % measure numbers at start of line
%%pagescale      0.8       % page size
%%maxshrink      1.0       % compact the notes
%%linebreak      <none>    % automatic line breaks
%%squarebreve    yes       % print breve note as square glyph
%%printparts     no        % don't print part names
```

As you can see, many directives have a self-explanatory name such as `%%pagewidth` or `%%leftmargin`, while others need some explanation. The most important directives will be described in the next few sections; a complete list of available directives is presented in Appendix A.4 (p. 145), where they are grouped by purpose.

Once the page format and margins are set, you should be able to adjust the number of pages of your scores just by changing the value of `%%pagescale` and `%%maxshrink`.

> ☞ **Warning**
>
> Some font directives have different default values in `abcm2ps` and `abc2svg`; PostScript and SVG, in fact, use different fonts. As a result, the score will look slightly different. Furthermore, `abc2svg` has no control on page layout; directives for margins, header and footer will not be obeyed exactly. Using `abc2svg`, final page layout will depend on the browser's print options.

When directives are written at the top of a file, they apply to all of the tunes that follow. In this case, they are called *global parameters* or *global definitions*. Directives can also be written within a tune and between tunes.

Most directives can also be used as `abcm2ps` options in the command line. For instance, the following command line typesets some music specifying the `%%pagewidth`, `%%pageheight`, `%%pagescale`, and `%%titlefont` directives:

```
abcm2ps -c -O= --pagewidth "21cm" --pageheight "29.7cm" \
  --pagescale 0.7 --titlefont "* 18" tunebook.abc
```

Spaces are not allowed in directive parameters that require a unit of length, when they are specified in the command line. Although quotes are optional, they should always be used to avoid clashes with the shell. In the command line above, `--titlefont "* 18"` means: "leave the title font unchanged, but change the font size to 18 pt". The asterisk must be enclosed in quotes, otherwise trouble may occur.

As a final note: if the Abc source contains directives that are also specified as command line options, the latter override the former. That is: if our source contains the global parameter `%%pagescale 0.85`, but we specify, say, `pagescale 0.7` as command line option, the latter will apply.

### 4.1.1   Using Format and Header Files

If we have many Abc files that share a common format, it is preferable not to write the directives in the sources. *Format files* and *header files* are a better option, because they keep the format separated from the music. Should we want to change the parameters of our scores, we would need to adjust them in just one file.

Format files have a `.fmt` extension, and contain directives without the leading double percent characters `%%`:

```
% This is example.fmt

pagescale                 0.85
topmargin                 2 cm
titlefont Times-Bold      24
subtitlefont Times-Bold   20
...
% End of the format file.
```

This is the corresponding Abc file, which we will call a *header file*. To make it clear that is's a header file and not an ordinary Abc file, it has a `.abh` extension:

```
% This is example.abh

%%pagescale                 0.85
%%topmargin                 2 cm
%%titlefont Times-Bold      24
%%subtitlefont Times-Bold   20
...
% End of the header file.
```

Let's suppose we saved `example.fmt` and `example.abh` in the same folder as our source files. To format the file `mytune.abc` using `example.fmt`, we insert the line:

```
%%format example.fmt
```

at the top of the source, or use `abcm2ps`'s `-F` option in the command line:

```
abcm2ps -O= -c -F example.fmt mytune.abc
```

The `.fmt` extension can be omitted, both in the command line and in the source. Two or more format files can be used at the same time.

To use the header file instead of a format file, insert the line:

```
%%abc-include example.abh
```

at the top of the source. If you use `abc2svg`, you will be prompted to load `example.abh` alongside the current source.

> ✍ **Note**
>
> abcm2ps can use both format and header files, while abc2svg only uses header files (only one per source). Using %%abc-include makes Abc files compatible with both abcm2ps and abc2svg, and is therefore highly recommended.

If we keep your format files in a folder, i.e. c:\music\format, we 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 and macOS usually store the format files in /usr/share/abcm2ps. As we already know, the command:

abcm2ps -V

reports the default format file directory. The current version of abcm2ps ships with three format files: flute.fmt, landscape.fmt, and musicfont.fmt; more could be included in future releases, along with corresponding header files.

Using format or header files is the best solution when we want to typeset a series of pieces that share the same style. Moreover, abcm2ps and abc2svg can be extended by defining additional symbols, as we will see in Chapter 7 (p. 107). Format files containing libraries of symbols can thus be employed when needed.

## 4.2 Changing Parameters

Parameter values can be redefined when necessary. In accordance with the directive's scope, changing its parameters may or may not take effect immediately. For example, page layout directives will only change after a page break; directives with an "immediate" or "generation" scope will take effect immediately after che change. Other directives cannot be changed mid-tune, unless we insert a directive that has a "restart" scope. The examples in the following sections will clarify how changing parameters works.

### 4.2.1 Directives as I: Fields

There are two ways to change a parameter: either write a directive on a line on its own, or an *instruction field* I: that is followed by the directive (spaces allowed). Instruction fields can also be written inline.

For example, the directive:

```
%%vocalfont Times-Roman 12
```

can be written as instruction field:

```
I:vocalfont Times-Roman 12
```

Now you understand what we did when we introduced I:clef.

A parameter change can be followed by the lock keyword. This means that any following change of the same parameter will not take effect, unless the lock is specified again.

> ✍ **Note**
>
> Directives should be preferable over `I:` fields , because they are ignored by applications that don't implement them. On the contrary, the equivalent `I:directive` field will probably cause a warning or error message if it's not recognised.

Let's use both methods to change the `%%vocalfont` parameter in a tune. This parameter has a "generation" scope, so the change is immediate:

```
%%titlefont Times-Italics 24
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,
+: mi- te~A- gnel- lo re- den- tor!
I: vocalfont Times-Italic 12
w: Voi- ci No- \"el, \^o dou- ce nuit! \
w: L'\'e- toile~est l\`a qui nous con- duit.
%%vocalfont Times-Roman 12
w: Si - lent night! Ho - ly night! All is calm,_ all is bright.
```

## Silent Night

F. Gruber



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 \
I: vocalfont Helvetica-BoldOblique 13
w: la la la la, la la la la \
%%vocalfont Helvetica 12
w: la la la la.
```
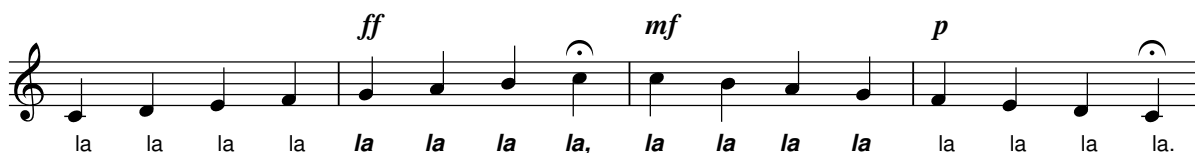
A better way to change fonts will be explained in Section 4.2.14 (p. 73).

### 4.2.2   Line, Staff, Tune, and Page Breaks

A *line break* ends the current staff, stretches the music between the page margins, and starts a new staff below the current one. This is the default behaviour in Abc.

If we don't use the `-c` option of `abcm2ps`, we can force line breaks anywhere in the source with a single ! character. In addition, we can define one or more line-breaking characters and keywords using the `%%linebreak` directive:

- valid characters are `$!` `*` `;` `?` `@` separated by a space if we use more than one;

- the <none> keyword causes line breaks to occur automatically, and makes all line-breaking characters invalid;

- the <EOL> keyword causes line breaks to occur at end of lines, and makes all line-breaking characters invalid except !,

- `%%linebreak` with no parameters corresponds to `%%linebreak` <none>.

The breaking character should always be placed after a barline.

In the next example, only the first two music lines obey line breaks set with `*` and !:

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



A *staff break* is like a line break, but it will not stretch the music between the page margins. This is done with the `%%stretchstaff 0` directive:

```
%%stretchstaff 0
X: 1
L: 1/4
K: C
CDEF|
CDEF|GABc|
CDEF|GABc|cdef|]
```



Forcing line and staff breaks should be done with special care, in order to avoid ugly results.

If a tune doesn't fit in the current page, it is printed on a new page; this is the default behaviour in abcm2ps, while abc2svg always splits the tune. To obtain the same behaviour in abcm2ps, we use the %%splittune directive followed by a logical value, or by the strings odd or even to split the tune on odd or even pages.

Finally, to force a new page we use the aptly named %%newpage directive.

### 4.2.3   Avoiding Line Breaks: \

By default, ⟨n⟩ measures in a source line produce ⟨n⟩ measures in the score. Sometimes it is not convenient to write, say, eight measures on the same line, because the source would become less readable. In such cases, we can write the \ continuation character at the end of a line to indicate that the staff continues on the next line. In other words, \ means "no line break here".

Most commonly, we will use \ in tunes with repeats; this is one of the cases when we don't want abcm2ps or abc2svg to format the music automatically. Typeset the following tune with and without \ to see the difference:

```
X:92
T:Breeches Full of Stitches
R:polka
M:2/4
L:1/8
K:Amaj
%
"A" A>B cA|"D" BA ce|"A" A>B cA|"D" BA F2 | \
"A" A>B cA|"D" BA ce|    A>B AF|"E" FE E2 :|
"A" e>f ec|"D" BA Bc|"A" e>f ec|"D" BA F2 | \
"A" e>f ec|"D" BA Bc|    A>B AF|"E" FE E2 :|
```

## Breeches Full of Stitches
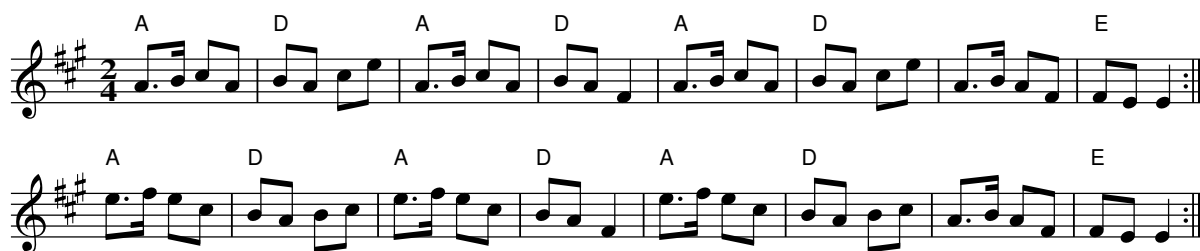


### 4.2.4 Controlling Measures

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

- in most cases, it is best to let abcm2ps or abc2svg do the work. Please refer to Section 2.1.1, p. 15;

- the opposite approach is inserting the exact number of measures in each line, then running abcm2ps without the -c option or adding I:linebreak <EOL> in the source;

- when we want each staff to contain ⟨*n*⟩ measures, we use the %%barsperstaff ⟨*n*⟩ directive in the source or the option -B ⟨*n*⟩ in the abcm2ps command line;

- %%alignbars ⟨*int*⟩ aligns the bars of the next ⟨*int*⟩ lines of music. It only works on single-voice tunes, and only with abcm2ps.

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

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 or abc2svg will rework the lines at their discretion.

### 4.2.5 Voice Size

In addition to the scale parameter in V: definitions (Section 3.1.1, p. 45), the note scale (size) for a particular voice can be set using the %%voicescale directive. This is useful to specify different sizes for the main voice and an overlay.

In this example, we want to typeset the main voice a bit larger than usual, and the overlay voice a bit smaller. The default value is 1, minimum size is 0.5, maximum is 1.5:

```
X:1
M:4/4
L:1/4
K:C
%%voicescale 1.2 % default=1
cdef & [I:voicescale 0.7] xxCD|gfed & EDCB,|
c2G2 & C2x2|c2 z2 & E2x2 & G2x2|
```
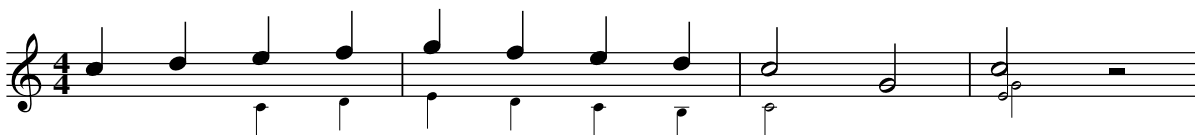
## 4.2.6 Repeated Sequences (*Simile*)

Sometimes, a tune contains repeated runs of notes or measures. Instead of printing the same sequence multiple times, we can print the *simile* symbol that means "repeat the previous notes/measures". This can be done with the inline field `I:repeat`, that replaces the previous sequence with the appropriate symbol.

This field can also be followed by two numbers ⟨*n*⟩ and ⟨*k*⟩: the first indicates the number of notes/measures to be replaced by the repeat symbol, the second the number of repetitions we want. Let's see an example:

```
X: 1
M: 4/4
L: 1/4
K: C
%
% let's repeat a single note three times
C [I:repeat 1 3] C C C|
% let's repeat a sequence of four notes twice
C//E//G//E// [I:repeat 4 2] C//E//G//E// C//E//G//E// C |
% let's repeat a whole measure once
CEGc | [I:repeat] CEGc |]
```



## 4.2.7 The Grand Staff

In piano music, the 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, respectively. The parameter can also be `+1` or `-1`, meaning the next or the previous staff.

```
X:1
M:C
L:1/4
U: m = !xstem!
K:none
%%score {1 2}
V:1
cc//c//c//c//c2 | [CEGc]2[CEGc]2|
V:2 bass
mC,C,//mC,//C,//mC,//mC,2|m[C,,C,]2[C,,C,]2|
```

```
V:1
c/G/E/C/ [I:staff +1] C/G,/E,/C,/|C,/E,/G,/C/ [I:staff -1]C/E/G/c/|
V:2
z4 | z4 |
```
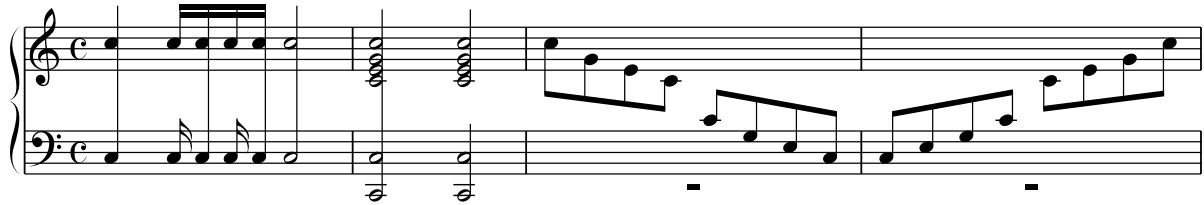


> ☞ **Warning**
>
> Unfortunately, cross-staff slurs are not yet implemented in abcm2ps and abc2svg.

### 4.2.8 Change of System

In some pieces (say, for soloist and choir), only some instruments or voices play at any given moment, while other instruments or voices are silent. Occasionally, we may want to print the solo part only, and omit the silent voices.

We could be tempted to change %%score as needed and write only the active voices, but for compatibility reasons %%score should only appear once and remain unchanged. We shall write voices that do not play using rests or multi-measure rests, then use the %%staffnonote directive.

%%staffnonote⟨*n*⟩ prints or hides voices according to its parameter:

- if ⟨*n*⟩ = 0, only staves that contain visible notes are printed;

- if ⟨*n*⟩ = 1, only staves that contain visible noted or visible rests are printed;

- if ⟨*n*⟩ = 2, all staves are printed.

Here is as example. The following rendition of "Orientis partibus" (a medieval song) for two voices has different stanzas and a refrain. Stanzas are sung by the first voice only; the refrain is sung by both voices. The %%staffnonote 0 directive instructs the scorewriter to print only the active voice, and hide voices thta only contain rests.

```
X: 1
T: Orientis Partibus
C: Anonymous
M: 2/4
L: 1/8
%%score [1 2]
V:1 name=V1 sname=V1
V:2 name=V2 sname=V2
K: FMix
%%linebreak <EOL>
%%staffnonote 0
```

```
%
% 1 - 4: VOICE 1 ONLY
%
[V:1] FG AF|GE F2|cc dB|cc A2|
w: O-ri-en-tis par-ti-bus ad-ven-ta-vit a-si-nus,
[V:2] Z4|
% 5 - 8
[V:1] AG BA|GF A2|cB AF|GE F2|
w: pul-cher et for-tis-si-mus, Sar-ci-nis ap-tis-si-mus.
[V:2] Z4|
%
% 9 - 14: BOTH VOICES
%
[V:1] F2 z2|FG AG|F2 z2|F2 z2|FG AG|F2 z2|
w: Hez, Hez sir as-nes, hez! Hez, hez sir as-nes, hez!
[V:2] A2 z2|AB cB|A2 z2|A2 z2|AB cB|A2 z2|
%
% 15 - 18: VOICE 1 ONLY
%
[V:1] FG AF|GE F2|cc dB|cc A2|
w: Hic in col-li-bus Si-chan iam nu-tri-tus sub Ru-ben
[V:2] Z4|
% 19 - 22
[V:1] AG BA|GF A2|cB AF|GE F2|
w: tran-si-it per Ior-da-nem sa-li-it in Be-thle-hem.
[V:2] Z4|
%
% 23 - 28: BOTH VOICES
%
[V:1] F2 z2|FG AG|F2 z2|F2 z2|FG AG|F2 z2|
w: Hez, Hez sir as-nes, hez! Hez, hez sir as-nes, hez!
[V:2] A2 z2|AB cB|A2 z2|A2 z2|AB cB|A2 z2|
```

### 4.2.9  Positioning Music Elements

The position of many music elements can be specified using the %%pos directive. It allows to place decorations, accompaniment chords and lyrics above or below the staff, and force note stem direction.

The syntax is:

%%pos ⟨*type*⟩ ⟨*position*⟩

The two parameters ⟨*type*⟩ and ⟨*position*⟩ are text strings. In particular, ⟨*type*⟩ can be one of the following:

- dynamic, to specify the position of decorations concerning dynamics (e.g. !<(!, !>(!, etc.);

- gchord, to specify the position of accompaniment chords;

- gstem, to specify the direction of grace notes stems;

- ornament, to specify the position of decorations concerning ornaments (e.g. !trill!, !fermata!, etc.);

Figure 4.1: A piece with a variable number of systems.

- stem, to specify the direction of normal notes stems;

- vocal, to specify the position of lyrics (once set, it cannot be changed);

- volume, to specify the position of decorations concerning loudness (e.g. !pp!, !pp!, etc.).

Parameter ⟨*position*⟩ can be one of the following:

- auto for automatic position;

- above or up, above the staff;

- below or down, below the staff;

- hidden, to hide the music element;

- opposite, for opposite direction (applies to stem and gstem).

In the following example, we change the positioning of several music elements:

```
X: 1
M: C
L: 1/4
K: C
%%pos dynamic up
%%pos volume up
%%pos stem up
%%pos gstem auto
%%pos gchord down
%%pos ornament up
!<(!"C"CDE!<)!!f!F|!>(!"G"GFE!>)!!p!D|"C"CDEF|!fermata!{GA}G2 z2|
%%pos dynamic down
%%pos volume down
%%pos stem down
%%pos gstem down
%%pos gchord up
%%pos ornament hidden
!<(!"C"CDE!<)!!f!F|!>(!"G"GFE!>)!!p!D|"C"CDED|!fermata!{CD}C2 z2|]
```

### 4.2.10 Customising Slurs, Beams, Stems, and Flags

The appearance of slurs, note beams, stems, and flags can be further customised. The `%%flat-beams` directive enables the drawing of flat (horizontal) note beams; `%%slurheight` modifies the height of note slurs; `%%straightflags` draws straight note flags, as commonly used in bagpipe music; finally, `%%stemheight` lets one specify the note stem height. Have a look:

```
X: 1
M: 4/4
L: 1/8
K: C
%%slurheight 2
%%flatbeams 1
%%straightflags 1
%%stemheight 30 % default: 20
(GABc) (cBAG)|G A B c G A B c|

X: 2
M: 4/4
L: 1/8
K: C
%%slurheight 0.5
%%flatbeams 0
%%straightflags 0
%%stemheight 20
(GABc) (cBAG)|G A B c G A B c|
```
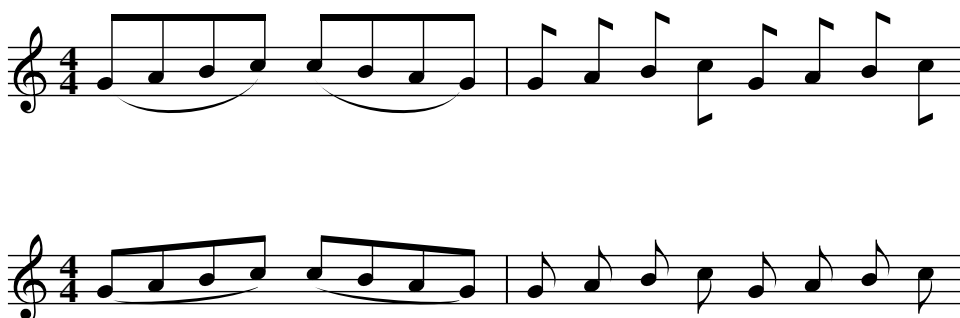


These parameters are applied once, and cannot change midtune. If we want to change their value, we will have to start a new tune, or use `%%vskip`.

### 4.2.11 Customising Tuplets

The `%%tuplets` directive allows for fine-grained tuplet indications. The syntax is:

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

Parameters are:

- `where` is a number that indicates where to draw the tuplet indication. `0` = automatic, `1` = never, `2` = always.

- `what` is a number that indicates what to draw. `0` = draw a bracket, `1` = draw a slur.

- `value` indicates the number to print. `0` = the value of ⟨*n*⟩ in the tuplet, `1` = no value at all, `2` = a ratio ⟨*n*⟩:⟨*t*⟩.

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

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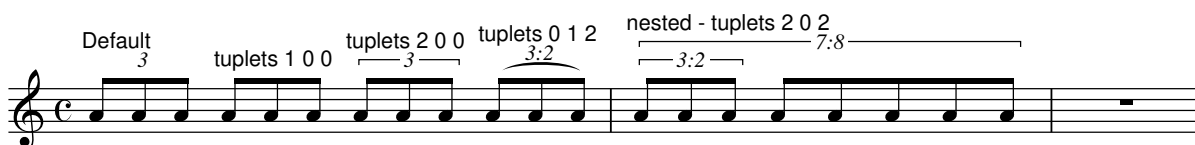


### 4.2.12   Customising Volta Brackets

By default, a volta bracket (the line above the staff) extends over four measures, or just two measures if the repeated section includes more than four measures. These two numbers can be customised using the %%rbmax and %%rbmin directives.

Let's suppose we want the volta bracket to extend at most for three measures, or just one if there are more than three measures in the repeated section:

```
X: 1
M: 2/4
L: 1/4
K: C
%%rbmax 3
%%rbmin 1
|:CD |1 EF|GA|Bc|cG :|2 EF|GG|C2||
```



### 4.2.13   Combining Voices

When notes or rests of different voices overlap because they are equal or nearly so, one wants to fine-tune the way they are printed. The %%voicecombine directive is followed by a parameter that produces different effects:

- -1: notes and rests are displayed as they are written;

- 0: rests of equal duration are merged as one rest;

- 1: notes of equal duration are merged as a chord, with the exception of chord inversions, unisons, second intervals;

- 2: notes of equal duration are always merged as a chord;

- 3: unisons are always merged as a single note.

```
X:1
M:none
L:1/4
K:C
%%score (1 2)
[V:1] [I:voicecombine -1] "^%%voicecombine -1"c2cz | c2 c2 | c2 z2 |\
[V:2] CEGz | z2 _B2 | c2 z2 |\
[V:1] [I:voicecombine 0]  "^%%voicecombine 0"c2cz | c2 c2 | c2 z2 |
[V:2] CEGz | z2 _B2 | c2 z2 |
[V:1] [I:voicecombine 1]  "^%%voicecombine 1"c2cz | c2 c2 | c2 z2 |\
[V:2] CEGz | z2 _B2 | c2 z2 |\
[V:1] [I:voicecombine 2]  "^%%voicecombine 2"c2cz | c2 c2 | c2 z2 |
[V:2] CEGz | z2 _B2 | c2 z2 |
[V:1] [I:voicecombine 3]  "^%%voicecombine 3"c2cz | c2 c2 | c2 z2 |
[V:2] CEGz | z2 _B2 | c2 z2 |
```



### 4.2.14   Using Fonts and Text

> ☞ **Warning**
>
> Again: let me remind that abcm2ps and abc2svg use different fonts. Scores typeset with abc2svg will look slightly different.

abcm2ps supports nearly all PostScript fonts, which are listed in Appendix A.5 (p. 158). 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 we can use anytime with any font directive. To use other fonts, we have to *declare* them inserting the %%font directive at the top of the source. Text-related directives are:

- %%font declares a font;

- %%textfont sets a font;

- %%begintext…%%endtext define a range of text lines;

- %%center centers the following text;

- %%vskip inserts the specified vertical space;

- %%sep inserts a short separator.

Here is an example that shows abcm2ps's capability of alternating text in different fonts with pieces of music.

```
% 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 Times-Roman 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
M: 4/4
L: 1/4
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 Times-Roman to AvantGarde-Book. Here is the same
Etude with a few small variations to make it more interesting:
%%endtext
X: 2
T: Etude
T: second version
M: 4/4
L: 1/4
Q: "Adagio"
K: C
%
.C{DCB,}C.G{AGF}G|A>AG2|.F{GFE}F.E{FED}E|D>DC2|
%%sep 0.2cm 0.2cm 7cm
% the following line increases the character size
%%textfont * 20
```

```
%%center End of the example.
%%sep 0.2cm 0.2cm 7cm
```

# 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 Times-Roman to AvantGarde-Book. Here is the same Etude with a few small variations to make it more interesting:

*Etude*
second version



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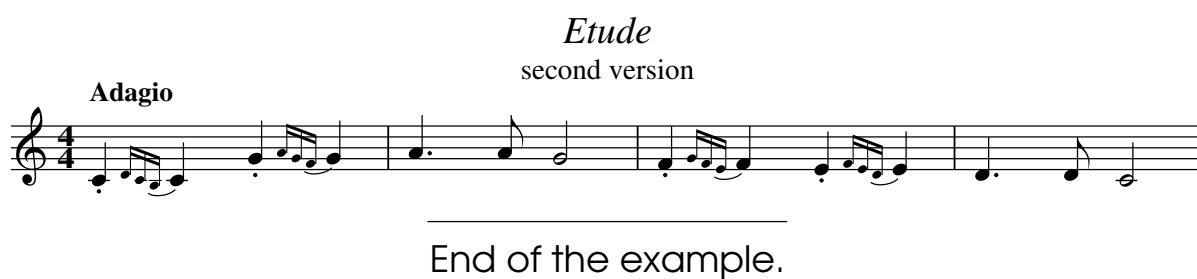
To use different fonts in string, we specify an alternate set of fonts with the %%setfont--1, %%setfont-2, %%setfont-3, and %%setfont-4 directives, followed by a font name and a font size in points. These can be referred to in strings as $1 $2 $3 $4, and have the effect of changing the text font:

```
%%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 is useful to emphasise $0some parts.
```

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



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

### 4.2.15   Voice Colour

Especially for teaching purposes, it may useful to highlight a voice or a run of notes. This can be done typesetting the voice at a different scale using the scale= in the corresponding V: field. A nice option is typesetting the voice (or the notes) using a different colour.

The %%voicecolor #rrggbb directive changes the colour of the following notes. rr gg bb are exadecimal values (i.e. in the range 00–ff, capital letters allowed) of the red, green, and blue colour components.

In the following example, we change the colour of some runs of notes:

```
X:1
M:C
L:1/4
K:C
%%score {1|2}
%
[V:1] [I:voicecolor #0000c0]cccd | e2d2 | cedd | c4 |
[V:2] C,G,E,G, | [I:voicecolor #c00000] C,G,F,G, \
[I:voicecolor #000000]| E,G,F,G, |[I:voicecolor #008000] C,G,C,2 |
```
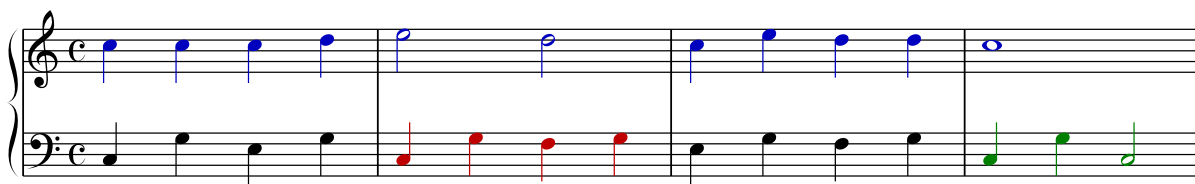


**Blank Sheet Music**

Setting the note colour as white, we can print sheet music (i.e. only empty staves) using this code:

```
X: 1
M: none
L: 1/4
K:C clef=none
%
%%voicecolor  #ffffff
%%linewarn    0
%%stretstaff  1
```

```
%%stretchlast 1
%%linebreak   !
%
C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] !
C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|] ! C4 [|]
```

The `%%linewarn 0` directive turns off the "Line underfull" warning messages.

### 4.2.16   Multi-column Output

Text and music can be laid out in multiple columns that are defined using the `%%multicol start`, `%%multicol new` and `%%multicol end` directives:

`%%multicol start` saves the current page margins and sets the vertical position for the beginning of a column. At this point we 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 we wish.

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

It sounds difficult, but it's actually quite easy as shown in the following 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.

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

Right column (margins: 13, 2)
Width: 21 - 13 - 2 = 6 cm



### 4.2.17  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 respective 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                                                                                     *march*
Second Title                                                                               *Anonymous*
The Last Title

                                                            Written many many years ago...



Note the strange third title. As you can see, it was rearranged to read "The Last Title". Whenever abcm2ps or abc2svg finds that the last word of a title starts with a capital letter and it is preceded by a comma and a space, the word is moved to the head of the title. So it is

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`.

### 4.2.18   Headers and Footers

The following directives define text that should be printed automatically on all pages: `%%header` for the page header, and `%%footer` for the page footer. These directives, 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 we 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;

- `$I`$\langle x \rangle$ prints the contents of a field ($\langle x \rangle$ 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 1.2.3, p. 7.)

Here is an example of the `%%footer` directive used to print the even page numbers on the left, the tune title in the centre, and the odd page numbers on the right. Note that the three areas are *not* separated by spaces, but by tabs!

```
%%footer "$P0    $T        $P1"
```

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

### 4.2.19   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. We use the `%%EPS` directive followed by the name of the file to insert (`abcm2ps` only):

```
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), it will need to be converted to EPS using an appropriate program. Please consult Section 9.7 (p. 136).

### 4.2.20   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 print a number every $\langle n \rangle$ measures, we use %%measurenb $\langle n \rangle$. Measures are numbered starting from 1, unless the first measure is incomplete (pick-up measure, or *anacrusis*); in this case, the anacrusis will count as measure 0.

To set the initial bar number as $\langle n \rangle$, use %%setbarnb $\langle n \rangle$.

In addition to header and footers (Section 4.2.18, p. 79), page numbers can be specified 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.

### 4.2.21   Staff Gaps

In some pieces, we may want to specify the original key signature and voice extension (*ambitus*) at the beginning; or we may want to write a coda. The staff (or the staves) can be interrupted with the %%staffbreak directive. This directive doesn't actually break the staff; rather, it makes a gap:

```
X: 1
L: 1/4
K: C alto4
%%setbarnb 0
%
!stemless![C,G]\
%%staffbreak 0.3cm "f"
K: C treble
CCEE|GGcc|"^al coda"ccee!coda!|fgc2|\
%%staffbreak 1.5cm
!coda!g2C2|]
```



Since we write a "fake" first measure, we also specify the %%setbarnb 0 directive to indicate that the first bar should be be numbered 0, not 1.

The letter f that follows %%staffbreak means that the staff break is forced even if it occurs at the beginning or end of a line. If the staff is part of a system, then the staff break must be applied to all staves in the system.

### 4.2.22   Ambitus (abc2svg)

Using abc2svg, writing an ambitus is much simpler. All we have to do is insert the %%ambitus 1 directive, which will automatically establish the ambitus and print it before the clef:

```
X: 1
L: 1/4
M: C
K: C
%
%%ambitus 1
CEGc|ecGE|C4|
```



### 4.2.23   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 directive %%pagescale ⟨*factor*⟩. By default, the score is produced with a scaling factor of 1. 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 directives for setting the vertical spacing of title, subtitle, lyrics, etc.

- if the -c option is used, the %%maxshrink ⟨*factor*⟩ can be used to reduce the horizontal spacing between notes. Compression is maximum with ⟨*factor*⟩ = 1, minimum with ⟨*factor*⟩ = 0.

- to flatten slurs, use the %%slurheight directive 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.

## 4.2.24   Transposition

Clarinet or sax players, and in general people who play transposing intruments, will find this feature especially useful.

Transposition is made possible by the %%transpose directive, which can be used in two ways:

- %%transpose ⟨*n*⟩, where ⟨*n*⟩ is the number of half-tones the music should be transposed;

- %%transpose ⟨*from-note*⟩ ⟨*to-note*⟩; the parameters are letters that represent the starting note and the transposed note.

The following tune is written in C major, then it is transposed up two half-tones, that is to D major. A second transposition is specified as C_E, which means "from C to E flat"; the fundamental note changes from C to E♭, so the tune in transposed to E♭ major. Finally, cC transposes the tune one octave down, since c in transposed down to C.

```
X: 1
L: 1/4
K: C
%
CDEF|GABc| % up two half-tones
%%transpose 2
CDEF|GABc| % from C to E flat
%%transpose C_E
CDEF|GABc| % down one octave
%%transpose cC
```

```
CDEF|GABc|
```



The `%%transpose` directive *only transposes the printed output*, and this is what we need when a tune written in concert pitch should be transposed for, say, B♭ clarinet. For example, a tune written in the key of C that includes a `%%transpose 2` directive will be printed as if it were written in the key of D, but the resulting MIDI will sound in C.

Another way to transpose our sources will be presented in Section 9.10 (p. 139).

### 4.2.25   Guitar Extensions (`abc2svg`)

`abc2svg` provides some directives that guitar players will find especially useful:

- `%%grid` ⟨*n*⟩ prints a chord grid above (⟨*n*⟩ = 1) or below (⟨*n*⟩ = -1) the tune;

- `%%grid2` *[n]* replaces the notes of a voice with a chord grid. *[n]* may be any digit; when it's omitted, the chord grid is not printed;

- `%%diagram 1` prints guitar chord diagrams above the notes, in addition to chord names.

Both `abcm2ps` and `abc2svg` provide the `%%capo` ⟨*n*⟩ directive, which specifies that a capo should be used on fret ⟨*n*⟩.

The following tune is written in E♭, which is difficult to play on the guitar; we then specify a capo on the third fret, and we also print the chord grid and chord diagrams:

```
X:1
L:1/4
M:4/4
K:Ebmaj
%
%%grid 1
%%diagram 1
%%capo 3
"Eb"EGBe|"Cm"CEGc|"Ab"AcAE|"Bb"BcBd|"Eb"e4|
```
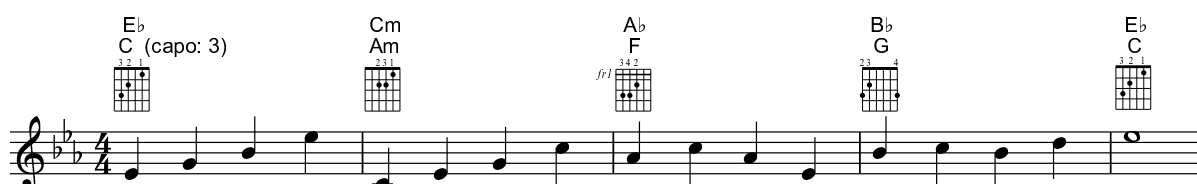
# Chapter 5

# Tune Collections

W<small>E LEARNT</small> in Section 1.2.4 (p. 8) that A<small>BC</small> files may contain several tunes. This feature, along with the ease of use of A<small>BC</small>, spurred the creation of many A<small>BC</small> *tune collections* on the Internet.

The ability to create tune collections is one of the major features of A<small>BC</small>, and is not replicated by any other music notation format or application.

## 5.1   Index Number: *X:*

In the examples examined so far, we dealt with A<small>BC</small> files containing a single tune, so we ignored the X: field. To be precise, since the X: field is obligatory we did use it, but we simply wrote X:1 all the time.

In tune collections, the X: field is used as an index number to "tag" each tune and make it unique. This feature can then be used by applications to create tune databases, or for extraction purposes. For example, we will see (Section 6.1.2, p. 92) that abc2midi creates MIDI files writing the X: field number in the file name.

In a tune collection, the X: field of each tune should be a different number. Tunes need not be sorted by their X: field. The following source is an example of tune collection:

```
%%header "   $T      "
%%footer "   $P      "
%%writefields X

X: 2
T: Apples In Winter
M: 6/8
L: 1/8
R: jig
K: Edor
|:G/A/|BEE dEE|BAG FGE|DA,D FDF|ABc ded|
BEE BAB|def ~g2 e|fdB AGF|GEE E2::
d|efe edB|e/f/ge fdB|dec dAF|DFA def|
|1 efe edB|def ~g2a|bgb afa|gee e2:| \
|2 edB def|gba ~g2e|fdB AGF|GEE E2||

X: 92
T: Breeches Full of Stitches
```

```
R: polka
M: 2/4
L: 1/8
K: Amaj
A>B cA|BA ce|A>B cA|BA F2 | \
A>B cA|BA ce|A>B AF|FE E2 :|
e>f ec|BA Bc|e>f ec|BA F2 | \
e>f ec|BA Bc|A>B AF|FE E2 :|

X: 88
T: Wind That Shakes The Barley, The
M: C|
L: 1/8
R: reel
K: Dmaj
A2AB AFED|B2BA BcdB|A2AB AFED|gfed BcdB|
A2AB AFED|B2BA BcdB|A2AB AFED|gfed Bcde|
f2fd g2ge|f2fd Bcde|f2fd g2fg|afed Bcde|
f2fd g2ge|f2fd Bcde|f2ae g2be|afed BcdB|

% ...many more tunes may follow...
```

As you can see, each tune starts with its `X:` field and ends with one or more blank lines. The `%%writefield` command, followed by X, prints the tune index number before the title.

> 🛑 **Avoid mistakes!**
>
> Do not write the `X:` number in the tune title; it's a very bad practice. Sooner or later, the index number could change, and one would have to edit the tune(s) to fix the title(s)!

## 5.2   Information Fields

A tune collection is especially useful if it contains information associated to each tune, such as the country of origin, related discography, instrument, and so on. Using this information, we can browse a collection and select only the tunes that meet specific criteria.

The following Abc fields are specifically designed to be used in tune collections:

`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: instruction. Example: `I:score (1 2)`

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`

If you contribute Abc files to public sites such as [http://www.thesession.org/](http://www.thesession.org/), it might be a good idea to include at least the `O:`, `R:`, and `D:` fields.

## 5.3   Including Abc Files

While it is possible to create a tune collection as a single big file, it could be more convenient to split it into several files. We may want to split a collection by tune type, or by title, or other criteria.

To print the whole collection, we do not have to join the files manually. The `%%abc-include` directive includes external Abc or format files in the current output. A tunebook could be as simple as:

```
%%format tunebook

%%abc-include a-f.abc
%%abc-include g-l.abc
%%abc-include m-s.abc
%%abc-include t-z.abc
```

## 5.4   Songbooks

The body of a tune may consist of just a set of `W:` lines. Strange though it may seem, melody lines can be omitted.

This means that we can write songbooks in Abc, as in the following example:

```
%%wordsfont Courier 12

X: 131
T: Hey Jude
C: The Beatles
K: Fmaj
W:\     F                   C
W:\ Hey Jude don't make it bad
W:\       C7                    F
W:\ take a sad song and make it better
W:\ Bb                           F
```

```
W:\ Remember to let her into your heart
W:\                  C                F
W:\ and then you can start to make it better
W:\ ...

X: 148
T: Rehab
C: Amy Winehouse
K: Cmaj
W:\       C
W:\ They try to make me go to rehab I say no no no
W:\ C
W:\ Yes, I've been black but when I come back you'll know, know, know
W:\ G                              F
W:\ I ain't got the time and if my daddy thinks I'm fine
W:\ C                           F           C
W:\ He's tried to make me go to rehab I won't go go go
W:\ ...

X: 203
T: Imagine
C: John Lennon
K: C
W:\ (intro)
W:\ C     Cmaj7   F
W:\ C     Cmaj7   F
W:\ C               Cmaj7 F
W:\ Imagine there's no    heaven
W:\ C            Cmaj7 F
W:\ It's easy if you    try
W:\ C        Cmaj7 F
W:\ No hell below us
W:\ C         Cmaj7 F
W:\ Above us only  sky
W:\ ...
```

We first set the words font (`W:` lines) as Courier, a monospaced font that makes it easy to align chords and words lines. Each `W:` line begins with `W:\`, which is necessary to force abcm2ps to print all spaces as they are. Blocks of lines that begin with `W:\` are not split between pages; you may want to divide stanzas with plain `W:` lines.

If you don't like an all-monospaced songbook, you may use Courier for the chords only. The alignment between lyrics and chords will require some manual adjustment, and it will be less obvious in the source:

```
%%setfont-1 Courier-Bold 12
%%setfont-2 Times-Roman  12

X: 131
T: Hey Jude
C: The Beatles
K: Fmaj
W: $1   F           C
W: $2Hey Jude don't make it bad
```

```
W: $1        C7          F
W: $2take a sad song and make it better
W: $1Bb                  F
W: $2Remember to let her into your heart
W: $1           C         F
W: $2and then you can start to make it better
W: ...
```

In this case, forcing spaces with `W:\` is not necessary.

You may wonder why one should write songbooks using Abc instead of, say, a regular word processor or even a simple text editor. The following section explains why.

## 5.5   Selecting Tunes

An incredibly useful feature of `abcm2ps` is the ability to output only the tune(s) in a tune collection that meet some user-defined criteria. For example, we may want to output only the tunes in a certain index range, or tunes of a certain kind (e.g. only jigs), or whose title matches a pattern ("Humours of". . .), or what have you. All this is done using the `%%select` directive, followed by one of two different types of parameter.

The first type is an *index range*, that is a series of numbers that indicate `X:` fields. For instance, if we write the following line at the top a tune collection, only the tunes numbered 10, 12, in the range 20–30 will be printed:

```
%%select 10,12,20-30

...tunes follow below...
```

The same result can be obtained using the `-e` command line option of `abcm2ps`:

```
abcm2ps -O selected_tunes.ps -c -e 10,12,20-30 tunebook.abc
abcm2ps-8.14.2 (2018-12-18)
File tunebook.abc
Output written on selected_tunes.ps (2 pages, 3 titles, 38501 bytes)
```

In an index range we can omit the last number to specify "to the end". That is, `20--` means "from tune #20 to the last tune".

If we want to select tunes by other fields, we can use a *regular expression*. This is quite a complex subject, but it can be simplified: a regular expression is basically a string, containing special characters, that matches several other strings.

Le't see a first example. This directive selects tunes that specify "jig" in their `R:` field:

```
%%select R:.*jig
```

More precisely, the above line means: select all tunes that contain an `R:` field, possibly followed by one or more spaces (the `.*` bit), followed by "jig". The corresponding command line option is:

```
abcm2ps -c -O jigs.ps -e "R:.*jig" tunebook.abc
abcm2ps-8.14.2 (2018-12-18)
File tunebook.abc
Output written on jigs.ps (31 pages, 64 titles, 340348 bytes)
```

To select all tunes in the key of A minor, we will use:

```
%%select K:.*Amin
```

To select all tunes that have the word "Humours" in the title, we will use:

```
%%select T:.*Humours
```

> ☞ **Warning**
>
> Global parameters and parameters between tunes will be included in the output of a
> `%%select` directive, probably causing unexpected results!

### 5.5.1   Incipits

Another type of tune selection is an *incipits file*, which is a special tunebook containing only
the first measures of each tune. Incipits files are especially useful for musicians who learn
the music by heart, but need a tune list before or during the performance. "Incipit" is Latin
and it means "(it) begins".

This feature is provided by a couple of directives working together: `%%tune` and `%%clip`.
The first is similar to the `%%select` directive we examined, and it is followed by a regular
expression to specify the tunes we are interested in. The second is followed by a *symbol
selection*: in its simplest form, the number of measure bars we want to print.

So, if we want to print only the first four measures of each tune in a tune collection, at
the top of the file we will write the following lines:

```
%%tune .*
%%clip 1-5
```

In plain English, these mean: select all of the following tunes (the `.*` bit), and print only
the measures included between the first and the fifth measure bars.

To print incipits of tunes marked as jigs in their `R:` field, we will do:

```
%%tune R:.*jig
%%clip 1-5
```

and similarly for any regular expression we may invent.

♫  ♫  ♫  ♫  ♫

# Chapter 6

# Playing

## 6.1 MIDI Conversion

Iᴺ ᴀᴅᴅɪᴛɪᴏɴ to printable output, Aʙᴄ can also produce audible output; specifically, MIDI files.

You might be surprised to learn that MIDI files *are not audio files*, like (say) MP3! In fact, 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. In fact, 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:

> ☞ **Warning**
>
> 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 software that is used to listen to it.

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

To this end, we use the free `abc2midi` program. 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.

`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 Aʙᴄ source files;

- midi2abc: conversion of MIDI files to Aʙᴄ;

- yaps: a command-line formatter analogous to `abcm2ps`, but less powerful.

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

### 6.1.1   A Software MIDI Player: TiMidity++

Some advanced sound cards have a built-in MIDI syntesizer, 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.

Unfortunatley, 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 and/or lack some voices.

Replacing the missing instruments is easy. By default, TiMidity++ 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.

As a better solution, you may want to install a more complete patch set. Most GNU/Linux distributions provide two packages called `fluid-soundfontgm` and `fluid-soundfont-gs`. Install the packages and configure TiMidity++ to use them by modifying `timidity.cfg`:

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

# alternatively, you can use the fluid-soundfont:
source /etc/timidity/fluidr3_gm.cfg
source /etc/timidity/fluidr3_gs.cfg
```

An alternate program to play MIDI files is FluidSynth, http://www.fluidsynth.org, which also uses the Fluid patches.

### 6.1.2   Our First Midi

Do you remember the first scale and the command-line instructions we examined in Section 1.2.4 (p. 8)? 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 of the tune. If you feed `abc2midi` a tune collection, a MIDI file will be created for each tune.

To play the MIDI file from the command line:

```
timidity scale11.mid
```

### 6.1.3  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 transfered `Ring1.mp3` to my smartphone.

> ✍ **Note**
>
> In the following sections, I will include some sound samples. In theory, MIDI files cre-
> ated 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 lis-
> ten to them. Therefore, all MIDI files were converted to MP3 files with Lame, as shown
> above.

### 6.1.4   Supported Decorations

Some of the decorations described in Section 2.2.11 (p. 41) are supported by `abc2midi` and
produce audible output:

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

Default volume is equivalent to `!f!`.

### 6.1.5   `%%MIDI` **Directives**

Just as `abcm2ps` provides directives for changing page layout details, `abc2midi` provides
directives 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.

   Directives can be written in two ways. One is the usual meta-comment syntax: a single
line containing `%%MIDI ⟨directive⟩ ⟨parameters⟩` is entered. The second way is using
an inline I field: `[I:MIDI = ⟨directive⟩ ⟨parameters⟩]` (spaces are optional). Note the
= character.

   To clarify, the two following sources are equivalent:

```
X: 1
T: MIDI directives as meta-comments
L: 1/4
K: C
%%MIDI program 1
CDEF|
%%MIDI program 109
GABc|
```

```
X: 1
T: MIDI directives 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.

### 6.1.6 Avoiding abcm2ps Extensions

Inevitably, a music typesetter is more complex and supports more features than a MIDI converter. As a result, in some cases abc2midi can't handle directives that only make sense in printed output.

For example, think of a block of text lines defined by %%begintext and %%endtext (see Section 4.2.14, p. 73). When abc2midi finds these directives, it simply ignores them. However, it will try and convert the text lines in between to MIDI, which will result in lots of error messages and garbage output. (Note that %%text lines will be rightly ignored with no side effects.)

To avoid this and other possible mishaps, use the %%MidiOff directive to tell abc2midi to ignore the next lines until the %%MidiOn directive is found:

```
...
%%MidiOff
%%begintext
  This text will be printed by abcm2ps,
  but will not be converted to MIDI.
%%endtext
%%MidiOn
...
```

### 6.1.7 Voices and Instruments

Let's consider "Ave Verum Corpus", which we examined in Section 3.1.2 (p. 48). 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 directives is %%MIDI program, which associates a voice with a particular instrument. Let us add these directives to associate each voice with the right instrument:

```
X: 1
T: Ave Verum Corpus
C: W. A. Mozart
M: 4/4
L: 1/4
Q: "Adagio"
%%score [(S A) (T B)] (MD1 MD2) | (MS1 MS2)
%%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
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
%
% ... 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 directives follow that associate each voice with an appropriate MIDI instrument, MIDI 54 ("Choir Oohs") or 20 ("Church Organ").

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.7 (p. 162). 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.

### 6.1.8   Changing Instruments in Repeats

Let's suppose we have a repeat, and we want it to play with different instruments the first and second time. We will have to insert a %%MIDI program directive just before the bar that ends the repeat.

In the example below, the first run is played as MIDI instrument 21 (accordion), the second as MIDI instrument 73 (flute):

```
X: 1
M: 4/4
L: 1/4
K:C
%%MIDI program 21
|: CDEF | GABc [I:MIDI program 73] :|
```

Unfortunately, there is no way to change instruments when repeats occur three or more times.

### 6.1.9   Accompaniment Chords

The accompaniment chords described in Section 2.2.9 (p. 37) are used by abc2midi to generate an accompaniment to the main melody. A lowercase 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.

Currently recognised chords are:

```
m 7 m7 m7b5 maj7 M7 6 m6 aug + aug7 dim dim7 9
m9 maj9 M9 11 dim9 sus sus4 sus9 7sus4 7sus9 5
```

Additional chords can be defined with %%MIDI chordname, explained in Section 6.1.10 (p. 99).

Accompaniment chords are rendered by abc2midi as a *sequence of fbcz* for each measure. f stands for the fundamental or root of the chord, b for the fundamental and the chord played together, c for the chord itself, and z for a rest. A fbcz sequence is designed to match a time signature: for example, in 4/4 time the accompaniment is fzczfzcz: fundamental, rest, chord, rest, fundamental, rest, chord, rest.

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

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 directive. 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 directive 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: 1/4=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's 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's 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: <span style="color:blue">garota.mp3</span>.

## 6.1.10  New Accompaniment Chords

The `%%MIDI chordname` directive allows us 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 <span style="color:red">6.1.9</span> (p. <span style="color:red">97</span>), ⟨*n1*⟩ is the chord fundamental and and the other notes (up to 6) are expressed as semitones above the fundamental.

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

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

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

## 6.1.11  Customising Beats

MIDI files usually sound artificial and expressionless, but there are several ways to improve them. The `%%MIDI beatstring` ⟨*fmp*⟩ directive 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*⟩ directive 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 %%MIDI nobeataccents directive. To resume it, use %%MIDI beataccents.

### 6.1.12   Arpeggios

In addition to fcz sequences, we can also specify *ghijz sequences* that allow us to play the individual notes comprising the accompaniment chord. This way, we can 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. Uppercase 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, we can use fcz and ghij sequences together, like fcbghijGHIJz.

Another way to play chords as arpeggios is the %%MIDI chordattack directive. When followed by a number $\langle n \rangle$ greater than 0, it delays each note by $\langle n \rangle$ MIDI time units (a quarter note is 480 MIDI units). Using %%MIDI randomchordattack $\langle n \rangle$ varies the delay of each note randomly between 0 and $\langle n \rangle$.

### 6.1.13 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's 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](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 `A3/2 B/2` but is actually played as `A4/3 B2/3`. 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`.

### 6.1.14 Drum Patterns

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

The `%%MIDI drum` directive is followed by a sequence of `dz`, where `d` represents a percussion beat and `z`, predictably, a rest. After the sequence, we write the codes for the desired percussion instruments (see Appendix A.7.2, p. 163) 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'|
```

Fractional length cannot be specified: to indicate a rhythm such as (3ddd d/d/d/d/ we must use the d4d4d4d3d3d3d3 sequence.

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 cccccccc
%%MIDI drumon
"C"CC EE GG AA|_BB AA GG EE|
CC EE GG AA|_BB AA GG EE|
"F"FF AA cc dd|"F7"_ee dd cc AA|
"C"CC EE GG AA|_BB AA GG EE|
%%MIDI gchordoff % no chords
"G"GG BB dd Bd|"F7"FF AA cc Ac|
%%MIDI gchordon  % turn chords back on
"C"CC EE GG AA|_BB AA GG EE|
%%MIDI gchord c8
%%MIDI program 1 60 % Brass Section
%%MIDI drumoff
!fermata!"C"[C8E8G8c8]|
```

### 6.1.15   Percussion Instruments and Drum Mapping

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 and correspond to a certain pitch. However, the special MIDI channel 10 has a peculiar feature: each note is associated with a different percussion instrument, rather than a pitch. Playing that note actually plays the corresponding percussion instrument; for example, playing A,, produces a Low Tom beat. Notes and corresponding percussion instrument are listed in Section A.7.2 (p. 163).

A convenient directive, `%%MIDI drummap`, is used to change the mapping between notes and instruments. Some notes, in fact, are too low or too high; writing runs of these notes would be a bit awkward. The following tune has a melody played on the piano, and low-tom, hi-hat and triangle accompaniment. We only use one note, `^A`, conveniently remapped as needed ([percussion.mp3](percussion.mp3)):

```
X: 1
L: 1/4
Q: 1/4 = 120
V: 1 name="Piano" % normal voice
V: 2 clef=perc name="Low Tom"        stafflines=1
%%MIDI channel 10
%%MIDI drummap ^A 45
V: 3 clef=perc name="Open Hi Hat"    stafflines=1
%%MIDI channel 10
%%MIDI drummap ^A 46
V: 4 clef=perc name="Open Triangle" stafflines=1
%%MIDI channel 10
%%MIDI drummap ^A 81
K: C
%
[V:1] CDEF |GABc | % Piano
[V:2] ^Az^Az |^Az^Az | % Low Tom
[V:3] z^Az^A |z^Az^A | % Open Hi Hat
[V:4] ^A/^A//^A//^A//^A/^A/^A/^A/^A/|\
      ^A/^A//^A//^A//^A/^A/^A/^A/^A/| % Open Triangle
```



Note that the `%%MIDI channel 10` directive must be written *after* the `V:` field that starts a new voice.

### 6.1.16  Portamento

A glide effect can be added using the `%%MIDI portamento [bass] [chord] ⟨n⟩` directive, 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 PFM. The first repeat has no glide effect, while the second repeat mimics the original Moog glide ([impressioni.mp3](impressioni.mp3)):

```
X: 1
T: Impressioni di Settembre (riff)
C: PFM, 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-|gfed- 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.

### 6.1.17   Drone

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

- %%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 directive 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 6.1. 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,, | ˆA,, | B,, | C, | ˆC, | D, | ˆD, | E, | F, | ˆF, | G, | ˆG, | A, |
|------|-----|------|-----|----|-----|----|-----|----|----|-----|----|-----|----|
| MIDI pitch | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 |

Table 6.1: 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](amazinggrace.mp3).

### 6.1.18   Global Settings

In a tune collection, the same MIDI settings could be shared by all or most tunes. Instead on inserting the same %%MIDI lines in each tune, it is possible to place them at the beginning of the file; the parameters will apply to all tunes. These are called "global settings".

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.

## 6.2   Differences and Incompatibilities

Unfortunately, abcm2ps/abc2svg and abcMIDI are not completely compatible with each other, because the scorewriters accept a more extended syntax. Moreover, it should be pointed out that some indications only make sense in a printed score. Consequently, when writing music in Aʙᴄ bear in mind that something could not work as expected. A few things that do not work in abcMIDI are:

- nested tuples;

- "grace chords" (Section 2.1.10, p. 22) are not played correctly;

- repeats indications like |::  ::|;

- P: fields cannot be inserted within repeats;

- several decorations (e.g. tremolos) 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;

- repeated sequences (Section 4.2.6, p. 66) are not played;

- ...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 will be explained in Section 9.9 (p. 137).

♫ ♪ ♫ ♪ ♫

# Chapter 7

# Advanced Customisation

SOMETIMES, it could be desirable to use musical features that are not directly supported in ABC; or to modify the output in specific ways. This is possible, since abcm2ps and abc2svg provide directives that modify or extend their functionality. For example, users can redefine note shapes, add colours and decorations, and implement new features. In general, users can add or modify PostScript and/or SVG routines for output customisation.

abcm2ps has full PostScript customisation, but it also understands some SVG. Simmetrically, abc2svg has full SVG customisation, but it also understands some PostScript. It must be stressed that these extensions *are not part of the ABC notation*, and are specific to abcm2ps and abc2svg only!

This chapter is only useful for expert users who are can write code.

## 7.1   New PostScript and SVG Routines

The %%ps directive, followed by code in the PostScript language, adds new routines or redefines existing ones. For example, the following commands redefine abcm2ps's internal routine dlw that causes all lines in the score to be drawn thinner:

```
%%ps /dlw
%%ps {0.3 setlinewidth} bind def  % default: 0.7
```

so the score looks better at high values of %%pagescale. This extension, however, will not work in abc2svg.

When writing several lines of PostScript code, using the sequence %%beginps... %%endps is recommended. In addition to PostScript, commands in SVG can also be specified using %%beginsvg... %%endsvg. Whenever possible, you should write both SVG and PostScript commands to make the source compatible with both abc2svg and abcm2ps.

### 7.1.1   Defining New Symbols

The %%deco directive adds new expression symbols, using PostScript/SVG routines defined by abcm2ps and abc2svg or new ones written by the user. The syntax is:

%%deco ⟨*name*⟩ ⟨*type*⟩ ⟨*function*⟩ ⟨*height*⟩ ⟨*wl*⟩ ⟨*wr*⟩ ⟨*string*⟩

where:

- ⟨*name*⟩ is the name of the new symbol, without the exclamation marks;

- ⟨*type*⟩ is an integer number 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 the staff; 6 and 7, expressions linked to the staff (like dynamic marks); 8, long decorations; 10, colour. To give an idea of symbol positioning, here is a listing:

  - 0: like `!tenuto!` or staccato dot;
  - 1: like `!slide!`;
  - 2: like `!arpeggio!`;
  - 3: generic expressions;
  - 4: below the staff;
  - 5: like `!trill(!` or `!trill)!` (abcm2ps only);
  - 6: generic;
  - 7: like long dynamics symbols;
  - 8: like long dynamics symbols;
  - 10: note colour.

- ⟨*function*⟩ is the name of the PostScript/SVG routine that draws the symbol;

- ⟨*height*⟩ expression height in points;

- ⟨*wl*⟩ and ⟨*wr*⟩ are not used;

- finally, ⟨*string*⟩ is an optional text string.

Let's 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` routine:

```
%       name  type   function  height  wl  wr  string
%%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



`%%deco` lines implement three new symbols: `!fp!`, `!(f)!`, and `!(ff)!`.

Let's 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.

```
%%beginsvg
<defs>
% triangle
<path id="newdot" class="fill"
  d="m -2 2 l 2 5 l 2 -5 l -4 0" />
% upside-down V
<path id="toe" class="stroke"
  d="m -5 -2 l 5 -5 l 5 5"  />
</defs>
%%endsvg

%       name  type  function  height  wl  wr  string
%%deco tu    0     newdot    5       0   0
%%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|]
```



## 7.2   Note Mapping

The appearance and position of notes can be programmed using the powerful %%map directive. New shapes can be programmed with %%beginps…%%endps and %%beginsvg…%%endsvg.

Note mapping matches one or more notes with new note shapes that are defined within a named set, called a *map*. A voice can then be matched with a map; notes belonging to the voice will be printed using the new shapes and/or positions.

The syntax is:

%%map ⟨*map_name*⟩ ⟨*mapped_note*⟩ *[heads=]*⟨*glyph names*⟩ *[print=]*⟨*note*⟩ *[color=]*⟨*#rrggbb*⟩

where:

- ⟨*map_name*⟩ is a text string;

- ⟨*mapped_note*⟩ is the Abc name of a note, complete with accidentals if needed. The note name can also be *, meaning "all notes";

- *[heads=]*⟨*glyph names*⟩ defines a set of glyph names. Glyphs are defined as PostScript or SVG paths;

- *[print=]*⟨*note*⟩ indicates that ⟨*mapped_note*⟩ will be printed as ⟨*note*⟩ (that is, on a different line or space);

- *[color=]⟨#rrggbb⟩* defines note colours.

It sounds complicated, but some examples will hopefully clarify how %%map works. Let's define two maps; one that prints notes heads C D E in green, and another that prints all note heads in red:

```
X:1
M:C
L:1/4
K:C
%  first map: only C D E note heads printed in green
%%map color_green C color=#00ff00
%%map color_green D color=#00ff00
%%map color_green E color=#00ff00
%  second map: all note heads printed in red
%%map color_red * color=#ff0000
%
[V:1][I:voicemap color_green] "^V:1 is mapped to color_green"CDEF|GABc|
[V:2][I:voicemap color_red]   "^V:2 is mapped to color_red"CDEF  |GABc|
% mapping continues
[V:1] cBAG|FEDC|
[V:2] cBAG|FEDC|
```
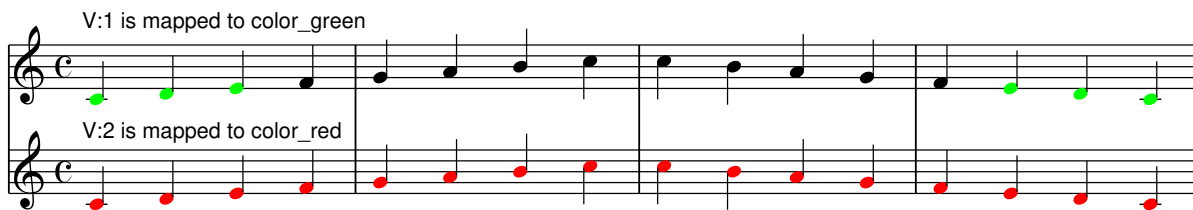


In the next example we define a new cross-shaped note head, and we map some notes to new positions on the staff. This is a simple implementation of *percussion notes*, that are notated using peculiar note heads and positions.

```
%%beginsvg
<defs>
<path id="x_head"
  d="m-3 -3l6 6m0 -6l-6 6"
  class="stroke" style="stroke-width:1.2"/>
</defs>
%%endsvg

% glyph is referred to as "x_head"

X:1
M:4/4
L:1/4
%%score (1 2)
K:C clef=perc
%
%  Note ^A, (hi-hat) will have its head replaced by
%  the 'x_head' glyph, and it will be printed as 'g'
%%map drum ^A, print=g heads=x_head
%  Note D, (snare) will be printed as 'c'
```

```
%%map drum  D, print=c
%  Note C, (bass drum) will be printed as 'F'
%%map drum  C, print=F
%
[V:1] [I:voicemap drum][I:MIDI channel 10]
^A,/^A,/[^A,D,]/^A,/ ^A,/^A,/[^A,D,]/^A,/ |
[V:2] [I:voicemap drum][I:MIDI channel 10]
C,xC,x |
```
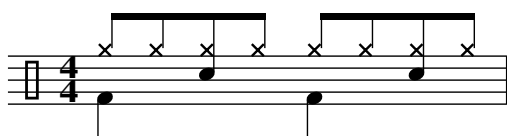
As you can see, notes `^A, D, C,` are printed as `g c F`; besides, `^A,` has a cross-shaped note head. The new note positions are standard in percussion notation; further, the `I:MIDI` inline fields map the notes to percussion instruments (Section 6.1.15, p. 102).

Summing up, the example above is displayed correctly and it also plays correctly when it is converted to MIDI.

## 7.3  Renaming Symbols

For historical reasons, several symbols have a long name; for instance, `!invertedfermata!`. We can define abbreviations employing the method we learnt in the previous section:

```
%%deco fmt  3  hld  12 7 7    % !fmt!  = !fermata!
%%deco lmd  3  lmrd 10 2 2    % !lmd!  = !lowermordent!
%%deco umd  3  umrd 10 2 2    % !umd!  = !uppermordent!
%%deco ,    3  brth 0 1 20    % !,!    = !breath!
%%deco lphr 3  lphr 0 1 1     % !lphr! = !longphrase!
%%deco mphr 3  mphr 0 1 1     % !mphr! = !mediumphrase!
%%deco sphr 3  sphr 0 1 1     % !sphr! = !shortphrase!
%%deco ifmt 3  hld 12 7 7     % !ifmt! = !invertedfermata!
%%deco trn  3  turn 10 0 5    % !trn!  = !invertedturn!
%%deco itrn 3  turnx 10 0 5   % !itrn! = !invertedturnx!
%%deco mrd  3  lmrd 10 2 2    % !mrd!  = !mordent!
%%deco arp  2  arp 12 10 0    % !arp!  = !arpeggio!
%%deco inv  32 0 0 0 0        % !inv!  = !invisible!
```

We have renamed the symbol definitions that are defined in one of `abcm2ps` source files (`deco.c`).

> ☞  **Warning**
>
> If we rename a symbol, `abc2midi` can't recognise it any more. Usually this is not a problem, since many symbols only produce printed output. However, other symbols like `!arpeggio!` also produce MIDI output. Hence, in such cases we should use `U:` to redefine the symbol.

# 7.4   Accompaniment Chords in Italian Notation

By default, accompaniment chords are printed using English note names. Chord names can be redefined using the PostScript commands (abcm2ps) or the %%chordnames directive (abc-2svg).

The following code is a format file for abcm2ps that redefines accompaniment chords using Italian note names:

```
% itachords.fmt
%
% Displays guitar chords using Italian note names.
% Written by Jean Fran\c cois Moine
% License: GNU GPL 2.

beginps
  /gcshow{
    dup 0 get
    dup dup 65 ge exch 71 le and{
    65 sub[(La)(Si)(Do)(Re)(Mi)(Fa)(Sol)]exch get show
    dup length 1 sub 1 exch getinterval
  } if show}!
  /agcshow{
    dup 0 get 0 get
    dup dup 65 ge exch 71 le and{
    65 sub[(La)(Si)(Do)(Re)(Mi)(Fa)(Sol)]exch get show
    dup length 1 sub 1 exch getinterval
  } if arrayshow}!
endps

% end of file itachords.fmt
```

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

```
X:1
L:1/4
K:Ebmaj
M:4/4
"Eb"EFGA|"Bb"Bcde|"Cm"CDEF|"Gm"GABc|
```



Unfortunately, this method will not accept UTF-8 characters.

abc2svg supports the %%chordnames directive that also accepts UTF-8 characters. For instance, chord names in French can be entered as:

```
%%chordnames Do,Ré,Mi,Fa,Sol,La,Si
```

## 7.5   Adding Fonts

Standard Ghostscript fonts are usually enough for most users. However, to add a special touch to our scores, more fonts can be added. Recent versions of Ghostscript support PostScript (`.pfb`), OpenType (`.otf`), and TrueType (`.ttf`) fonts.

> ✍ **Note**
>
> A few notes before we begin:
>
> 1. the following procedure will not work on the Android platform, which lacks Ghostscript support;
>
> 2. Ghostscript does not use system fonts, i.e. the ones that the operating system provides to all other applications. You will have to add new fonts following the procedure outlined in this section;
>
> 3. the procedure was tested on GNU/Linux Mint 18.3 (Ghostscript 9.18), Windows 7 and Windows 10 (Ghostscript 9.26 installed in `C:\gs\gs9.26`). You will need root/administrator privileges.

An excellent site boasting a wide collection of free and high-quality PostScript and True-Type fonts is http://www.moorstation.org/typoasis/typoasis1.htm. Under the "Designers" section, you'll find several pages devoted to font designers; each page offers many fonts for download. Select Dieter Steffmann's page.

Let's see how to add a new font called Holla Script, downloaded as `Holla.zip`. Obviously, the procedure will be very similar with other fonts.

Unzip the archive and copy `HollaScript.pfb` (the TrueType version, `HollaScript.ttf`, will also work) to one of the directories where Ghostscript searches for fonts. Supposing that you installed Ghostscript (let's assume version 9.26) and its fonts in default locations, such directory is:

- Windows: `C:\gs\gs9.25\lib`

- GNU/Linux: `/usr/share/fonts/`

To list all available search directories, we can issue the command:

```
gs -h
GPL Ghostscript 9.25 (2018-09-13)
...
Search path:
   /usr/share/ghostscript/9.25/Resource/Init :
   /usr/share/ghostscript/9.25/lib :
   /usr/share/ghostscript/9.25/Resource/Font :
   /usr/share/ghostscript/fonts : /var/lib/ghostscript/fonts :
   /usr/share/cups/fonts : /usr/share/ghostscript/fonts :
   /usr/local/lib/ghostscript/fonts : /usr/share/fonts
```

Subdirectories are also searched by Ghostscript.

Now edit Ghostscript's font list, which is a simple text file:

- Windows: `C:\gs\gs9.25\lib\Fontmap`

- GNU/Linux: `/var/lib/ghostscript/fonts/Fontmap`

Note that this file does not have a `.txt` extension. If the file is missing, create a new one. Move to the bottom of the file and add this line:

`/HollaScript (HollaScript.pfb) ;`

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

`/Jazz /HollaScript ;`

We are now ready to use the HollaScript font in Abc files. First of all, we will have to declare it using the `%%font` directive followed by the font name. Look at this:

```
%%font HollaScript
%%titlefont HollaScript 24
%%textfont HollaScript 18
%%composerfont Jazz 16 % alias
%%vocalfont Jazz 16
%%gchordfont Jazz 18
X: 1
T: Test: HollaScript font (Jazz)
L: 1/4
K: C
%
"C"CDEF|"G"GA"G7"Bc|"C"cBAG|"G"FEDC|"C"C4|]
w: Do Re Mi Fa... |||||
%%text ABCDEFGHIJKLMNOPQRSTUVWXYZ
%%text abcdefghijklmnopqrstuvwxyz 1234567890
```



Bear in mind that some fonts available 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.

## 7.6   Using SMuFL Fonts

By default, `abcm2ps` and `abc2svg` use their own libraries of symbols (glyphs) to draw notes and other music elements. Generally, these glyphs look quite good, but there are a few exceptions; `abcm2ps` piano symbols, for instance, are quite rudimentary.

To change the appearance of our scores, we can replace the glyphs using SMuFL-compliant music fonts. SMuFL stands for Standard Music Font Layout (https://www.smufl.org); it is a specification designed to provide a standard way of mapping musical symbols to characters. Fonts following the SMuFL specs can be used by abcm2ps and abc2svg.

Bravura is a free SMuFL-compliant font, available in several formats from https://github.com/steinbergmedia/bravura. The redist/otf directory contains the latest release in Open-Type format, Bravura.otf, which works with both Ghostscript and SVG; hence, with abcm-2ps and abc2svg. Other SMuFL-compliant fonts are available on the web; not all appear to work correctly, though. An ancillary file, BravuraText.otf, is used to insert Bravura symbols in the text.

To be usable by abcm2ps, the font must be installed in one of Ghostscript search directories, as explained in Section 7.5 (pag. 113). Adding the font to the Fontmap file is not required. On Windows, the font can be installed normally by double clicking on it; this is called a "local" font installation.

The %%musicfont directive instructs abcm2ps and abc2svg to use the new glyphs provided by Bravura:

```
%%pagescale 1.2
%%musicfont url(/usr/share/fonts/Bravura.otf)

X: 1
T: Using the Bravura music font
M: 4/4
L: 1/4
K: C
%
%%text New note heads
%
C4 | =C2 _C ^C | [CDEFGA]2 [CDEFGA] [GABcde] |
%
%%text Some decorations
%
C !segno!E !invertedturn!G !turn!c | \
!ped!C !fermata!E !ped-up!G !^!c | \
!mordent!C !uppermordent!E !D.S.!G !coda!c  |
```



Using the Bravura music font

New note heads



Some decorations
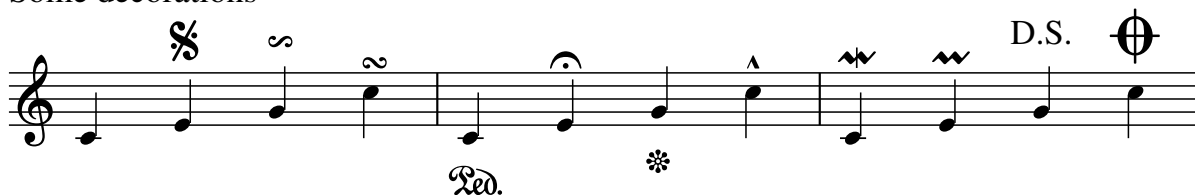
As you can see, note heads and decorations are different than the default glyphs.

Not all symbols, though, are replaced by the SMuFL font; for example, volume decorations like !p! are normally implemented as strings. The `bravura.fmt` format file, available at the Abc 2 home page, provides more symbols.

## 7.7   Fingering Charts

A *fingering chart* is a simplified notation format that indicates instrument fingerings instead of notes. It is meant to be easier than standard notation, and it is usually employed by folk musicians and in introductory music courses for children.

abcm2ps supports native fingering charts for the tin whistle, and provides support for user-defined charts. The latter are quite complex to implement, and will be covered in future releases of this manual.

### 7.7.1   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.

The format file `flute.fmt` contains fingerings definitions for tin whistles in several keys; for the *galoubet*, a French three-hole fipple flute, and other three-hole instruments (English pipe (and tabor) and Basque txistu).

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 outlined in Table 7.1.

| **Digit** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| **Key** | D | C | E♭ | B♭ | F | G | A | C (galoubet) |

Table 7.1: Keys for the `flute.fmt` format file.

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 8

# Abc and MusicXML

## 8.1   Introducing MusicXML

L ET's suppose we want to exchange our music written in Abc with a friend who uses a commercial music program. Most probably, our friend will be unable to load our Abc sources into their program. This is a typical problem that plagues people using different programs running on different operating systems.

Besides, occasionally we may need to use a different music typesetter. Although `abcm-2ps` and `abc2svg` produce excellent scores most of the time, in some cases other programs produce better-looking results. The same holds for MIDI output.

In these cases, MusicXML is a possible solution. MusicXML is a file format that was designed for the exchange of sheet music files between music applications. Most major programs support it, including market leaders Sibelius® and Finale®, and popular freeware Finale NotePad. MuseScore and LilyPond, the most powerful scorewriters in the open source world, are a precious alternative.

`abc2xml.py` and `xml2abc.py` are the gateway between Abc and MusicXML-enabled programs. The `abc2xml.py` translator supports nearly all elements of the Abc language, implements some directives, and provides some useful extensions. Similarly, `xml2abc.py` implements the parts of MusicXML that are also included in Abc; the former, in fact, supports more features than Abc. While you should not expect to obtain a perfect conversion all the time, the Abc/MusicXML translators work impressively well: manual customisation of the resulting files is usually trivial.

## 8.2   Using `xml2abc-js`

Besides `abc2xml.py` and `xml2abc.py`, we can use a convenient MusicXML to Abc online converter called `xml2abc-js`. It's a Javascript version of `xml2abc.py`, and is available at https://wim.vree.org/js/xml2abc-js.html.

Let's use it to convert one of the MusicXML test files available here: http://www.musicxml.com/music-in-musicxml/example-set/. Click on the Recordare Editions menu and download the MusicXML version of one of the provided samples; say, Debussy's "Mandoline". Save the file as `mandoline.xml`.

Open the `xml2abc-js` page, click on the Choose File button and load `mandoline.xml`. The file will be immediately translated to Abc; the source will be displayed on the left, the score will be typeset using `abc2svg` and displayed on the right.

You can now save the source using the save button, and use the Options button to exper-
iment with xml2abc-js' translation options.
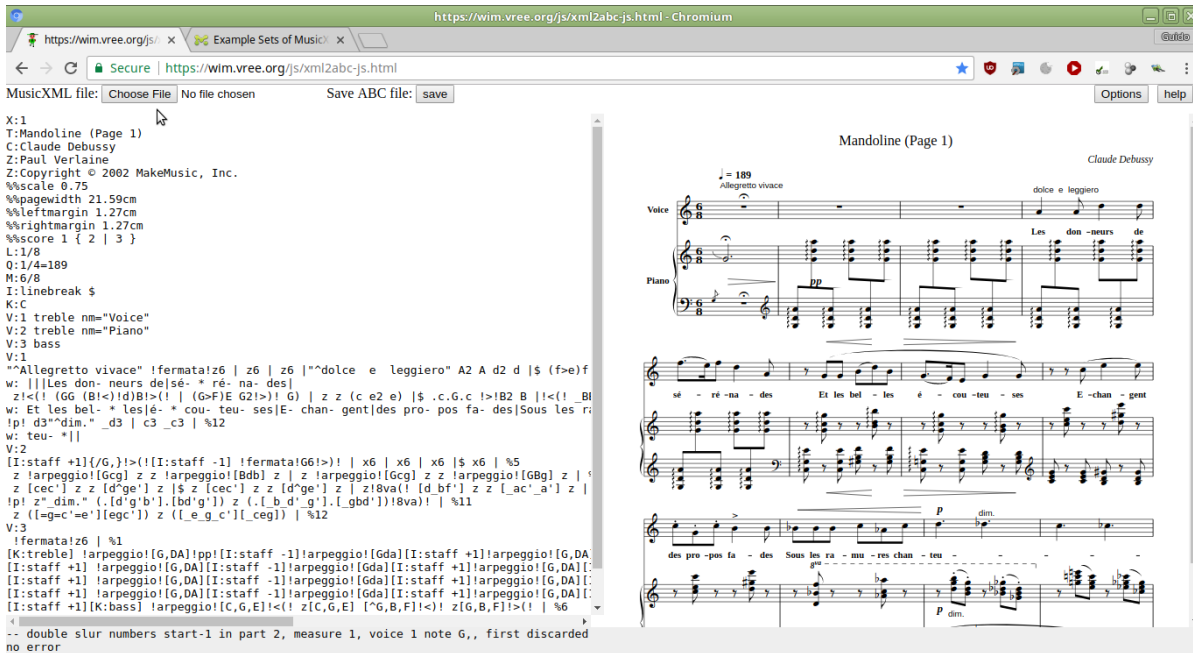


Figure 8.1: Importing and displaying MusicXML files in xml2abc-js.

## 8.3   Using abc2xml.py and xml2abc.py

abc2xml.py and xml2abc.py are command line programs that require Python. The syntax
is:

```
$ abc2xml.py [switches] -o. file.abc
$ xml2abc.py [switches] -o. file.xml
```

The following command, for example, translates a set of MusicXML files in current direc-
tory into the corresponding ABC sources:

```
~/XML$ xml2abc.py -o. *.xml
```

In plain English: for each .xml file, xml2abc.py creates the corresponding .abc file in
the current directory. xml2abc.py also converts compressed MusicXML files, which have a
.mxl extension.

This command performs the opposite conversion from .abc to .xml:

```
~/Abc$ abc2xml.py -o. *.abc
```

Omitting the -o. switch (output to files in current directory), output can be redirected.
This command creates a single ABC file from .xml files:

```
~/XML$ xml2abc.py *.xml > alltunes.abc
```

The above commands work on GNU/Linux and macOS, and possibly on Windows systems
with Bash and Python installed. Alternatively, we can use EasyABC, which can import and
export MusicXML using the translators internally.

### 8.3.1  Basic Options

Running `abc2xml.py` with the `-h` switch gives a synopsis of its options. The most useful are:

- `-m` ⟨*n1*⟩ ⟨*n2*⟩: if given a tune collection, skip ⟨*n1*⟩ tunes and read at most ⟨*n2*⟩ tunes. By default, `abc2xml.py` reads the first tune only.

- `-p` ⟨*page format*⟩: specify the page geometry. ⟨*page format*⟩ is a set of 7 decimal numbers that specify the scale, page height, page width, left margin, right margin, top margin, and bottom margin in mm The numbers are separated by commas, without spaces. Default values are `0.75,297,210,18,18,10,10` (European A4 paper size).

- `-b`: break lines at end of line. Equivalent to the `I:linebreak <EOL>` field.

Similarly, running `xml2abc.py` with `-h` lists its options. The most useful are:

- `-d` ⟨*N*⟩: set the `L:` field to 1/⟨*N*⟩.

- `-n` ⟨*NC*⟩: set the maximum number of characters per line to ⟨*NC*⟩ (default: 100).

- `-b` ⟨*NB*⟩: set the maximum number of bars per line to ⟨*NB*⟩.

- `-x`: don't output line breaks.

- `-p` ⟨*page format*⟩: set the page geometry. ⟨*page format*⟩ works as with `abc2xml.py`.

Running the translators without switches usually gives good results.
ABC music elements are translated, but only a few directives are:

- `%%pagewidth`, `%%pageheight`, `%%leftmargin`, `%%rightmargin`, `%%score`, `%%staves`, `%%scale`, and `%%abc-include`.

- `%%MIDI program` and `%%MIDI channel` (with some limitations), `%%MIDI transpose`, `%%MIDI drummap`.

These commands suffice to produce full-featured MusicXML files that can successfully imported in other applications.

## 8.4  Converting ABC to MuseScore

MuseScore is a multiplatform, free and open source visual scorewriter; it's one of the most powerful and high-quality music programs available. Its home page is http://musescore.org/.

Since MuseScore can read MusicXML, it can be used to typeset music originally written in ABC. In the following example we convert `scale1.abc` to MusicXML, run the resulting file through MuseScore, and produce a PDF file:

```
$ abc2xml.py -o. scale1.abc
-- decoded from utf-8
-- skipped header: (field X,1)
./scale1.xml written
-- done in 0.02 secs
```

```
$ musescore scale1.xml -o scale1.pdf
initScoreFonts 0x30d33c0
convert <scale1.xml> to <scale1.pdf>
setFirstInstrument: no instrument found for part 'P1'
~$ _
```

As you can see, we typeset the score from the command line using MuseScore instead of abcm2ps. Alternatively, we can run MuseScore interactively and load the MusicXML source for manual customisation and PDF export. In particular, lyrics and dynamics may clash; you will have to reposition dynamics symbols by hand.

MuseScore can also convert to other formats: `png`, `svg`, `wav`, `flac`, `ogg`, `mp3`, `mid`, `xml`, and `mxl`. In the following examples, scores will be typeset by MuseScore.

## 8.5   Converting Abc to Lilypond

Lilypond is a very powerful, free and open source music typesetter. It employs its own text-based music notation language, and produces professional-grade sheet music. While Abc is somewhat specialised in folk music, Lilypond is widely used for notating classical music. Its home page is [http://lilypond.org/](http://lilypond.org/).

Lilypond notation can become quite complex, especially for orchestral scores. Writing music in Abc notation is much simpler; conversion to Lilypond format can be beneficial for both Abc and Lilypond users. Lilypond includes a utility called `abc2ly` that converts Abc files to Lilypond (`.ly`) files, but this program does not work very well with multi-voice files.

In this case, using MusicXML as an intermediate format produces better results. In addition to `abc2xml.py`, one uses the `musicxml2ly` command also provided by Lilypond. The following commands convert `scale1.abc` to MusicXML, run the resulting file through `musicxml2ly` to convert it to Lilypond format, then run Lilypond to produce a PDF file:

```
$ abc2xml.py scale1.abc > scale1.xml
-- decoded from utf-8
-- skipped header: (field X,1)
$ musicxml2ly scale1.xml
musicxml2ly: Reading MusicXML from scale1.xml ...
musicxml2ly: Converting to LilyPond expressions...
musicxml2ly: Output to `scale1.ly'
$ lilypond scale1.ly
GNU LilyPond 2.18.2
Processing `scale1.ly'
Parsing...
Interpreting music...
Preprocessing graphical objects...
Finding the ideal number of pages...
Fitting music on 1 page...
Drawing systems...
Layout output to `scale1.ps'...
Converting to `./scale1.pdf'...
Success: compilation successfully completed
$ _
```

LilyPond's own MusicXML translator is apparently less capable than MuseScore's, and occasionally it may fail to convert the input. In most cases, though, conversion will be quite good.

> ✎ **Note**
>
> Just to confirm that ABC is just a notation and is not tied to a specific program, we now have four different programs to typeset our music!

## 8.6 `abc2xml.py` Extensions

Since MusicXML supports more musical features than ABC, `abc2xml.py` provides a few extensions that MusicXML applications can exploit.

### 8.6.1 Tremolo

Tremolo indications `!/-!` `!//-!` `!///-!` are similar to `!trem1!` `!trem2!` `!trem3!` `!trem4!` that are supported by `abcm2ps` and `abc2svg`, but are placed differently:

```
X: 1
L: 1/4
M: none
K: C
%
!/-!CD !//-!EF !///-!GA !/!C !//!D !///!E|
```

That is, in front of note pairs instead of between notes pairs. Using MuseScore to convert the corresponding MusicXML file, we obtain the following score:



(can you see that the glyphs are slightly different?) and this MIDI file: trem.mp3. Unfortunately, `abc2midi` does not support any of the tremolo decorations. MIDI output, however, can be made by MuseScore.

### 8.6.2 Percussion Maps

As we know, the `perc` clef type is used for percussion notes. To specify percussion instruments and note positions, `abc2xml.py` provides a new directive to implement *percussion maps*:

%%`percmap` ⟨*note*⟩ *[mapped_note]* ⟨*percussion*⟩ *[note shape]*

where:

- ⟨*note*⟩ is the ABC name of a note (e.g. `_B,`) that will be mapped to a percussion instrument;

- *[mapped_note]* is the Abc name of the mapped note (e.g. A). Any note equal to ⟨*note*⟩ will be printed as *[mapped_note]*. Use * to print ⟨*note*⟩ on its normal position;

- ⟨*percussion*⟩ is the MIDI code of a percussion instrument, or its standard name (please see Section A.7.2, p. 163). Spaces in the instrument name must be replaced by a hyphen `-`. Instrument names may be shortened; for example, you can write `acou-ba-dr` instead of `acoustic bass drum`;

- *[note shape]* is a string that specifies the shape of the note head: `slash`, `triangle`, `diamond`, `square`, `cross`, `x`, `circle-x`, `normal`, `cluster`, `inverted triangle`, `arrow down`, `arrow up`, `slashed`, `back slashed`, `do`, `re`, `mi`, `fa`, `so`, `la`, `ti`, and `none`.

Percussion maps can be turned off specifying any clef different than `perc`, or with the new option `map=off`.

The following example, kindly provided by Willem Vree, can be typeset and also played correctly in MuseScore:

```
X: 1
L: 1/8
V: 1 clef=treble
V: 2 clef=perc
K: C
% abc2xml.py percussion maps
%%percmap C D 52 triangle
%%percmap A * 60 x
%%percmap c * 47 diamond
%
V:1 % treble clef
cA[CA]A AA[CA]A | cA[CA]A AA[CA]A |
V:2 % percussion
%%MIDI channel 10
cA[CA]A AA[CA]A | cA[CA]A AA[CA]A |
```
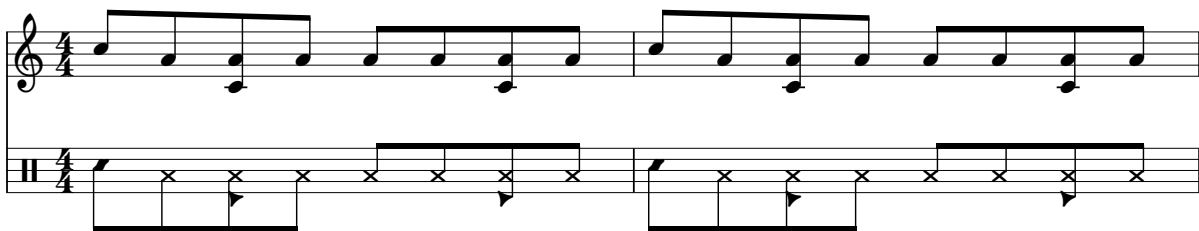


Let's now examine a more complete example that implements all percussion instruments described in the "Percussion notation" Wikipedia page. To play the file correctly with abc-2midi, we also specify the MIDI channel 10 (percussion instruments):

```
%%percmap D  pedal-hi-hat  x
%%percmap E  bass-drum-1
%%percmap F  acou-ba-dr     % acoustic bass drum (abbreviation)
%%percmap G  low-floor-tom
%%percmap A  high-floor-tom
%%percmap B  low-tom
%%percmap ^B tambourine     triangle
```

```
%%percmap c  acou-snare            % acoustic snare
%%percmap _c elec-snare            % electric snare
%%percmap ^c low-wo-blo    triangle % low wood block
%%percmap =c side-stick
%%percmap d  low-mi-tom            % low-mid tom
%%percmap ^d hi-wo-blo     triangle % high wood block
%%percmap e  hi-mi-tom             % high-mid tom
%%percmap ^e cowbell       triangle
%%percmap f  high-tom
%%percmap ^f ride-cymbal-1 x
%%percmap g  closed-hi-hat x
%%percmap ^g open-hi-hat   diamond
%%percmap a  crash-cym-1   x
%%percmap ^a open-tri      triangle

X: 1
M: none
T: Sibelius Drum Legend
T: https://en.wikipedia.org/wiki/Percussion_notation
L: 1/4
V: 1 clef=perc
K: C
%
[V:1 clef=perc]
%%MIDI channel 10   % activate abc2midi map
%
   D        E        F        G        A             |
w: pedal    bass     acoustic   lowfloor highfloor
w: hihat    drum~1   bass~drum tomtom    tomtom
%
   B        ^B       c        _c       ^c            |
w: low      tambourine acoustic   electric   low
w: tomtom * snare    snare     wood~block
%
   =c       d        ^d       e        ^e            |
w: side     lowmid   high      highmid  cowbell
w: stick    tomtom   wood~block tomtom  *
%
   f        ^f       g        ^g       a        ^a |
w: high     ride     closed    open     crash    open
w: tomtom cymbal~1   hihat     hihat    cymbal~1 triangle
```

# Sibelius Drum Legend

https://en.wikipedia.org/wiki/Percussion_notation



pedal     bass      acoustic   low-floor  high-floor      low      tambourine  acoustic  electric  low
hi-hat    drum 1    bass drum  tom-tom    tom-tom         tom-tom              snare     snare     wood block

side      low-mid   high       high-mid   cowbell         high     ride      closed    open      crash     open
stick     tom-tom   wood block tom-tom                    tom-tom  cymbal 1  hi-hat    hi-hat    cymbal 1  triangle

The above examples can only be typeset by MuseScore. Fortunately, we learnt in Section 7.2 (p. 109) how to implement percussion notes for abcm2ps and abc2svg using note mapping. Let's take our first example and add note mapping, to make it compatible with all programs:

```
%%beginsvg
<defs>
% x-shaped head
<path id="x_head"
  d="m -3 -3 l 6 6 m 0 -6 l -6 6"
  class="stroke" style="stroke-width:1.2"/>

% triangle head
<path id="triangle"
  d="m -3.5 3.5 l 3.5 -7 l 3.5 7 l -7 0"
  class="fill" />

% diamond head
<path id="diamond"
  d="m 0 -3.5 l -3.5 3.5 l 3.5 3.5 l 3.5 -3.5 l -3.5 -3.5"
  class="fill" />
</defs>
%%endsvg

X: 1
L: 1/8
V: 1 clef=treble
V: 2 clef=perc
K: C
% abc2xml.py percussion maps
I:percmap C D 52 triangle
I:percmap A * 60 x
I:percmap c * 47 diamond
%
% abcm2ps / abc2svg percussion maps
%
%%map drummap C print=D heads=triangle
%%map drummap A heads=x_head
%%map drummap c heads=diamond
%
V:1 % treble clef
cA[CA]A AA[CA]A | cA[CA]A AA[CA]A |
%
V:2 % perc clef
%%MIDI channel 10 % percussion map for abcMIDI
I:voicemap drummap % percussion map for abcm2ps/abc2svg
cA[CA]A AA[CA]A | cA[CA]A AA[CA]A |
```

### 8.6.3  Tablatures

A simple extension to the K: or V: fields prints the notes as tablatures for guitar or other stringed instruments. Alternate tunings and finger positions can also be specified.

The syntax is:

K: or V: ⟨*clef=tab*⟩ *[strings=]*⟨*tuning*⟩ *[capo=*⟨*n*⟩*] [nostems]*

Parameters are:

- *[strings=]*⟨*tuning*⟩ specifies an alternate strings tuning. ⟨*tuning*⟩ is a text string that defaults to `E2,A2,D3,G3,B3,E4`, i.e. the default guitar tuning. These are notes in International Pitch Notation, not Abc notes. Spaces are not allowed between the notes.

- *[capo=]*⟨*n*⟩ indicates a capo on fret ⟨*n*⟩; tuning is changed accordingly.

- *[nostems]* prints tab notes with no stems.

Alternate tunings are notated specifying the right notes in the `strings=` option. Guitar DADGAD tuning, for example, is defined as `D2,A2,D3,G3,A3,D4`; standard Irish bouzouki as `G2,D2,A3,D3`; bass guitar as `E1,A1,D2,G2`.

Guitar music is normally notated using the `clef=treble-8` clef specifier. To obtain sensible finger positions, `octave=-1` must also be specified in tablature voice definitions.

Optimal string and finger positions are calculated by the scorewriter, but strings can be forced using decorations `!0!` … `!6!`; finger positions on the specified strings will be calculated accordingly. In the following example, we print a tablature for the C chord using default strings in the first measure, then specifying alternate string positions:

```
X: 1
L: 1/4
M: 2/4
K: C
%
V:1 clef=tab octave=-1
%   standard positions
[CEG] [CEG] [CEG]2 |
%   alternate fingering
[CEG] [!6!C!5!E!4!G] [CE!4!G]2 |
```



Note that adding `!4!` to `G` in the second measure suffices to alter the other strings; there is no other way to get a C chord.

The following example is taken from MuseScore's documentation, with minor modifications suggested by Willem Vree. We want to print two voices in staff notation and as a tablature; we then write two almost identical pairs of voices. We add `clef=treble-8` to the K: field, and add tablature options to the V: definition of voices 3 and 4:

```
X: 1
M: 4/4
L: 1/4
%%score (1 2) (3 4)
K: C clef=treble-8
```

```
% standard notation
V:1
g E/G/ G,/e/-e | d E/G/ G,/c/-c | B2   g  f- | f  D/G/ G,/B/- B |
V:2
C E   G,   E | C E   G,    E | G, D G, D  | G, D   G,     D |
% tablature voices
V:3 clef=tab nostems octave=-1
g E/G/ G,/e/-e | d E/G/ G,/c/-c | B2   g  f- | f  D/G/ G,/B/- B |
V:4 clef=tab nostems octave=-1
C x   x   E | C x   x    E | G, D G, D  | G, x   x     D |
```



As you can see, a few notes were replaced by invisible rests in voice 4 to avoid a minor issue with MuseScore's handling of unisons in tablatures. In fact, when MuseScore finds a unison like E/ and E in voices 1 and 2, it puts the notes on two different strings: this may lead to unplayable positions. Other programs handle unisons correctly.

The last example, kindly provided by Willem Vree, shows how to implement a tablature for banjo. We specify 5 notes in `strings=` and add `%%scale` to make the tablature fit in one line. The resulting tablature was edited in MuseScore to reduce the title size.

```
%%scale 0.65
X:1
T:Blue Moon Of Kentucky (banjo)
C:http://moinejf.free.fr/abcm2ps-doc/banjo.xhtml
M:4/4
L:1/4
V:1 treble-8
V:2 tab strings=G4,D3,G3,B3,D4 octave=-1
K:C
%
[V:1] z[Bg]    [cg]   [^cg]    |: ^c/[dd]/ g/d/       d/g/        d/d/ |
[V:2] z[B!5!g][c!5!g][^c!5!g] |: ^c/[dd]/ !5!g/!2!d/ d/!5!g/ !2!d/d/ |
%
[V:1]  B/d/ B/d/ A/d/ G/d/ | E/c/ G/d/    g/G/ d/g/     |
[V:2]  B/d/ B/d/ A/d/ G/d/ | E/c/ G/d/ !5!g/G/ d/!5!g/ |
%
[V:1] G/e/    g/G/ e/G/ E/e/ | D/d/    g/G/ d/g/    B/d/ |
[V:2] G/e/ !5!g/G/ e/G/ E/e/ | D/d/ !5!g/G/ d/!5!g/ B/d/ |
```

## Blue Moon Of Kentucky (banjo)

http://moinejf.free.fr/abcm2ps-doc/banjo.xhtml



### 8.6.4 Tablatures for abc2svg

abc2xml.py creates tablatures by adding commands in the MusicXML output. Applications like MuseScore then read these commands, perform tablature allocations, and print the score accordingly.

abc2xml.py and xml2abc.py provide command line switches that make tablatures available to abc2svg, without the need of MusicXML programs. This is a two-step procedure:

1. we first convert our original ABC source to MusicXML specifying the new -f switch;

2. we then convert the MusicXML file back to ABC, specifying the new -t switch.

The first switch, -f, forces abc2xml.py to perform the allocation of notes to string and fret positions. This step is usually performed by the MusicXML application. The second switch, -t, instructs xml2abc.py to write note maps for abc2svg that implement the tablatures we need. The original ABC file and the new one should be named differently.

This source, bassdrum.abc, is taken from Willem Vree's excellent tutorial at https://wim. vree.org/js2/tabDrumDoc.html:

```
X:1
L:1/8
M:4/4
%%score (1 2) 3
K:Cm
%
V:1 perc nm="Drum"
I:percmap ^g closed-hi-hat x
%
V:2 perc
I:percmap c acoustic-snare
I:percmap _c side-stick triangle
I:percmap E acoustic-bass-drum square
%
V:3 tab octave=-1 strings=E1,A1,D2,G2 nm="Bass"
%%MIDI program 36
%
[V:1] ^g^g ^g^g ^g^g ^g^g | ^g^g ^g^g ^g^g ^g^g |
[V:2] EE    c x  EE   c x | EE    _c x  EE _ c x |
[V:3] C,3     E,3    G,A,| G,3   F,3 !2!G,D,   |
```

The required commands are:

```
~$ abc2xml.py -f bassdrum.abc > bassdrum.xml
-- decoded from utf-8
-- skipped header: (field X,1)
-- done in 0.03 secs
~$ xml2abc.py -t bassdrum.xml > bassdrum2.abc
bassdrum.abc written with 3 voices
~$ _
```

Please note that we chose a different name for the final Abc file.

Loading the resulting source bassdrum2.abc in abc2svg, we obtain this score:
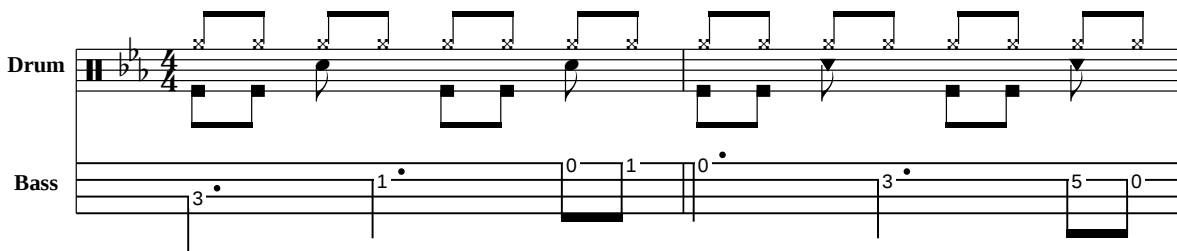


whose Abc source is much longer and more complex than the original bassdrum.abc, as it contains note maps. Try the conversion yourself: the original file is 20 lines long, the final file is 71 lines long.

To sum up, the -f and -t switches let us choose which application to use for the final rendering of the tablature.

## 8.7   Abc/MusicXML Conversion: Limitations

Converting simple tunes to MusicXML is usually pointless: abcm2ps, abc2svg, and abcMIDI are excellent programs, and can easily manage the music in nearly all cases. Besides, Abc can manage tune collections, while MusicXML cannot. Conversion can be useful in the case of complex scores, such as choral or piano music; or when you want to use tablatures.

In general, conversion of melody and harmony will work, but several elements will be ignored and/or skipped. Among these: nested tuplets; format files; many formatting parameters; several fields; a few note decorations; backslash sequences (use UTF-8 characters instead); PostScript and SVG customisations and extensions; and maybe more.

In spite of these limitations, most Abc files can be turned to MusicXML and imported in MuseScore or LilyPond with very little or no manual customisation.

♫ ♫ ♫ ♫ ♫

# Chapter 9

# Advanced Possibilities

THIS chapter is written for expert users who can work comfortably with the command line. GNU/Linux and macOS, by default, provide a terminal that runs the Bash shell; this is what we need to run the programs in the following sections. Windows too provides a shell environment, but it works quite differently.

For the sake of simplicity and compatibility, I recommend that Windows users install MSYS2, http://www.msys2.org/, which provides a Bash shell and a Unix-like environment that works almost exactly like the original. Installing a virtual machine running GNU/Linux is another possibility, but is probably overkill.

All examples in the following sections were tested in GNU/Linux only, but they can easily be adapted to work in Windows (MSYS2) and Mac OS X.

## 9.1   Cropping PostScript and PDF Files

> ✍ **Note**
>
> To use the procedures outlined in this section, Windows users must install Ghostscript in three steps to circumvent a bug:
>
> 1. install the latest release of Ghostscript as a native Windows package in `C:\gs`;
>
> 2. run the MSYS2 terminal and install Ghostscript as a native MSYS2 package:
>
>    ```
>    $ pacman -S mingw-w64-x86_64-ghostscript
>    ```
>
> 3. run this command to replace MSYS2's bugged `gs.exe` with the latest version:
>
>    ```
>    $ cp /c/gs/gs9.25/bin/gswin64c.exe /mingw64/bin/gs.exe
>    ```

By default, `abcm2ps` makes PostScript files that consist of whole pages, even when the music only spans one or a few lines. The same holds for MuseScore, which creates whole-page PDF files.

If we need to crop the PostScript or PDF file to its actual contents, we can use the command-line programs provided by Ghostscript. This procedure only works for single-page PostScript files:

```
$ pdf2ps scale1.pdf    # scale1.ps is created - skip if unneeded
```

```
$ ps2epsi scale1.ps     # scale1.epsi is created
$ epstopdf scale1.epsi # scale1.pdf is overwritten or created
$ _
```

Some PostScript files may cause ps2epsi to fail. In this case, we will need the Perl script ps2eps, https://ctan.org/pkg/ps2eps.

Let's use ps2eps instead of ps2epsi:

```
$ pdf2ps scale1.pdf     # scale1.ps is created - skip if unneeded
$ ps2eps scale1.ps
Input files: scale1.ps
Processing: scale1.ps
Rendering with existing %%BoundingBox: 0 0 595 842
Calculating Bounding Box...ready. %%BoundingBox: 27 596 567 732
Creating output file scale1.eps ... ready.
$ epstopdf scale1.eps  # scale1.pdf is overwritten or created
$ _
```

This is the procedure that I used in this manual to include the music examples.

To crop a multi-page PDF file, the recommended program is PDFCrop, http://pdfcrop. sourceforge.net. This program too requires Perl.

```
$ pdfcrop tunes.pdf
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 49 pages written on `tunes-crop.pdf'.
$ _
```

Each page in the PDF file will be cropped to its contents.

## 9.2   Using abc2svg in the Command Line

We normally use abc2svg in its browser-based editor, but in this environment abc2svg cannot handle large Abc files; say, tune collections with more than 300 tunes. It may become too slow or unresponsive.

Luckily, abc2svg can also be used directly from the command line; in this case, it can manage much larger Abc files. Being a Javascript program, we will need the Node.js® runtime, which is usually available as a native GNU/Linux package. It's also available as a native package in MSYS2.

Find out how to install node and the Javascript package manager npm on your system. To install abc2svg, run this command (as root user in GNU/Linux and macOS):

```
# npm -g install abc2svg
```

From now on, we can turn our music to PDF using abc2svg, which translates the Abc source into an xhtml file:

```
$ abc2svg file.abc > file.xhtml
$ _
```

The `file.xhtml` file can then be loaded into any web browser and printed to PDF. As stated earlier, Chrome or Chromium are the recommended browsers.

We can run the whole procedure non-interactively, entirely in the command line. Unfortunately, this only works in GNU/Linux and Mac OS X:

```
$ abc2svg tunes.abc > tunes.xhtml
$ chromium-browser --headless --print-to-pdf=tunes.pdf tunes.xhtml
... many log messages ...
... Written to file tunes.pdf.
$ _
```

However, in this case Chrome will add headers and footers to every page. To remove them and obtain a clean PDF file, run the command:

```
$ pdfcrop --margins "0 -9 0 -9" --clip tunes.pdf
PDFCROP 1.38, 2012/11/02 - Copyright (c) 2002-2012 by Heiko Oberdiek.
==> 79 pages written on `tunes-crop.pdf'.
$ _
```

In the resulting file `tunes-crop.pdf`, page geometry is slightly different from that of the original file. Another method that removes headers and footers but respects the page geometry employs LaTeX, as explained in Section 9.5 (p. 134).

Another program that translates `xhtml` to PDF is `wkhtmltopdf`, https://wkhtmltopdf. org/. Unfortunately, its PDF output may be slightly bugged.

## 9.3  Converting with `abc2odt`

The `abc2svg` package contains another useful program, `abc2odt`, which converts Abc files to the ODT format. This is the native file format in LibreOffice Writer.

To use `abc2odt`, the `jszip` JavaScript package must be installed beforehand:

```
# npm -g install jszip
```

To convert an Abc file to ODT:

```
$ abc2odt file.abc -o file.odt
ODT generation started
file.odt created
~$ _
```

`file.odt` can be opened with LibreOffice and in general with any application that supports this file format.

LibreOffice can also be used to turn an Abc file to PDF, interactively using the "Export to PDF" option or in the command line. The latter option is better for large Abc files:

```
$ abc2odt file.abc -o file.odt
ODT generation started
file.odt created
~$ loffice --headless --convert-to pdf file.odt
convert /home/guido/file.odt -> /home/guido/file.pdf \
  using filter : writer_pdf_Export
~$ _
```

The unoconv Python script, https://github.com/dagwieers/unoconv, can also be used to convert ODT files to PDF:

```
$ abc2odt file.abc -o file.odt
ODT generation started
file.odt created
$ unoconv -f pdf file.odt
~$ _
```

Making a PDF file using LibreOffice or unoconv is much slower than using abc2svg and Chrome, but headers and footers work correctly as in abcm2ps.

## 9.4   Inserting Music in LaTeX

This guide is written in LaTeX, which you may want to use instead of a word processor. Inserting ABC music in LaTeX documents is very simple; I assume you will turn your LaTeX sources to PDF using pdflatex.

Turn your music excerpts to PDF and, if needed, crop them as described in the previous section. 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.pdf}
\end{document}
```

You can also insert whole pages of music using the pdfpages package. The simplest way is this LaTeX source:

```
\documentclass[oneside]{article}
\usepackage{pdfpages}
\usepackage[a4paper,margin=1.5cm]{geometry}

\begin{document}

% pages=- means "all pages"
\includepdf[pages=-,pagecommand={},width=\textwidth]{music.pdf}

\end{document}
```

## 9.5   Typesetting with abc2svg and LaTeX

Another way to typeset music to PDF using abc2svg employs LaTeX for the final stage.

Let's suppose we have tunes.abc that we want to convert to PDF, but respecting the page geometry. We first convert the file to obtain a first PDF draft:

```
$ abc2svg tunes.abc > tunes.xhtml
$ chromium-browser --headless --print-to-pdf=tunes1.pdf tunes.xhtml
$ ...
$ ... Written to file tunes1.pdf.
$ _
```

Now we have `tunes1.pdf` that is "ruined" by headers and footers added by Chrome. To remove them we can use LaTeX.

Save this LaTeX source as `tunes.tex`:

```
\documentclass{article}
\usepackage{pdfpages}
\pagestyle{empty} % no page numbers

\def\tunes{tunes1.pdf} % PDF file to trim

\begin{document}

\includepdf%
[pages=-,pagecommand={},width=\paperwidth,%
trim={0 0.9cm 0 0.9cm},clip]{\tunes}

\end{document}
```

This source uses `pdfpages` to include pages from a PDF file we specify, and it also trims every page to remove headers and footers. Run this command to typeset `tunes.tex`:

```
$ pdflatex tunes.tex
...lots of messages...
Output written on tunes.pdf (79 pages, 966672 bytes).
Transcript written on tunes.log.
$ _
```

`pdflatex` will typeset `tunes.pdf` without header and footer. We can also use other LaTeX packages, such as `geometry`, to further customise the output.

## 9.6   Using `abc.sty`

Prof. Enrico Gregorio of University of Verona (Italy) has written the `abc.sty` package that enables the inclusion of ABC code in LaTeX documents. The relevant archive can be freely obtained from https://www.ctan.org/pkg/abc. `abc.sty` is also included in `texlive-music`, provided by several GNU/Linux distributions.

`abc.sty` provides the following facilities:

- the abc environment

- the `\abcinput` command

- the `\abcwidth` parameter.

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

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

\begin{document}

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

This is a short piece.

\medskip

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

\medskip

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

\medskip

\abcinput{tune}

\end{document}
```

Save it as `abc.tex` and compile it with this command:

```
$ pdflatex -shell-escape abc.tex
...lots of messages...
abc.pdf (1 page, 39540 bytes).
Transcript written on abc.log.
```

## 9.7   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 directive is best done with yet another command-line program, bmeps. It's 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 we omit -c, the resulting EPS file will be black and white.

## 9.8  Converting MIDI to Abc

Another way of getting music in Abc format is the conversion from MIDI files, which can be accomplished using two programs.

midi2abc, part of abcMIDI, is a command line program that converts a MIDI file to the corresponding Abc source:

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

By default, the resulting Abc file will contain only one measure per line.

Much better MIDI conversion can often be obtained using MuseScore, which can open MIDI files and export them in MusicXML format. We then use xml2abc.py to convert the MusicXML files to Abc.

> ✐ **Note**
>
> The quality of input MIDI files heavily affects the quality of the converted Abc files. Results may vary from near perfection to unreadable junk.

## 9.9  The abcpp Preprocessor

A *preprocessor* is a program that modifies a text file, in accordance with the directives contained in the file. abcpp is a preprocessor expressly designed for Abc files. It allows to

- exclude or include parts of a piece following specified conditions;

- define *macros*, i.e. symbols and sequences of customised directives;

- rename directives, symbols, and notes;

- include parts of other files.

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

### 9.9.1  Basic Usage

Let's 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
#ifndef ABCMIDI
|::cdef gabc'::|
#else
|:|1-3 cdef gabc':|
#endif
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. Similarly, the fourth directive means: "if the symbol ABCMIDI is not defined, then...". The following two measures are written for abcm2ps; after the #else directive, two measures are written in an abc2midi-compatible way.

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
|:|1-3 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's 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 we write the tune, we will use !x! to denote invisible rests. When we convert the source for abcm2ps or other programs, the !x! symbol will be turned into x or z in accordance with the presence of the symbol OLD.

### 9.9.2   Redefining Symbols

In Section 7.3 (p. 111), we renamed some symbols in a way that makes them unrecognisable to abc2midi. A better approach is using abcpp to define shorter versions of long symbols. We include these lines at the top of our source:

```
#define !fmt!  !fermata!
#define !lmd!  !lowermordent!
#define !umd!  !uppermordent!
#define !ub!   !upbow!
#define !db!   !downbow!
#define !,!    !breath!
#define !lphr! !longphrase!
#define !mphr! !mediumphrase!
#define !sphr! !shortphrase!
#define !ifmt! !invertedfermata!
#define !trn!  !invertedturn!
#define !itrn! !invertedturnx!
#define !mrd!  !mordent!
#define !arp!  !arpeggio!
#define !inv!  !invisible!
```

then write the tune using the "new" symbols !fmt!, !lmd!, and so on. Before converting the source, we preprocess it with the command:

```
$ abcpp mytune.abp mytune.abc
$ _
```

## 9.10   Conversions: 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 ⟨*x*⟩  outputs only voice ⟨*x*⟩ of a polyphonic file.

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

As usual, here is an example. Let's modify this scale (file cde.abc):

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

running abc2abc with this command line:

$ **abc2abc cde.abc -n 2 -t 2**
$ _

This command reformats the source to get two measures per line and transpose the music up two half-tones. 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.

♫ ♫ ♫ ♫ ♫

# Chapter 10

# The End

## 10.1   What Now?

Congratulations: you have just begun to master Abc, which is a good thing for any musician.

You have learnt to use a cleverly designed music notation format that has been implemented in several smart pieces of software, written by competent and passionate musicians/programmers. Now you can edit your own music, or use the vast amount of pieces in Abc format that you can freely download. What about giving something back in exchange?

If you are a programmer, it would be great if you contributed to the development of existing Abc software. In particular, `abcm2ps` needs a new maintainer.

As a musician, please consider contributing your music to existing Abc collections; or create your own, and make it public.

In any case: use Abc, spread the word, enjoy music!

## 10.2   Final Comments

I started with Abc several years ago, and it is always been my tool of choice whenever I needed to make a score. 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 to release this manual as free documentation.

A big "thank you!" to the author of `abcm2ps` and `abc2svg`, Jean-François Moine, for writing these fantastic programs; 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 Willem Vree for the Abc/MusicXML translators; to Chris Walshaw for creating Abc.

The following people provided suggestions and code samples that helped me a lot: D. Glenn Arthur, Gianni Cunich, Reinier Maliepaard, Sandro Pasqualetto, Norman Schmidt, and many others at the Abc users mailing list.

Last but not least, thanks to all the people who contribute to Abc publishing tune collections on the web!

## 10.3   In Loving Memory of Annarosa Del Piero, 1930–2000

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

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

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

♫ ♫ ♫ ♫ ♫

# Appendix A

# Bits & Pieces

## A.1   Web Links

- The original Abc page:
  http://abcnotation.com

- The Abc 2 home page:
  http://abcplus.sourceforge.net

- The abcm2ps and abc2svg home page and paramater list:
  http://moinejf.free.fr
  http://moinejf.free.fr/js/index.html
  http://moinejf.free.fr/abcm2ps-doc/index.html

- The abcMIDI home page and user guide:
  http://ifdo.ca/~seymour/runabc/top.html
  http://ifdo.ca/~seymour/runabc/abcguide/abc2midi_guide.html

- The UTF-8/ISO-8859-1 converter:
  https://www.charset.org/utf8-to-latin-converter

- Hudson Lacerda's Abc page, including advanced customisation and many new Post-Script routines:
  http://hudsonlacerda.webs.com/

- The MusicXML home page:
  http://www.musicxml.com

- A list of sites that host music in MusicXML:
  http://www.musicxml.com/music-in-musicxml

- Willem Vree's abc2xml.py and xml2abc.py:
  https://wim.vree.org/svgParse/abc2xml.html
  https://wim.vree.org/svgParse/xml2abc.html

- MuseScore is a multiplatform, free and open source scorewriter:
  https://musescore.org

- GNU LilyPond is a multiplatform, free and open source text-based scorewriter:
  http://lilypond.org/web/

- The abc package for LaTeX:
  http://www.ctan.org/tex-archive/macros/latex/contrib/abc/

- The Abc Users mailing list:
  http://tech.groups.yahoo.com/group/abcusers/

## 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` |
| 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 half-tone` |
| I: | body | Meta-comment. `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 ABC Cheat Sheet

Note pitch, bass clef



**A,,,, B,,,, C,,, D,,, E,,, F,,, G,,, A,, B,, C, D, E, F, G, A, B, C**

Note pitch, treble clef



**C D E F G A B c d e f g a b c' e' f' g' a' b' c'' d'' e''**

Note and rest length



**A16 A8 A4 A2 A A/2 A/ A/A/ A// A/// A////**



**Z4 z16 z8 z4 z2 z z/ z// z/// z////**
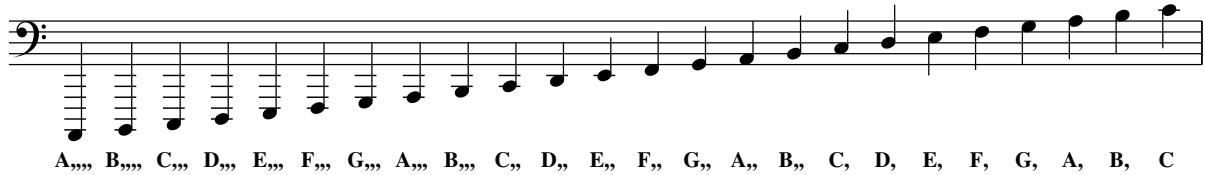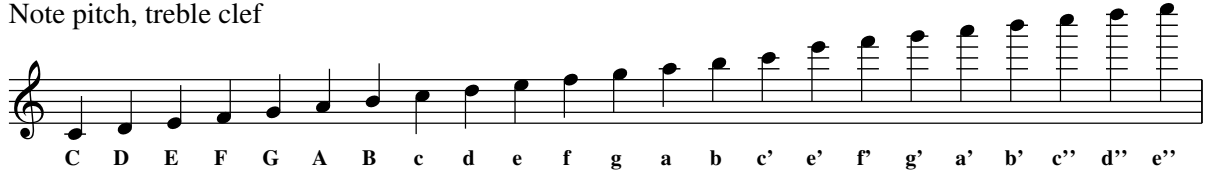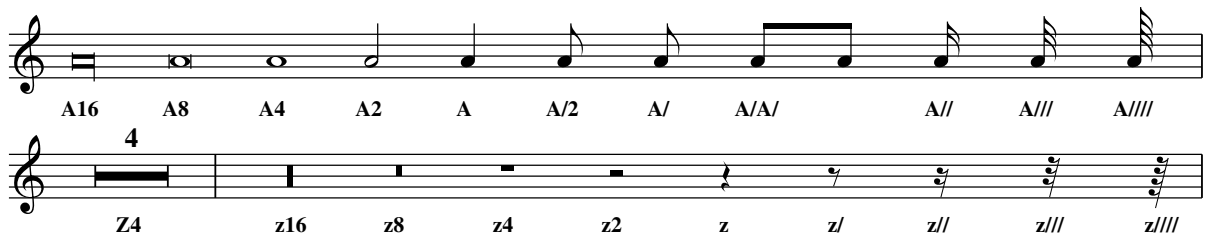
Accidentals     Dotted notes     Ties, slurs, staccato



**A ^A ^^A _A __A =A   A3/2 A>A A>>A   A - A (CEGc) .(CEGc)**

Chords     Tuplets     Grace notes



**[CEGc] [CEGc]2 [CEGc]>[CEGc]   (3C/E/G/   (3:2:2G2c   {B}A {AB}A**

Clefs



**K:clef=none K:treble K:treble-8 K:treble+8 K:bass K:bass3 K:alto4 K:alto K:alto2 K:alto1**

Bars, repeats, endings



**| || [| |] [: :] :|: ::   |1   :|2   :|["Last"   |]**
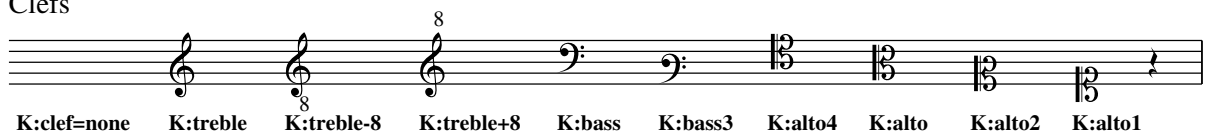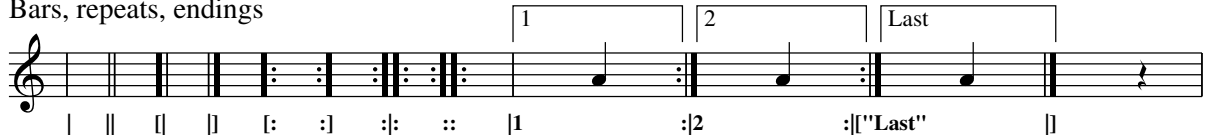
Accompaniment chords     Annotations
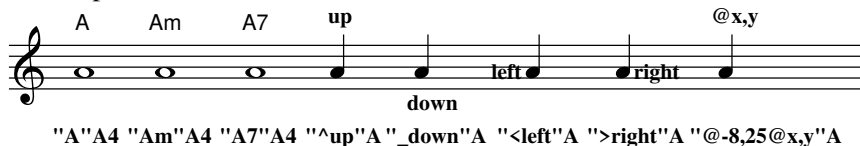


**"A"A4 "Am"A4 "A7"A4 "^up"A "_down"A "<left"A ">right"A "@-8,25@x,y"A**

# A.4 Formatting Directives

Directive parameters will be specified as follows:

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

These directives are also listed and explained at the `abcm2ps` parameters page.
Directives only available in `abcm2ps`:

```
%%abc2pscompat
%%alignbars
%%autoclef
%%botmargin
%%clip
%%dateformat
%%decoration
%%EPS
%%footerfont
%%footer
%%format
%%glyph
%%gutter
%%header
%%headerfont
%%landscape
%%leftmargin
%%micronewps
%%oneperpage
%%pageheight
%%pagewidth
%%pango
%%pdfmark
%%rightmargin
%%setdefl
%%splittune
%%staffwidth
%%tablature
%%topmargin
```

Directives only available in `abc2svg`:

```
%%beginml
%%chordnames
%%grid
%%grid2
%%ottava
%%fullsvg
%%printmargin
```

```
%%singleline
%%sth
```

### A.4.1  Page Format

These directives set the page geometry.

`%%botmargin` ⟨***length***⟩  sets the page bottom margin to ⟨*length*⟩. Default: 1 cm; scope: page; not available in `abc2svg`.

`%%footer` ⟨***text***⟩**:** sets the text to be printed as footer on each page. Default: none; scope: page; not available in `abc2svg`.

`%%header` ⟨***text***⟩**:** sets the text to be printed as header on each page. Default: none; scope: page; not available in `abc2svg`.

`%%indent` ⟨***length***⟩**:** sets the indentation for the first line or system to ⟨*length*⟩. Default: 0; scope: tune.

`%%landscape` ⟨***logical***⟩**:** if 1, sets the page layout as landscape. Default: 0; scope: page; not available in `abc2svg`.

`%%leftmargin` ⟨***length***⟩**:** sets the page left margin to ⟨*length*⟩. Default: 1.8 cm; scope: page, restart.

`%%multicol` ⟨***command***⟩**:** defines columns. ⟨*command*⟩ may be `start`, `new`, and `end`. Default: none; scope: immediate, restart.

`%%pageheight` ⟨***length***⟩**:** sets the page height to ⟨*length*⟩. Default: 11 inches; scope: page; not available in `abc2svg`. For European A4 paper, the right value is `29.7cm`.

`%%pagewidth` ⟨***length***⟩**:** sets the page width to ⟨*length*⟩. Default: 8.5 inches; scope: page; not available in `abc2svg`. For European A4 paper, the right value is `21cm`.

`%%rightmargin` ⟨***length***⟩**:** sets the page right margin to ⟨*length*⟩. Default: 1.8 cm; scope: page, restart.

`%%staffwidth` ⟨***length***⟩**:** used as an alternative to the `%%pageheight` and `%%pagewidth` directives. Default: none; scope: generation.

`%%topmargin` ⟨***length***⟩**:** sets the page top margin to ⟨*length*⟩. Default: 1 cm; scope: page; not available in `abc2svg`.

### A.4.2  Text

These directives are used to write text lines within a tune and between tunes. Fonts and spacing are set with other directives 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 %%begintext directive 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
```

Scope: immediate, restart.

%%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. Default: 0; scope: generation.

### A.4.3   Fonts

These directives specify the character fonts used in various parts of a score. Note that the common True Type fonts used by the operating system *are not the same fonts* used by abcm-2ps. In fact, abcm2ps uses the PostScript fonts, provided for and managed by Ghostscript.

Standard fonts are shown in Appendix A.5 (p. 158). I remind you that indications for adding new fonts are given in Section 7.5 (p. 113).

%%annotationfont ⟨***string***⟩**:** font of annotations. Default: Helvetica 12, sans-serif 12; scope: immediate.

%%capofont ⟨***string***⟩**:** font of %%capo indications.

%%composerfont ⟨***string***⟩**:** font of C: field(s). Default: Times-Italic 14, serifItalic 14; scope: tune.

%%footerfont ⟨***string***⟩**:** font of %%footer lines. Default: Times-Roman 12; scope: page.

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

%%gchordfont ⟨***string***⟩ *[box]***:** accompaniment chords font. If the parameter box is present, a box is drawn around the accompaniment chord. Default: Helvetica 12, sans-serif 12; scope: immediate.

%%gridfont ⟨***string***⟩: font of chord grids. lines. Default: serif 16; scope: tune.

%%headerfont ⟨***string***⟩: font of %%header lines. Default: Times-Roman 12; scope: page.

%%historyfont ⟨***string***⟩: font of H: field. Default: Times-Roman 16, serif 16; scope: tune.

%%infofont ⟨***string***⟩: font of I: field. Default: Times-Italic 14, serifItalic 14; scope: tune.

%%measurefont ⟨***string***⟩ *[box]*: text font of measure numbers. If the parameter box is present, a box is drawn around the measure number. Default: Times-Italic 14, serifItalic 14; scope: generation.

%%musicfont ⟨***string***⟩: defines a music font that replaces PostScript and/or SVG glyphs.

%%partsfont ⟨***string***⟩: font of P: field. Default: Times-Roman 15, serif 15; scope: generation.

%%repeatfont ⟨***string***⟩: font of repeat numbers or text. Default: Times-Roman 13, serif 13; scope: generation.

%%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. Scope: generation.

%%subtitlefont ⟨***string***⟩: font of the second T: field. Default: Times-Roman 16, serif 16; scope: generation.

%%tempofont ⟨***string***⟩: font of Q: field. Default: Times-Bold 15, serifBold 15; scope: tune.

%%textfont ⟨***string***⟩: font in %%text lines. Default: Times-Roman 16, serif 16; scope: generation.

%%titlecaps ⟨***logical***⟩: if 1, writes the title in capital letters. Default: 0; scope: tune.

%%titlefont ⟨***string***⟩: font of the first T: field. Default: Times-Roman 20, serif 20; scope: tune.

%%titleformat ⟨***string***⟩: defines the format of the tune title. This format overrides %%titleleft, %%infoline, and %%composerspace. Scope: tune.

%%titleleft ⟨***logical***⟩: if 1, writes the title left-aligned instead of centered. Default: 0; scope: tune.

%%voicefont ⟨***string***⟩: font of voice names. Default: Times-Bold 13, serifBold 13; scope: tune.

%%vocalfont ⟨***string***⟩: font of the text in w: lines. Default: Times-Bold 13, serifBold 13; scope: generation.

%%wordsfont ⟨***string***⟩: font of the text in W: lines. Default: Times-Roman 16, serif 16; scope: tune.

### A.4.4   Spacing

These directives specify spacing between score elements.

`%%barsperstaff` ⟨***int***⟩**:** attempts to typeset the score with ⟨*int*⟩ bars on each line. Default: 0; scope: generation.

`%%breaklimit` ⟨***float***⟩**:** used together with `%%maxshrink`, it lets the user control where line breaks may occur. The parameter can range between 0.5 (line break occurs when the line is 50% full) and 1.0. Default: 0.7; scope: generation.

`%%break` ⟨***int***⟩**:** inserts a break after ⟨*int - 1*⟩ symbols. Default: none; scope: generation.

`%%breakoneoln` ⟨***logical***⟩**:** if 1, treats an end of line as if it were a space (i.e. it breaks note beams). Default: 1; scope: generation.

`%%composerspace` ⟨***length***⟩**:** sets the vertical space above the `C:` field to ⟨*length*⟩. Default: 0.2 cm; scope: tune.

`%%gracespace` ⟨***float***⟩ ⟨***float***⟩ ⟨***float***⟩**:** defines the space before, within and after grace notes. Default: 6.5, 8.0, 12.0; scope: generation.

`%%gutter` ⟨***length***⟩**:** specifies the amount of gutter, i.e. the space between facing pages. Default: 0; scope: page.

`%%infospace` ⟨***length***⟩**:** sets the vertical space above the `I:` field to ⟨*length*⟩. Default: 0; scope: tune.

`%%linebreak` ⟨***string***⟩**:** defines or undefines the line break separators. The string may be empty or contain the values <EOL>, $, !, and <none>. Default: <EOL>; scope: immediate.

`%%lineskipfac` ⟨***float***⟩**:** sets the factor for spacing between lines of text to ⟨*float*⟩. Default: 1.1; scope: generation.

`%%maxshrink` ⟨***float***⟩**:** sets how much to compress horizontally when staff breaks are chosen automatically. ⟨*float*⟩ must be between 0 (don't shrink) and 1.0 (full shrink). Default: 0.65; scope: generation.

`%%maxstaffsep` ⟨***length***⟩**:** sets the maximum vertical space between systems. Default: 2000 pt; scope: generation.

`%%maxsysstaffsep` ⟨***length***⟩**:** sets the maximum vertical space between staves in a system. Default: 2000 pt; scope: generation.

`%%musicspace` ⟨***length***⟩**:** sets the vertical space before the first staff to ⟨*length*⟩. Default: 0.2 cm; scope: tune.

`%%newpage`**:** sets a page break. Scope: immediate, restart.

`%%notespacingfactor` ⟨***float***⟩**:** sets the proportional spacing of notes. 1 makes all notes equally spaced. Default: 1.414 ($\sqrt{2}$); scope: generation.

%%pagescale ⟨***float***⟩**:** sets the music scale factor to ⟨*float*⟩. Default: 1.0; scope: generation, restart.

%%parskipfac ⟨***float***⟩**:** sets the factor for spacing between text paragraphs to ⟨*float*⟩. Default: 0.4; scope: generation.

%%partsspace ⟨***length***⟩**:** sets the vertical space above the P: field in the header to ⟨*length*⟩. Default: 0.3 cm; scope: generation.

%%scale ⟨***float***⟩**:** sets the music scale factor to ⟨*float*⟩, assuming a 72 PPI resolution; you should use %%pagescale instead. Default: 0.75; scope: generation, restart.

%%sep ⟨***length1***⟩ ⟨***length2***⟩ ⟨***length3***⟩**:** prints a centered line long ⟨*length3*⟩, with ⟨*length1*⟩ space above and ⟨*length2*⟩ space below. Default; none; scope: immediate, restart.

%%sep**:** prints a centered separator. Default; none; scope: immediate, restart.

%%slurheight ⟨***float***⟩**:** sets the slur height factor; lesser than 1 flattens the slur, greater than 1 expands it. Default; 1.0; scope: generation.

%%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. Default: none; scope: immediate, voice.

%%staffsep ⟨***length***⟩**:** sets the vertical space between different systems to ⟨*length*⟩. Default: 46 pt; scope: generation.

%%stretchlast ⟨***float***⟩**:** stretches the last staff of a tune when underfull by more than ⟨*float*⟩. Default; 0.25; scope: generation.

%%stretchstaff ⟨***logical***⟩**:** if 1, stretches underfull staves across the page. Default: 1; scope: generation.

%%subtitlespace ⟨***length***⟩**:** sets the vertical space above the second T: field to ⟨*length*⟩. Default: 0.1 cm; scope: tune.

%%sysstaffsep ⟨***length***⟩**:** sets the vertical space between staves in the same system to ⟨*length*⟩. Default: 36 pt; scope: generation.

%%textspace ⟨***length***⟩**:** sets the vertical space above H: fields at the end of a tune to ⟨*length*⟩. Default: 0.5 cm; scope: generation.

%%titlespace ⟨***length***⟩**:** sets the vertical space above the first T: field to ⟨*length*⟩. Default: 0.2 cm; scope: tune.

%%topspace ⟨***length***⟩**:** sets the vertical space above a tune to ⟨*length*⟩. Note that a tune may begin with %%text directives before the title. Default: 0.8 cm; scope: tune.

%%vocalspace ⟨***length***⟩**:** sets the vertical space above w: lines to ⟨*length*⟩. Default: 23 pt; scope: generation.

%%voicescale ⟨***length***⟩**:** sets the scale of a voice, or of all voices if the comand is included in the header. Default: 1 (0.5–1.5); scope: generation, voice.

`%%vskip` ⟨***h***⟩**:** adds ⟨*h*⟩ vertical space. ⟨*h*⟩ may be a negative value. Default: none; scope: immediate, restart.

`%%wordsspace` ⟨***length***⟩**:** sets the vertical space above `W:` lines at end of the tune to ⟨*length*⟩. Default: 0 pt, 5 pt; scope: tune.

### A.4.5   Other Directives

Miscellaneous directives are grouped in this section.

`%%abc` ⟨***string***⟩**:** defines some Abc code. Along with `%%abcm2ps`, this directive is used for conditional output generation. Default: none; scope: immediate.

`%%abc2pscompat` ⟨***logical***⟩ **:** if 1, reverts to the old metod of dealing with notes in bass and other clefs. Default: 0; scope: generation.

`%%abc-include` ⟨***string***⟩**:** includes the Abc file named ⟨*string*⟩. Default: none; scope: immediate.

`%%alignbars` ⟨***int***⟩**:** aligns the bars of the next ⟨*int*⟩ lines of music. It only works on single-voice tunes. Default: 0; scope: immediate, restart; not available in `abc2svg`.

`%%aligncomposer` ⟨***int***⟩**:** specifies where to print the `C:` field. A negative value means "on the left", 0 means "centre", and a positive value means "on the right". Default: 1; scope: tune.

`%%ambitus` ⟨***1***⟩**:** prints an ambitus before the clef. Scope: generation. Not available in `abcm2ps`.

`%%autoclef` ⟨***logical***⟩**:** if 1, the clef set by the `K:` is initialised to "auto"; if 0, the clef is set as "treble". Default: 1; scope: generation.

`%%beginml`–`%%endml`**:** starts/ends a sequence of XHTML commands to be included in SVG output. Scope: immediate. Not available in `abcm2ps`.

`%%beginps`–`%%endps`**:** starts/ends a sequence of PostScript commands; please see Section 7.1 (p. 107). Default: none; scope: immediate.

`%%beginsvg`–`%%endsvg`**:** starts/ends a sequence of SVG commands. Default: none; scope: immediate.

`%%bgcolor` ⟨***string***⟩**:** defines the background colour of SVG images. The parameter can be a colour name, like 'green', or an RGB colour value, like `#00ff00`. Default: none; scope: page.

`%%bstemdown` ⟨***logical***⟩**:** if 1, the stem of the note on the third staff line points downwards. Otherwise, it points upwards or downwards in accordance with the previous note. Default: 0; scope: immediate, voice.

`%%cancelkey` ⟨***logical***⟩**:** if 1, accidentals are cancelled using natural signs when the key signature changes. Default: 1; scope: generation.

%%capo ⟨***int***⟩: for accompaniment chords, specifies that a capo should be used on fret ⟨*n*⟩. Scope: generation.

%%clef ⟨***clef***⟩ *[line] [octave]*: inserts a clef change, in a manner similar to K:. ⟨*clef*⟩ may be treble (or G2), alto (C3), tenor (Cf), bass (F4), perc (P2), G, C, G, C, F, P, or none. The latter prints no clef on the staff. The name of a user-provided PostScript glyph may also be used. *[line]* specifies the staff line on which the clef sits. *[octave]* can be +8 or −8, to print "8" above or below the clef; ^8 or _8, to print "8" above or below the clef and perform octave transposition. Default: none; scope: immediate, voice.

%%clip ⟨***string***⟩: defines a subset of tune(s) to be printed. Default: none; scope: generation.

%%combinevoices ⟨***int***⟩: equivalent to %%voicecombine applied to all voices. Default: 0; scope: immediate, voice.

%%contbarnb ⟨***logical***⟩: if 1, the bar number of the second repeat(s) is reset to the number of the first repeat. If 0, bars are sequentially numbered. Default: 0; scope: generation.

%%custos ⟨***logical***⟩: if 1, a custos is added at the end of each music line. A custos is a symbol that indicates the pitch for the first note of the next line. This directive works with single voice tunes only. Default: 0; scope: generation.

%%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). Default: %b %e, %Y %H:%M; scope: page.

%%dblrepbar ⟨***string***⟩: defines how double repeat bars are drawn; e.g. :][: (default), or ::, or :|:. Scope: generation.

%%deco ⟨***string1***⟩ ⟨***int1***⟩ ⟨***string2***⟩ ⟨***int2***⟩ ⟨***int3***⟩ ⟨***int4***⟩ ⟨***string3***⟩: adds a new decoration. Default: none; scope: immediate.

%%decoerr ⟨***logical***⟩: if 1, an error message is issued when a decoration is undefined. Default: 1; scope: immediate.

%%decoration ⟨***char***⟩: defines the decoration separator, which can be ! (default) or +. Default: !; scope: immediate; not available in abc2svg.

%%diagram ⟨***int***⟩: if 1, prints guitar chords diagram above the notes. Scope: generation. Not available in abcm2ps.

%%dynalign ⟨***logical***⟩: if 1, horizontally aligns the dynamics marks. Default: 1; scope: generation.

%%EPS ⟨***string***⟩: includes an external EPS file in the score. Default: none; scope: immediate, restart; not available in abc2svg.

%%flatbeams ⟨***logical***⟩: if 1, forces flat (horizontal) note beams. Default: 0; scope: generation.

%%format ⟨***string***⟩: reads the format file specified as parameter. Default: none; scope: immediate.

%%gchordbox ⟨**logical**⟩: if 1, draws a box around accompaniment chords. Default: 0; scope: generation.

%%glyph ⟨**string1**⟩ ⟨**string1**⟩: sets the name of a Unicode glyph. The first parameter is a hexadecimal value, the second is the name of the associated glyph in the font files. Default: none; scope: generation.

%%graceslurs ⟨**logical**⟩: if 1, draws slurs on grace notes. Default: 1; scope: generation.

%%graceword ⟨**logical**⟩: if 1, aligns lyrics words under grace notes when they are present. Default: 0; scope: generation.

%%grid ⟨**int**⟩: prints a chord grid above (⟨*n*⟩ = 1) or below (⟨*n*⟩ = -1) the tune. Default: 0; scope: generation; not available in abcm2ps.

%%grid2 *[int]*: replaces a voice with a chord grid. When *[int]* is missing, the grid is disabled. Default: 0; scope: voice; not available in abcm2ps.

%%hyphencont ⟨**logical**⟩: if 1, and if lyrics under the staff end with a hyphen, puts a hyphen in the next line. Default: 1; scope: generation.

%%infoline ⟨**logical**⟩: if 1, displays R: and A: fields on the same line above the first music line. Default: 0; scope: tune.

%%infoname ⟨**string**⟩ ⟨**string**⟩: defines the name of information fields that are printed when %%writefields is also set to 1. Defaults: fieldR:, "Rhythm: "; B:, "Book: "; S:, "Source: "; D:, "Discography: "; N:, "Notes: "; Z:, "Transcription: "; H:, "History: "; scope: tune.

%%keywarn ⟨**logical**⟩: if 1, a cautionary key signature is printed when a key signature change occurs at the beginning of a music line. Default: 1; scope: generation.

%%linewarn ⟨**logical**⟩: if 1, outputs a warning message if there are too many or too few elements in a music line. Default: 1; scope: generation.

%%map ⟨**string**⟩: defines note mapping. Default: none; scope: immediate.

%%measurenb ⟨**int**⟩: if ⟨*int*⟩ is greater than 0, prints measure numbers every ⟨*int*⟩ bars; if it is 0, prints the measure number at the beginning of the staff. Default: -1; scope: generation.

%%measurebox ⟨**logical**⟩: if 1, draws a box around measure numbers. Default: 0; scope: generation.

%%micronewps ⟨**logical**⟩: if 1 uses the new PostScript functions to draw microtonal accidentals. Default: 0; scope: page; not available in abc2svg.

%%microscale ⟨**int**⟩: defines the denominator for microtone accidentals. Default: none; scope: immediate, voice.

%%oneperpage ⟨**logical**⟩: if 1, outputs one tune per page. Default: 0; scope: page.

%%pango ⟨**logical**⟩: if abcm2ps was compiled with Pango support, this directive enables or disables PostScript output with Pango. Default: 1; scope: generation.

%%partsbox ⟨***logical***⟩: if 1, draws a box around part names (P: fields.) Default: 0; scope: generation.

%%pdfmark ⟨***int***⟩: if the parameter is 1, inserts marks in the PostScript output for titles. Post-Script to PDF translators can use the marks to create a PDF tune index. If the parameter is greater than 1, marks are inserted for titles and subtitles. Default: 0; scope: page.

%%pos ⟨***type***⟩ ⟨***position***⟩: defines the direction and/or the position of some music elements. ⟨*type*⟩ can be dynamic, gchord, gstem, ornament, stem, vocal, and volume. ⟨*position*⟩ can be auto, above or up, below or down, hidden, and opposite. Default: auto; scope: immediate, voice.

%%postscript ⟨***string***⟩: lets the user add a new PostScript routine, or change an existing one. Default: none; scope: immediate; not available in abc2svg.

%%ps ⟨***string***⟩: same as %%postscript.

%%rbdstop ⟨***logical***⟩: if 1, repeat brackets stop on a double bar. Default: 1; scope: generation.

%%rbmax ⟨***int***⟩: sets the maximum number of bars over which a repeat bracket is drawn. Default: 4; scope: generation.

%%rbmin ⟨***int***⟩: sets the minimum number of bars over which a repeat bracket is drawn. Default: 2; scope: generation.

%%repbra ⟨***logical***⟩: if 0, prevents displaying repeat brackets for the current voice. Default: 1; scope: generation, voice.

%%repeat ⟨***int1***⟩ ⟨***int2***⟩: repeats a sequence of notes or bars. Default: none; scope: immediate.

%%score ⟨***string***⟩: defines how staves are to be printed in multivoice tunes. Default: none; scope: immediate.

%%select ⟨***string***⟩: select a subset of tunes to be printed. Default: none; scope: generation.

%%setbarnb ⟨***int***⟩: sets the number of the next measure, excluding the first. To set the number of the first measure, the directive must be written in the tune header. Default: none; scope: immediate.

%%setdefl ⟨***logical***⟩: if 1, outputs some indications about the note/chord and/or decorations for customization purposes. These indications are stored in the PostScript variable defl. Default: 0; scope: generation; not available in abc2svg.

%%shiftunison ⟨***logical***⟩: if 1, shifts note heads that belong to different voices that are in unison. It applies to dotted notes and notes that are shorter than a half note. Default: 0; scope: generation.

%%splittune ⟨***logical***⟩: if 1, splits tunes that do not fit in a single page. Default: 0; scope: generation.

%%squarebreve ⟨***logical***⟩: if 1, displays "brevis" notes in square format. Ending brevis notes are always squared. Default: 0; scope: generation.

`%%ss-pref` ⟨***string***⟩**:** changes the pseudo-comment prefix(es). Default: %; scope: immediate.

`%%straightflags` ⟨***logical***⟩**:** if 1, prints straight flags on note stems in bagpipe tunes. Default: 0; scope: generation.

`%%staff` ⟨***int***⟩**:** prints the next symbols of the current voice on the ⟨*int*⟩-th staff. Default: none; scope: immediate, voice.

`%%stafflines` ⟨***int***⟩**:** sets the number of staff lines of the current voice. Default: 5; scope: generation, voice.

`%%staffnonote` ⟨***int***⟩**:** avoids printing a staff, depending on the parameter. If it is 0, the staff is not printed if it contains only invisible rests and notes; if it is 1, the staff is not printed if it contains only rests and invisible notes; if it is 2, the staff is printed. Default: 1; scope: generation.

`%%staffscale` ⟨***float***⟩**:** sets the scale of the staff of the current voice. This directive corresponds to the `staffscale V:` parameter. Default: 1; scope: generation, voice.

`%%staves` ⟨***string***⟩**:** defines how staves are to be printed. This directive is deprecated; please use `%%score` instead. Default: none; scope: immediate.

`%%stemheight` ⟨***float***⟩**:** sets the stem height to ⟨*float*⟩. Default: 20.0; scope: generation.

`%%tablature`**:** defines a tablature. Default: none; scope: generation.

`%%timewarn` ⟨***logical***⟩**:** if 1, 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. Default: 0; scope: generation.

`%%titletrim` ⟨***logical***⟩**:** if 1, move the last word of a title to the head if it starts with a capital letter and it is preceded by a space and a comma. Default: 1; scope: tune.

`%%transpose` ⟨***int***⟩ *[char]*: transposes the music by ⟨*int*⟩ semitones. The optional parameter may be # or b, meaning that the new key signature will have sharps or flats. Default: 0; scope: immediate.

`%%tune` ⟨***string***⟩**:** select the tune(s) to which the following options will be applied. Default: none; scope: generation.

`%%tuplets` ⟨***int1***⟩ ⟨***int2***⟩ ⟨***int3***⟩ ⟨***int4***⟩**:** defines how tuplets are to be drawn. Default: 0, 0, 0, 0; scope: immediate.

`%%user` ⟨***string***⟩**:** this directive is the equivalent of the `U:` field, and can be used to redefine default decorations in format files. Default: none; scope: generation.

`%%voice` ⟨***string***⟩**:** select the voice(s) to which the following options will be applied. Default: none; scope: generation.

`%%voicecolor` ⟨***string***⟩**:** defines the colour of the symbols in the current voice. The parameter is an RGB colour, like `#aaff80`. Default: none, #000000; scope: immediate, voice.

`%%voicecombine` ⟨***int***⟩**:** defines how to combine notes and rests of several voices on the same staff. Default: 0; scope: immediate, voice.

`%%voicemap` ⟨***string***⟩**:** defines the note mapping of a voice. Default: none; scope: immediate, voice.

`%%writefields` ⟨***string***⟩ ⟨***logical***⟩**:** if 1, displays the information fields defined in ⟨*string*⟩. Default: `COPQTWw`, 1; scope: generation.

### A.4.6 Deprecated Directives

The following directives were available in previous releases of `abcm2ps` and should be replaced with their newer counterparts.

`%%abcm2ps`: replaced by `%%ss-pref`.

`%%barnumbers`: replaced by `%%measurenb`.

`%%comball`: replaced by `%%combinevoices 2`.

`%%continueall`: replaced by `%%linebreak none` along with `%%linewarn 0`.

`%%dynamic`: replaced by `%%pos dynamic`.

`%%exprabove`: replaced by `%%pos dynamic above` or `%%pos volume above`.

`%%exprbelow`: replaced by `%%pos dynamic below` or `%%pos volume below`.

`%%gchord`: replaced by `%%pos gchord`.

`%%gstemdir`: replaced by `%%pos gstem`.

`%%measurefirst`: replaced by `%%setbarnb`.

`%%musiconly`: replaced by `%%writefields w`.

`%%ornament`: replaced by `%%pos ornament`.

`%%printparts`: replaced by `%%writefields P`.

`%%printtempo`: replaced by `%%writefields Q`.

`%%stemdir`: replaced by `%%pos stem`.

`%%vocal`: replaced by `%%pos vocal`.

`%%vocalabove`: replaced by `%%pos vocal above`.

`%%volume`: replaced by `%%pos volume`.

`%%withxrefs`: replaced by `%%writefields X`.

`%%writehistory`: replaced by `%%writefields BDHNRSZ`.

## A.5    PostScript Fonts

There are 35 standard PostScript fonts. They are all listed below, with the exception of ZapfD-ingbats which is not supported by `abcm2ps`.

Bookman-Demi

*Bookman-DemiItalic*

Bookman-Light

*Bookman-LightItalic*

`Courier`

`Courier-Oblique`

**`Courier-Bold`**

**`Courier-BoldOblique`**

AvantGarde-Book

*AvantGarde-BookOblique*

AvantGarde-Demi

*AvantGarde-DemiOblique*

Helvetica

*Helvetica-Oblique*

**Helvetica-Bold**

***Helvetica-BoldOblique***

Helvetica-Narrow

*Helvetica-Narrow-Oblique*

**Helvetica-Narrow-Bold**

***Helvetica-Narrow-BoldOblique***

Palatino-Roman

*Palatino-Italic*

**Palatino-Bold**

***Palatino-BoldItalic***

NewCenturySchlbk-Roman

*NewCenturySchlbk-Italic*

**NewCenturySchlbk-Bold**

***NewCenturySchlbk-BoldItalic***

Times-Roman

*Times-Italic*

**Times-Bold**

***Times-BoldItalic***

Σψμβολ

*ZapfChancery-MediumItalic*

## A.6    `abcMIDI` Directives

Most of these directives are also listed and explained at the `abcMIDI` page, http://ifdo.ca/~seymour/runabc/abcguide/abc2midi_guide.html.

`%%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 in accompaniment chords to ⟨*int*⟩ (0–127).

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

%%MIDI beataccents: reverts to normally emphasised notes. See also %%MIDI nobeatac-
cents.

%%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 bendstring ⟨***int***⟩ *[int]...*: specifies pitch trajectory for bent notes.

%%MIDI bendvelocity ⟨***int1***⟩ ⟨***int2***⟩: specifies the change in pitch and rate of change of
bent notes.

%%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 redefi-
nes existing ones as was seen in Section 6.1.10 (p. 99).

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

%%MIDI chordvol ⟨***int***⟩: sets the volume (velocity) of 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 controlcombo: indicates that the next %%MIDI controlcombo directive does not
replace previous ones.

%%MIDI controlstring ⟨***int1 int2 ...***⟩: used for note shaping; this directive is still in de-
velopment.

%%MIDI deltaloudness ⟨***int***⟩: by default, !crescendo! and !dimuendo! modify the beat
variables ⟨*vol1*⟩ ⟨*vol2*⟩ ⟨*vol3*⟩ 15 volume units. This directive 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 6.1.14 (p. 101).

`%%MIDI drummap ⟨`***str***`⟩ ⟨`***int***`⟩:` associates the note ⟨*str*⟩ (in Abc notation) to the a percussion instrument, as listed in Section A.7.2 (p. 163).

`%%MIDI drumoff` turns drum accompaniment off.

`%%MIDI drumon` turns drum accompaniment on.

`%%MIDI expand:` expand notes to make them overlap the following notes.

`%%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 chord ⟨*str*⟩ over ⟨*n*⟩ consecutive bars of equal length. The length of ⟨*str*⟩ should be evenly divisible by ⟨*n*⟩, otherwise the chords will not play properly.

`%%MIDI gchord ⟨`***str***`⟩:` sets up how accompaniment chords are generated; please see Section 6.1.9 (p. 97).

`%%MIDI gchordoff:` turns accompaniment chords off.

`%%MIDI gchordon:` turns accompaniment 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 directive 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 directive, 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.

`%%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 ptstress ⟨*string*⟩: uses the stress pattern in the file name given as ⟨*string*⟩.

%%MIDI randomchordattack: delays the start of chord notes by a random number of MIDI units.

%%MIDI ratio ⟨*int1 int2*⟩: sets the ratio of note lengths in broken rhythm. Normally, c>c will make the first note three times as long as the second; this ratio can be changed with %%ratio 2 1.

%%MIDI rtranspose ⟨*int1*⟩: transposes relatively to a prior %%transpose directive by ⟨*int1*⟩ semitones; the total transposition will be ⟨*int1 + int2*⟩ semitones.

%%MIDI snt ⟨*int*⟩ ⟨*float*⟩: alters the standard MIDI pitch of a note ⟨*0–127*⟩; e.g. %%MIDI snt 60 60.5 alters the pitch of C.

%%MIDI stressmodel ⟨*int*⟩: uses the stress model given as parameter.

%%MIDI temperamentequal ⟨*int1*⟩ *[int2] [int3] [int4]*: uses a tempered scale defined by ⟨*int1*⟩ equal divisions of *[int2]* cents (default: 1200). The parameters *[int3]* and *[int4]* give the size of the fifth interval and of accidentals.

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

%%MIDI temperamentnormal: restores normal temperament.

%%MIDI transpose ⟨*int*⟩: transposes the output by ⟨*int*⟩ semitones. ⟨*int*⟩ may be positive or negative.

%%MIDI trim ⟨*int1*⟩ ⟨*int2*⟩: controls the articulation of notes and chords by placing silent gaps between the notes. The length of these gaps is determined by ⟨*int1*⟩/⟨*int2*⟩ and the unit length is specified by the L: field. These gaps are produced by shortening the notes by the same amount. If the note is already shorter than the specified gap, then the gap is set to half the length of the note. It is recommended that ⟨*int1*⟩/⟨*int2*⟩ be a fraction close to zero. Trimming is disabled inside slurs as indicated by parentheses. Trimming is disabled by setting ⟨*int1*⟩ to 0.

%%MIDI tuningsystem ⟨*string*⟩: when ⟨*string*⟩ is comma53, the Holdrian scale (53 microtones) will be used instead of the standard 12-note tempered scale.

%%MIDI vol ⟨*int*⟩: increases the loudness of the next note by ⟨*int*⟩.

%%MIDI volinc: equivalent to %%MIDI vol.

# A.7 MIDI Instruments

## A.7.1 Standard instruments

The following is a complete list of the General MIDI standard instruments, subdivided in instrument families. Remember, when using `abc2midi` the instrument number must be decreased by 1.

**Piano**
**1**. Acoustic Grand
**2**. Bright Acoustic
**3**. Electric Grand
**4**. Honky-Tonk
**5**. Electric Piano 1
**6**. Electric Piano 2
**7**. Harpsichord
**8**. Clavinet

**Guitar**
**25**. Nylon String Guitar
**26**. Steel String Guitar
**27**. Electric Jazz Guitar
**28**. Electric Clean Guitar
**29**. Electric Muted Guitar
**30**. Overdriven Guitar
**31**. Distortion Guitar
**32**. Guitar Harmonics

**Ensemble**
**49**. String Ensemble 1
**50**. String Ensemble 2
**51**. SynthStrings 1
**52**. SynthStrings 2
**53**. Choir Aahs
**54**. Voice Oohs
**55**. Synth Voice
**56**. Orchestra Hit

**Pipe**
**73**. Piccolo
**74**. Flute
**75**. Recorder
**76**. Pan Flute
**77**. Blown Bottle
**78**. Skakuhachi
**79**. Whistle
**80**. Ocarina

**Synth Effects**

**Chromatic Percussion**
**9**. Celesta
**10**. Glockenspiel
**11**. Music Box
**12**. Vibraphone
**13**. Marimba
**14**. Xylophone
**15**. Tubular Bells
**16**. Dulcimer

**Bass**
**33**. Acoustic Bass
**34**. Electric Bass(finger)
**35**. Electric Bass(pick)
**36**. Fretless Bass
**37**. Slap Bass 1
**38**. Slap Bass 2
**39**. Synth Bass 1
**40**. Synth Bass 2

**Brass**
**57**. Trumpet
**58**. Trombone
**59**. Tuba
**60**. Muted Trumpet
**61**. French Horn
**62**. Brass Section
**63**. SynthBrass 1
**64**. SynthBrass 2

**Synth Lead**
**81**. Lead 1 (square)
**82**. Lead 2 (sawtooth)
**83**. Lead 3 (calliope)
**84**. Lead 4 (chiff)
**85**. Lead 5 (charang)
**86**. Lead 6 (voice)
**87**. Lead 7 (fifths)
**88**. Lead 8 (bass+lead)

**Ethnic**

**Organ**
**17**. Drawbar Organ
**18**. Percussive Organ
**19**. Rock Organ
**20**. Church Organ
**21**. Reed Organ
**22**. Accordion
**23**. Harmonica
**24**. Tango Accordion

**Solo Strings**
**41**. Violin
**42**. Viola
**43**. Cello
**44**. Contrabass
**45**. Tremolo Strings
**46**. Pizzicato Strings
**47**. Orchestral Strings
**48**. Timpani

**Reed**
**65**. Soprano Sax
**66**. Alto Sax
**67**. Tenor Sax
**68**. Baritone Sax
**69**. Oboe
**70**. English Horn
**71**. Bassoon
**72**. Clarinet

**Synth Pad**
**89**. Pad 1 (new age)
**90**. Pad 2 (warm)
**91**. Pad 3 (polysynth)
**92**. Pad 4 (choir)
**93**. Pad 5 (bowed)
**94**. Pad 6 (metallic)
**95**. Pad 7 (halo)
**96**. Pad 8 (sweep)

**Percussive**

**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)

**105**. Sitar
**106**. Banjo
**107**. Shamisen
**108**. Koto
**109**. Kalimba
**110**. Bagpipe
**111**. Fiddle
**112**. Shanai

**113**. Tinkle Bell
**114**. Agogo
**115**. Steel Drums
**116**. Woodblock
**117**. Taiko Drum
**118**. Melodic Tom
**119**. Synth Drum
**120**. Reverse Cymbal

**Sounds Effects**
**121**. Guitar Fret Noise
**122**. Breath Noise
**123**. Seashore
**124**. Bird Tweet
**125**. Telephone Ring
**126**. Helicopter
**127**. Applause
**128**. Gunshot

## A.7.2 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 | | |

THIS manual explains how to make beautiful sheet music and MIDI files using a computer, some free and open source software, and the ABC 2 music notation. It is aimed at musicians with some computer expertise who don't want to spend a lot of money on commercial music software. Music teachers, students, amateur and professional musicians may benefit greatly from this guide and from the software it describes.