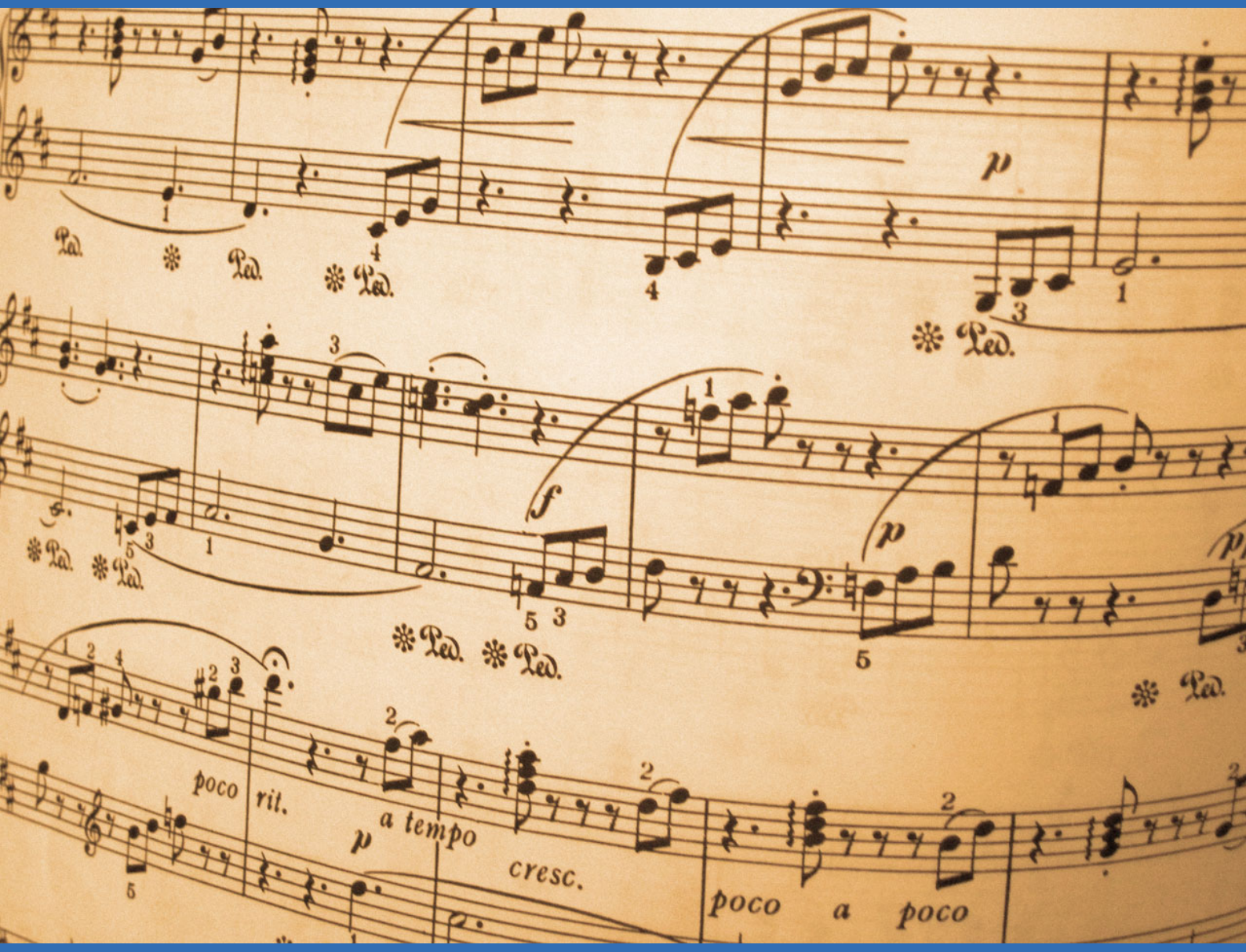
Making Music with ABC 2

A practical guide to the ABC notation

Guido Gonzato

Making Music with ABC 2

Making Music with ABC 2 (formerly, *Making Music with ABC PLUS*)
Version: October 2021
Copyleft © Guido Gonzato, PhD, 2003–2021

Image cover by Vera Kratochvil, via http://www.publicdomainpictures.net

Typeset with LaTeX, with the help of the Jed editor and LaTeXJed.

This manual is released under the terms of the GNU Free Documentation License 1.3:
http://www.gnu.org/licenses/fdl.html

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

To Annarosa,
Bruno,
Lorenzo
♡

Contents

About This Guide xv

1 Music on the Computer with Abc 2 1

1.1 Introduction . . . 1
1.1.1 Requirements . . . 2
1.1.2 Software . . . 2
1.1.3 Why Abc? . . . 3
1.2 Getting Started . . . 4
1.2.1 Getting the Programs . . . 5
1.2.2 Which Program? . . . 5
abcm2ps . . . 6
abc2svg . . . 6
abcMIDI . . . 6
Abc/MusicXML Translators . . . 6
TradMusician . . . 7
Other Programs . . . 7
1.2.3 Abc in a Nutshell . . . 7
1.2.4 Our First Abc File . . . 8
1.2.5 Using the Command Line . . . 9
Windows . . . 9
GNU/Linux, macOS . . . 11
Android (expert users only!) . . . 13
Explanations, for everybody . . . 14

2 Melody 15

2.1 Notes and Symbols . . . 15
2.1.1 Manual or Automatic Formatting? . . . 15
2.1.2 Note Pitch: *A-G a-g ,'* . . . 15
2.1.3 Note Length: *L:* . . . 17
2.1.4 Rests and Spacing: *z Z x y* . . . 18
2.1.5 Accidentals: *^_ =* . . . 19
2.1.6 Dotted Notes: *< >* . . . 19
2.1.7 Ties, Slurs, Staccato: *- () .* . . . 20
2.1.8 Tuplets: *(n* . . . 21
2.1.9 Chords: *[]* . . . 22
2.1.10 Grace Notes: *~ {}* . . . 23
2.1.11 Inline Fields . . . 23
2.2 Music Properties . . . 24

2.2.1 Key signatures and Clefs: K: . . . 24
Key Signatures . . . 24
Clefs . . . 25
Changing the Clef: I:clef . . . 27
Bass and Alto Clefs Compatibility Issues . . . 27
2.2.2 Time Signature: M: . . . 28
2.2.3 Bars, Repeats, Endings: | / : [ ] . . . 29
2.2.4 Lyrics: W: w: . . . 30
2.2.5 Title, Composer, Tempo: T: C: Q: . . . 32
2.2.6 Text and Foreign Characters . . . 33
2.2.7 UTF-8 Characters and Symbols . . . 34
2.2.8 Parts: P: . . . 36
2.2.9 Accompaniment Chords: "" . . . 37
2.2.10 Text Annotations: "^_ <>@" . . . 38
Courtesy Accidentals . . . 39
2.2.11 Decorations: !symbol! . . . 40
2.2.12 Symbol Lines: s: . . . 42
Figured Bass . . . 43
2.2.13 Redefinable Symbols: U: . . . 43

3 Harmony **45**
3.1 Polyphony in ABC 2 . . . 45
3.1.1 Voices and Systems: V: . . . 45
3.1.2 Positioning Voices: %%score . . . 49
3.1.3 Typesetting Reductions . . . 53
3.1.4 Voice Overlay: & . . . 55
3.1.5 Writing Clean Sources . . . 56

4 Formatting **59**
4.1 Formatting Parameters . . . 59
4.1.1 How Directives Work . . . 60
4.1.2 Using Format and Header Files . . . 62
4.1.3 Changing Parameters . . . 63
4.1.4 Directives as I: Fields . . . 63
4.1.5 Using Fonts and Text . . . 65
4.1.6 Line, Staff, Tune, and Page Breaks . . . 68
4.1.7 Avoiding Line Breaks . . . 70
4.1.8 Controlling Measures . . . 71
4.1.9 Voice Size . . . 72
4.1.10 Repeated Sequences (*Simile*) . . . 72
4.1.11 The Grand Staff . . . 73
4.1.12 Change of System . . . 73
Blank Sheet Music . . . 74
4.1.13 Positioning Music Elements . . . 76
4.1.14 Customising Slurs, Beams, Stems, and Flags . . . 77
4.1.15 Customising Accompaniment Chords . . . 78
4.1.16 Customising Tuplets . . . 79
4.1.17 Customising Volta Brackets . . . 80

4.1.18 Combining Voices . . . 80
4.1.19 Colours . . . 81
4.1.20 Multi-column Output . . . 82
4.1.21 Customising Titles . . . 83
4.1.22 Headers and Footers . . . 84
4.1.23 Inserting Graphics Files (abcm2ps) . . . 85
4.1.24 Numbering Measures and Pages . . . 86
4.1.25 Staff Gaps . . . 86
4.1.26 Ambitus (abc2svg) . . . 87
4.1.27 Embedding XHTML (abc2svg) . . . 87
4.1.28 Saving Space . . . 88
4.2 Transposition . . . 88
4.2.1 Using %%transpose . . . 89
4.2.2 K: and V: Modifiers (abc2svg) . . . 89

5 Tablatures and Fingering Charts 91
5.1 Tablatures . . . 91
5.2 Chord Grids . . . 92
5.3 Chord Diagrams . . . 93
5.4 Fingering Charts (abcm2ps) . . . 95

6 Tune Collections 97
6.1 Index Number: *X:* . . . 97
6.2 Information Fields . . . 98
6.3 Including ABC Files . . . 99
6.4 Songbooks . . . 99
6.5 Selecting Tunes . . . 101
6.6 Incipits . . . 102

7 Playing 105
7.1 MIDI Conversion . . . 105
7.1.1 A Software MIDI Player: TiMidity++ . . . 106
7.1.2 Our First Midi . . . 106
7.1.3 Example: How To Make a Ringtone . . . 107
7.1.4 Supported Decorations . . . 108
7.1.5 %%MIDI Directives . . . 108
7.1.6 Avoiding abcm2ps Extensions . . . 109
7.1.7 Voices and Instruments . . . 109
7.1.8 MIDI Transposition . . . 110
7.1.9 Changing Instruments in Repeats . . . 111
7.1.10 Accompaniment Chords . . . 111
7.1.11 New Accompaniment Chords . . . 113
7.1.12 Customising Beats . . . 114
7.1.13 Arpeggios . . . 114
7.1.14 Broken Rhythm . . . 115
7.1.15 Drum Patterns . . . 116
7.1.16 Percussion Instruments and Drum Mapping . . . 117
7.1.17 Portamento . . . 118

7.1.18 Drone . . . 118
7.1.19 Global Settings . . . 119
7.2 Differences and Incompatibilities . . . 120

8 Advanced Customisation **121**
8.1 Font Directives, Revisited (abc2svg) . . . 121
8.2 New PostScript and SVG Routines . . . 122
8.2.1 Defining New Symbols . . . 123
8.3 Note Mapping . . . 125
8.4 Renaming Symbols . . . 126
8.5 Accompaniment Chords in Italian Notation . . . 127
8.6 Adding Text Fonts (abcm2ps) . . . 128
8.7 Adding Text Fonts (abc2svg) . . . 130
8.8 Using SMuFL Fonts . . . 131

9 ABC and MusicXML **133**
9.1 Introducing MusicXML . . . 133
9.2 Using xml2abc-js . . . 133
9.3 Using abc2xml.py and xml2abc.py . . . 134
9.3.1 Basic Options . . . 135
9.4 Converting ABC to MuseScore . . . 135
9.5 Converting ABC to Lilypond . . . 136
9.6 Converting ABC to Verovio . . . 137
9.7 abc2xml.py Extensions . . . 137
9.7.1 Tremolo . . . 138
9.7.2 Percussion Maps . . . 138
9.7.3 Tablatures, Revisited . . . 141
9.8 ABC/MusicXML Conversion: Limitations . . . 144

10 Advanced Usage **147**
10.1 Cropping PDF Files . . . 147
10.2 Using abc2svg in the Command Line . . . 148
10.2.1 Running abc2svg Scripts . . . 148
10.2.2 Converting html to PDF . . . 149
10.2.3 Inserting Music in Word Processors . . . 150
10.2.4 Inserting Music in LaTeX . . . 150
10.3 Using abc.sty . . . 151
10.4 Converting MIDI to ABC . . . 152
10.5 The abcpp Preprocessor . . . 153
10.5.1 Basic Usage . . . 153
10.5.2 Redefining Symbols . . . 154
10.6 Conversions: abc2abc . . . 155

11 The End **157**
11.1 What Now? . . . 157
11.2 Final Comments . . . 157
11.3 In Loving Memory of Annarosa Del Piero, 1930–2000 . . . 157

A Bits & Pieces **159**
A.1 Web Links . . . 159
A.2 ABC Fields . . . 160
A.3 ABC Summary . . . 160
A.4 Common Errors and Bad Practices . . . 164
A.4.1 Syntax Errors . . . 164
A.4.2 Character Encoding . . . 164
A.4.3 Lines Too Long . . . 164
A.4.4 `X:` Number in Titles . . . 164
A.4.5 Ties vs. Slurs . . . 165
A.4.6 Outdated Syntax . . . 165
`Q:` Syntax . . . 165
`!` as End of Line . . . 165
`+` for Decorations and Chords . . . 165
\ in `w:` Lines . . . 165
`middle=` Specifier . . . 165
A.5 Formatting Directives . . . 165
A.5.1 Page Format . . . 167
A.5.2 Text . . . 167
A.5.3 Fonts . . . 168
A.5.4 Spacing . . . 170
A.5.5 Other Directives . . . 172
A.5.6 Deprecated Directives . . . 177
A.6 PostScript Fonts . . . 179
A.7 `abcMIDI` Directives . . . 179
A.8 MIDI Instruments . . . 183
A.8.1 Standard instruments . . . 183
A.8.2 Percussion Instruments . . . 184

List of Tables

1.1 Note names in English and Latin notation. 14

2.1 Modal scales. 24
2.2 Clefs and associated `K:` fields. 26
2.3 How to obtain characters of foreign languages. 34
2.4 Types of accompaniment chords. 37

4.1 Default fonts for `abcm2ps` and `abc2svg`. 65

5.1 Keys for the `flute.fmt` format file. 95

7.1 Standard notes and corresponding MIDI pitches. 119

List of Figures

1.1 Writing a tune in a web browser using `abc2svg`. 9

2.1 Standard decorations. 41

4.1 A piece with a variable number of systems. 75

9.1 Importing and displaying MusicXML files in `xml2abc-js`. 134

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

MUSICIANS 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. The ABC notation was introduced by Chris Walshaw in 1993, and 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. For more information, please refer to the official ABC home page: http://abcnotation.com.

ABC was later expanded to provide multiple voices (polyphony), page layout details, and MIDI commands. This major release of the ABC notation is called ABC 2.

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 benefit 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 unofficial ABC 2 home page was created to provide ready to use software and documentation: 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, rather than 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.

In conclusion: 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 typeset, i.e. turned to music printed on paper or displayed in web pages. It can also be played and/or turned 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`, the ABC/MusicXML translators `abc2xml.py` and `xml2abc.py`, and the TradMusician player. 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 called 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. `abc2svg` 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.

Last but not least: TradMusician, written by Nicolas Jouve, is a complete ABC player and viewer for Android.

Please note that `abcm2ps`, `abc2svg`, `abcMIDI`, the ABC/MusicXML tools, and TradMusician *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 playback. 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`, `abc2midi`, part of `abc2svg`, and the `abc2xml-.py/xml2abc.py` translators are *command-line 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. Currently, one of the most popular is EasyABC, 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.

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;

multiple formats: Abc programs make music that can be printed on paper, or viewed on computer screens, or managed with tablets and phones;

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, there are two main options: you could use `abc2svg` in a web browser, or the command line programs. The first option is easier, the second is more powerful but more complex.

In the latter case, you'll follow these steps:

1. using a text 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`.

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 (ready to install packages)
- the `abc2svg` online scorewriter:
  http://moinejf.free.fr/js/edit-1.xhtml
- `abcMIDI`:
  http://ifdo.ca/~seymour/runabc/top.html
  http://abcplus.sourceforge.net/#abcMIDI (ready to install packages)
- the Abc/MusicXML translators:
  https://wim.vree.org/svgParse/abc2xml.html
  https://wim.vree.org/svgParse/xml2abc.html
- TradMusician:
  http://nicolasjouve.fr/tradmusician/
- Ghostscript (all platforms except Android):
  http://www.ghostscript.com/
  As of this writing, the latest version is 9.55; you may want to download the free, GNU Affero licensed version.
- Ghostscript for macOS:
  http://pages.uoregon.edu/koch/Ghostscript-9.55-Full.pkg
- Sumatra PDF, a fast PDF and PostScript viewer (Windows):
  https://www.sumatrapdfreader.org

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 used to be the best option for printing on paper. It also produces SVG output, but abc2svg is better at that.

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, or if you use a program that runs abcm2ps internally, then abcm2ps is the best program for you.

abc2svg

abc2svg typesets music in SVG/HTML, which is meant to be displayed in web pages. Output can also be converted to PDF.

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.

The main disadvantage of abc2svg is its lack of fine control on PDF output. Printing, in fact, is delegated to external tools, which may produce slightly different PDF files. Furthermore, when it is integrated in browser-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 as a single file, 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.

TradMusician

TradMusician is a complete Abc player and viewer for Android that also manages tune collections. It lets the user browse and select tunes, display them in standard notation, and play them at adjustable speed.

TradMusician is free and available on Google Play; it's probably the best choice for the Android platform.

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`, possibly 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 let me stress this point: *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 1.2.5, page 9.

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 playback. That said, ABC 2 does provide directives for specifying such details; these will be examined in Chapter 4 (page 59).

If you don't have a US keyboard, some important characters may be missing. To obtain these characters in 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* `‘`.
- `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 Chromium (or derivatives such as Chrome), because it supports a handy, multiplatform "Save as PDF" print option. Other browsers may work.

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 `MyMusicFile.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.

You can use `abc2svg` online, visiting the page http://moinejf.free.fr/js/edit-1.xhtml, or offline, using the portable `abc2svg` that you can download at the ABC 2 page. Start it the way you prefer.

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 (Figure 1.1).

Click on the text input area and type (or copy and paste) this source *verbatim*:

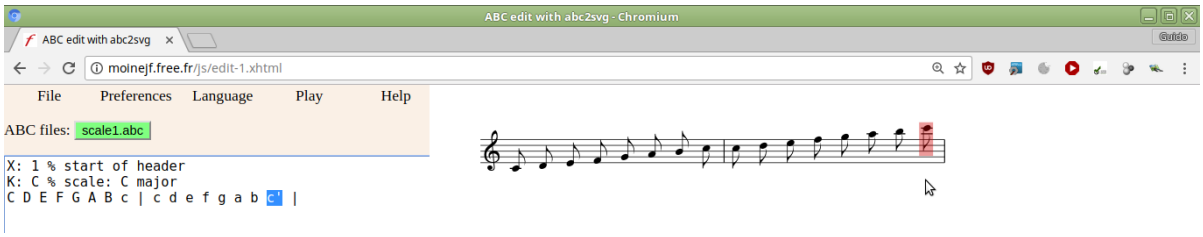
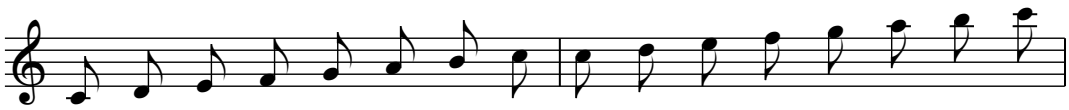


Figure 1.1: Writing a tune in a web browser using `abc2svg`.

```
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 listen to the tune, right click on the right-hand side and select Play/Tune from the popup menu; notes will be highlighted as they are played.

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.

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. The command `abctopdf`, which is part of `abc2svg`, will only be described for GNU/Linux and macOS.

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-8.14.12 (2021-07-14)
Options: A4_FORMAT PANGO
Default format directory: /usr/share/abcm2ps
C:\Users\Guido>abc2midi -ver
4.61 October 11 2021 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*:

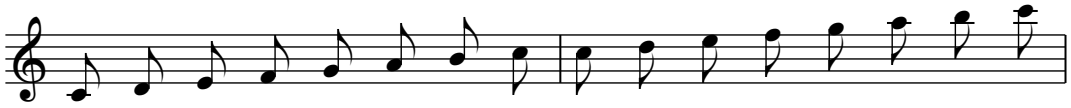
```
X: 1 % start of header
K: C % scale: C major
C D E F G A B c | c d e f g a b c' |
```

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-8.14.12 (2021-07-14)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 20236 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-8.14.12 (2021-07-14)
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, 20214 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
4.61 October 11 2021 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-8.14.12 (2021-07-14)
Options: A4_FORMAT PANGO
Default format directory: /usr/share/abcm2ps
$ abc2midi -ver
4.61 October 11 2021 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' |
```

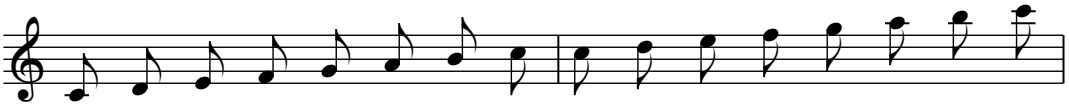
Save it, then switch back to the command prompt. We are ready to make the PostScript output using `abcm2ps`:

```
~/abcmusic$ abcm2ps -c -O= scale1.abc
abcm2ps-8.14.12 (2021-07-14)
File scale1.abc
Output written on scale1.ps (1 page, 1 title, 20236 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; in 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-8.14.12 (2021-07-14)
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, 20214 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
4.61 October 11 2021 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-8.14.12 (2021-07-14)
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
4.61 October 11 2021 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; you may have to install additional software.

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 4.1.2, page 62).

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 many 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 `abcm2ps`, by default, automatically set the metre as four quarters, and the note length as eighths.

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

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, the answer to these questions depends on the type of music we want to write.

In the case of simple tunes of fixed length, 16 or 32 bars for instance, the best option is probably *manual formatting*; specifically, writing four measures per line. Most traditional music (jigs, reels, bourrées, etc.) may look better when typeset with manual formatting. Besides, some programs like TradMusician only support manual formatting.

For longer pieces such as choral or classical music, the best option is usually *automatic formatting*: we write as many measures per line as we prefer, then `abcm2ps` or `abc2svg` rearrange them for optimal output. 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 this manual, we will use automatic formatting. All scores will be typeset using `abc2svg` and automatic line breaks, unless otherwise specified.

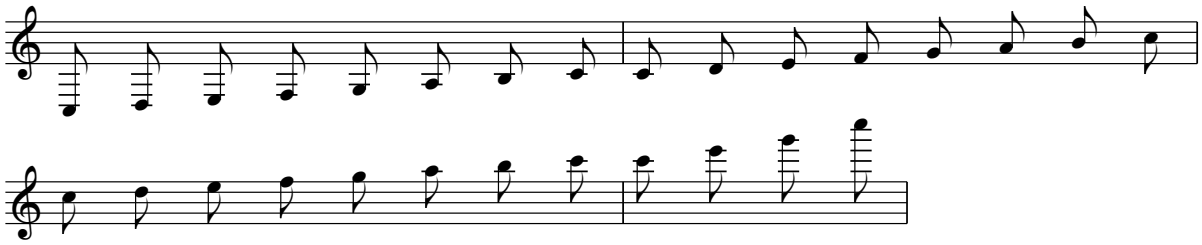
2.1.2 Note Pitch: *A–G a–g ,'*

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 (page 24).

`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 eight notes); 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.

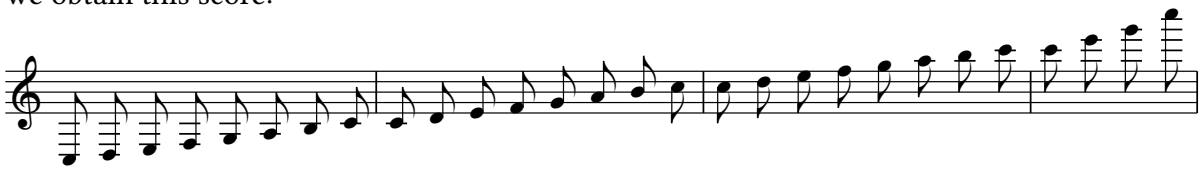
We can instruct `abcm2ps` and `abc2svg` to typeset the score with automatic formatting. This is done inserting the following line:

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

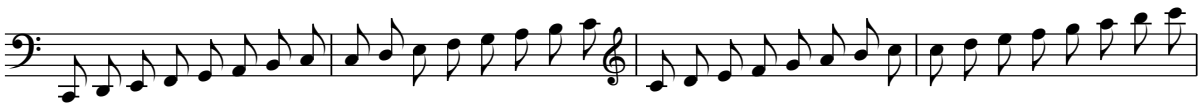
at the top of the source. This line is one of the many directives available that let us fine-tune the appearance of the score; we will study them in Chapter 4, page 59.

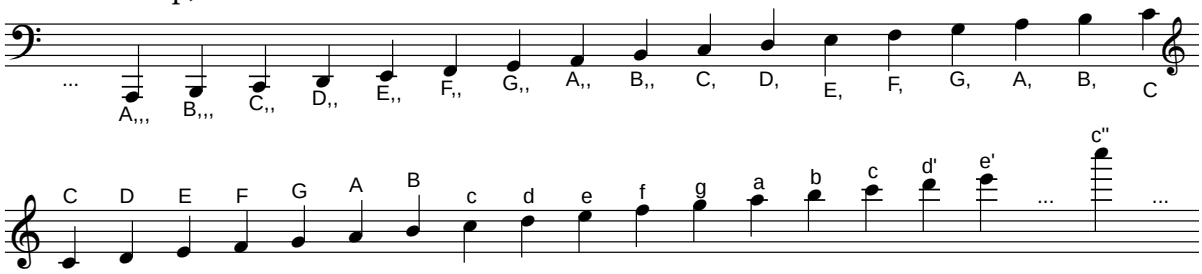
Converting the new source:

```
%%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. Let's remove the `treble` indication in the `K:` field:

```
%%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:*

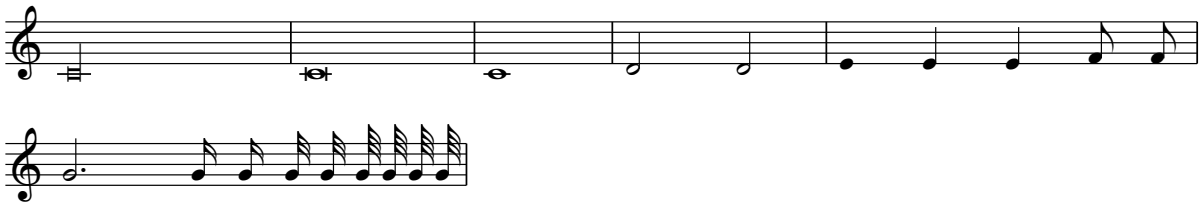
Unless otherwise indicated, note length is automatically set depending on the tune metre.

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, page 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.

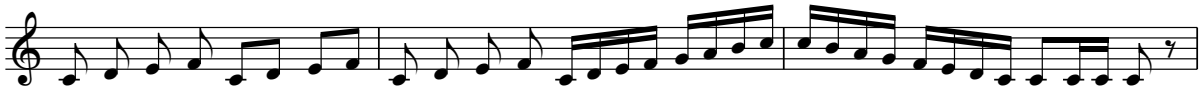
```
X: 1
L: 1/4
K: C
% 16/4  8/4  4/4  2/4  2/4  1/4 1/4 1/4 1/8 1/8
  C16 | C8 | C4 | D2   D2 | E0  E   E   F/  F/|
% 3/4  1/16  1/16  1/32  1/32  1/64  1/64  1/64  1/64
  G3   G//   G/4   G/8   G/8   G/16  G/16  G/16  G/16|
```



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 is 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 (page 59) 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.

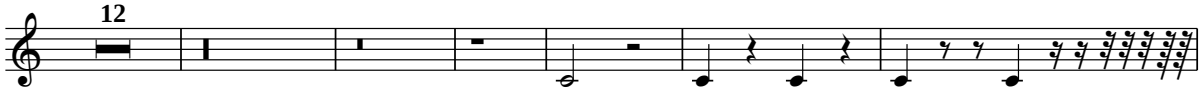
Spaces between notes and measure bars can be freely inserted when the notes are longer than one eighth, and are used to improve the legibility of the source. But *spaces between notes that are equal or lesser than one eighth are not optional.* If these notes are not separated by spaces, they will be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

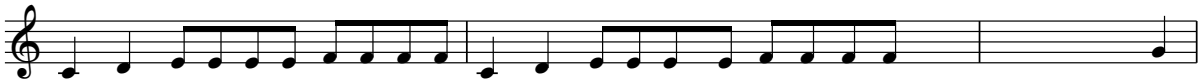
```
X: 1
K: C
%
C D E F CD EF | C D E F C/D/E/F/ G/A/B/c/ |
c/B/A/G/ F/E/D/C/ CC/C/ C z |
```

Commonly, notes are beamed "by beat": each group of eigth (or less) notes corresponds to a beat in the measure. Unbeamed notes may be used in vocal music; see Section 2.2.4, page 30.

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 also apply to rests, 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:

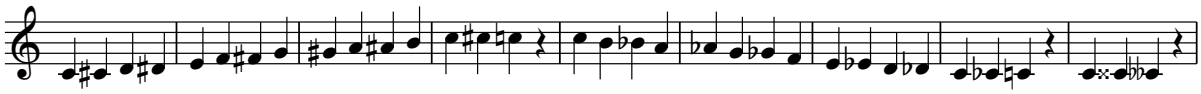
```
X: 1
L: 1/4
K: C
Z12 | z16 | z8 | z4 | C2 z2 | C z C z |
C z/ z/ C z//z//z///z///z///z////z//// |
```

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

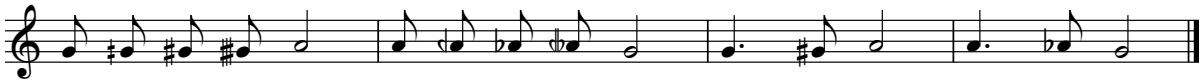
```
X: 1
L: 1/4
K: C
C D E/E/E/E/ F/F/F/F/ | C D E/E/E/yyE/ F/yF/yF/yF/ yyyy | xxxG |
```

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 (page 49).

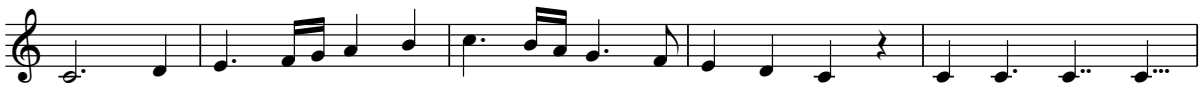
2.1.5 Accidentals: ^ _ =

Sharp ♯ is denoted by a caret `^` 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 `^^` and double flat by `__`:

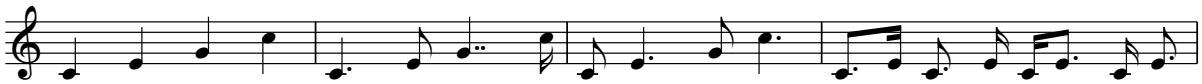
```
X: 1
L: 1/4
K: C
C ^C D ^D | E F ^F G | ^G A ^A B | c^c=cz |
c B _B A | _A G _G F | E _E D _D | C_C=Cz |
C ^^C __C z |
```

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: < >

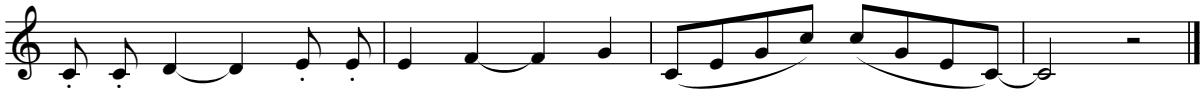
Notes followed by a dot are typeset in two ways. First, by specifying the right length:

```
X: 1
L: 1/4
K: C
C3 D | E3/2 F//G// A B | c3/2 B//A// G>F | E D C z |
C C3/2  C7/4 C15/8 |
```

Please note the three last dotted notes: we specified `3/2`, `7/4`, and `15/8` to obtain one, two, or three dots. Admittedly, this notation is not very handy, but isolated dotted notes are not very common. More often, one note is dotted and the following is halved; or vice versa. In this case, 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*; the tie should be placed immediately after the first note. 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.

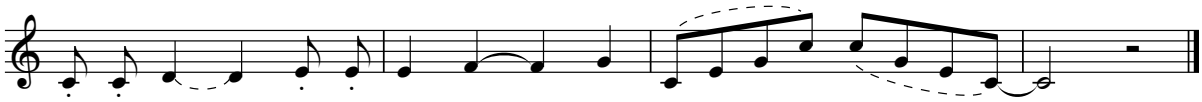
```
X: 1
L: 1/4
K: C
.C/ .C/ D- D .E/ .E/ | EF-FG | (C/E/G/c/) (c/G/E/C/-) | C2 z2 |]
```

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 7.1, page 105) 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) before the open parenthesis:

```
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:

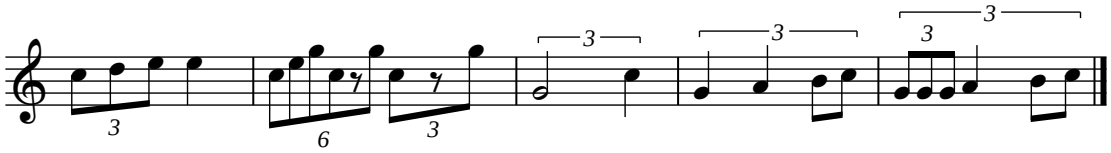
- (2: 2 notes in the time of 3
- (3: 3 notes in the time of 2
- (4: 4 notes in the time of 3
- (5: 5 notes in the time of ⟨*n*⟩ (see below)
- (6: 6 notes in the time of 2
- (7: 7 notes in the time of ⟨*n*⟩
- (8: 8 notes in the time of 3
- (9: 9 notes in the time of ⟨*n*⟩

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

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

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

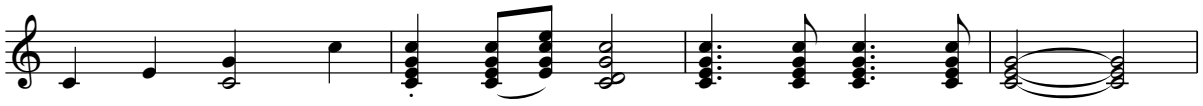


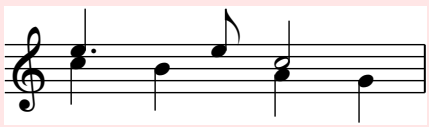
To fine-grain the printing of tuplet indications, please see Section 4.1.16 (page 79).

> ☞ **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 chord length: instead of specifying the length of each note in the chord, 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 typeset chords containing unequal notes as they are, abcMIDI extends shorter notes to match the longer ones.

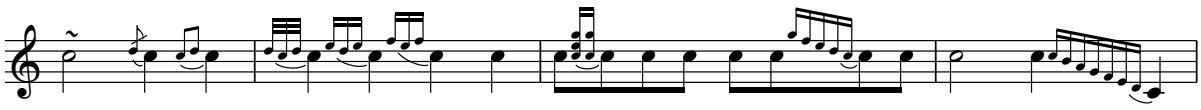
> **Avoid mistakes!**
> Do not mistake chords for something completely different! If you want to obtain something like this:
>
>
>
> you should be aware that these are not chords, but different *voices* on the same staff. Voices will be the subject of Section 3.1 (page 45).

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.)

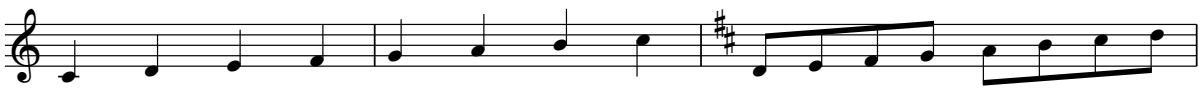
Chords can also be specified instead of a single grace note. However, abc2midi can't play these "grace chords" correctly.

```
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 on. (For the curious: %%graceslurs.)

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 remark][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 (page 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. This is accomplished with *key modifiers*.

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

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

Parameters in ⟨*angular brackets*⟩ are required, while those in *[square brackets]* are optional. Brackets must be left out when writing a parameter!

The only required parameter is the key specified after the `K:` field. All others are optional:

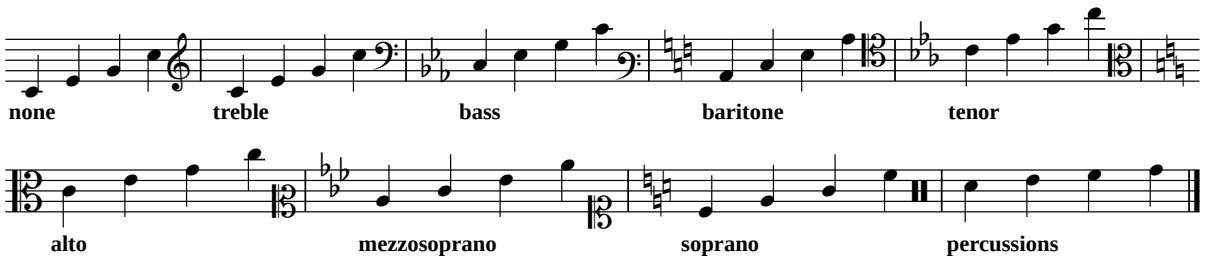
- *[clef=]* may be omitted before the clef type when it is not `none`;
- ⟨*clef type*⟩ can be:
  - the Abc name of a note: `G` or `g` (treble clef), `C` or `c` (alto clef), `F` or `f` (bass clef) are allowed. The specified note will be positioned on the relevant line, as explained below;
  - a clef name: `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*⟩*]*) transposes playback;
- *[+8]* and *[-8]* print `8` above or below the clef;
- *[^8]* 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 their respective fields are listed in Table 2.2 and in the following example (the `w:` lines will be explained in Section 2.2.4):

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`

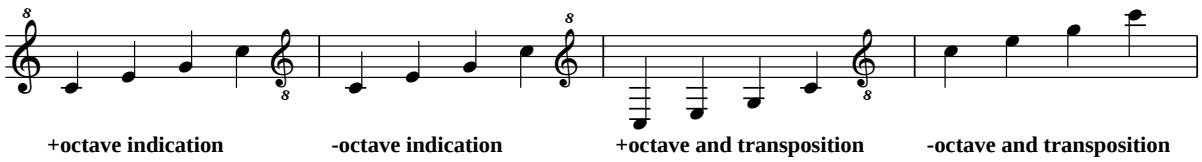
Table 2.2: Clefs and associated `K:` fields.

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



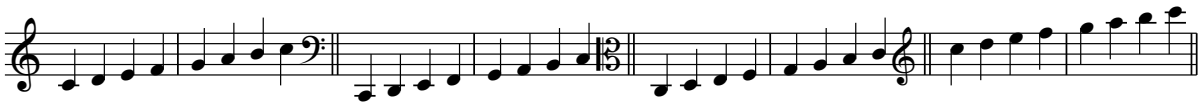
Note the difference between the `+8`/`-8` and `^`/`_8` in clefs indications. The latter prints `8` above or below the clef, and also transposes the notes one octave lower or higher:

```
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, page 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 (page 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 typeset 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 typeset 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
```

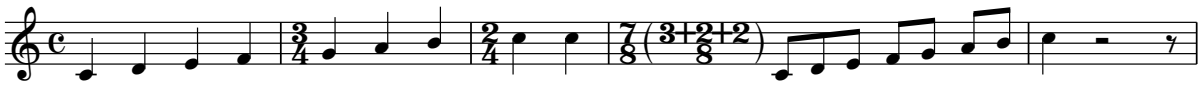
2.2.2 Time Signature: *M:*

The `M:` field specifies the tune time signature (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)`. No spaces are allowed 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:

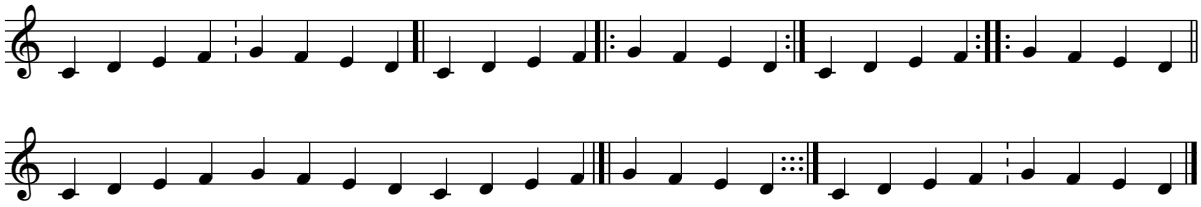
```
X: 1
M: C
K: C
L: 1/4
C D E F | [M: 3/4] G A B | [M: 2/4] c c |
```

```
[M:7/8 (3+2+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: | / : [ ]

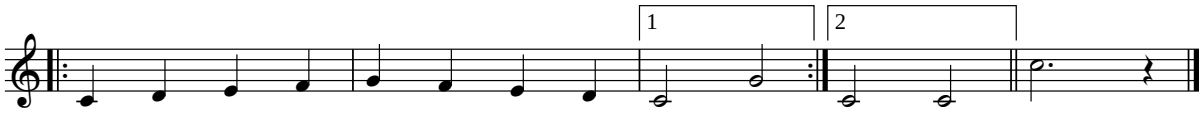
In addition to the basic measure bar, others types of bars can be obtained using combinations of these characters: `|`, `.`, `[`, `]`, and `:`.

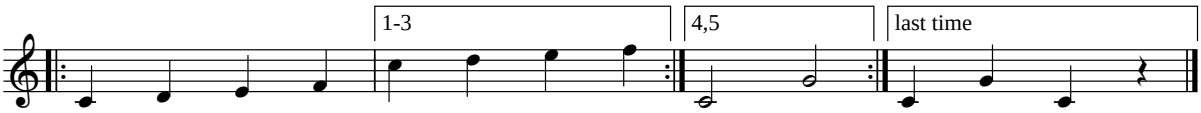
```
X: 1
L: 1/4
K: C
CDEF .| GFED [| CDEF |: GFED :|CDEF :: GFED ||
CDEF [|] GFED CDEF |[| GFED :::|] CDEF : GFED |]
```



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

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 (page 40) 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 `abcMIDI` 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.

> ✍ **Note**
> `abcm2ps` typesets `W:` lines left-aligned under the music, while `abc2svg` typesets them centred.

If stanzas are separated by an empty `W:` line, i.e. not followed by any words, they are automatically arranged in columns so as to take up less space. Stanzas may also be numbered, as in the following example:

```
X:1
K:C
%
W: 1. This is the first column
W: la la la la la
W: 2. First column again
W: la la la la la
W: % an empty W: line starts a new column
W: 3. This is the second column
W: la la la la la
W: 4. Second column again
W: la la la la la
W: % last stanza is centred
W: 5. Final column, centred
W: la la la la la
```

1. This is the first column
   la la la la la
2. First column again
   la la la la la
3. This is the second column
   la la la la la
4. Second column again
   la la la la la

5. Final column, centred
   la la la la la

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. ⟨*n*⟩ `-` characters separated from the previous syllable skip ⟨*n*⟩ 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 typeset a literal `*` character);
- `~` (tilde) joins two syllables under a note;
- `\-` inserts a `-` character.

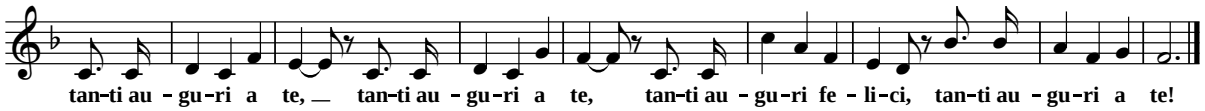
If a `w:` line grows too long, it can be broken and continued on the following line, which must begin with the `+` field. These two lines will match the notes on the previous melody line.

Avoid mistakes!

Previous versions of ABC allowed to continue a lyrics line using \ followed by another `w:` line. Although `abcm2ps` still supports this syntax, its use is deprecated: please use `+` instead!

The following tune is “Happy Birthday” with Italian lyrics:

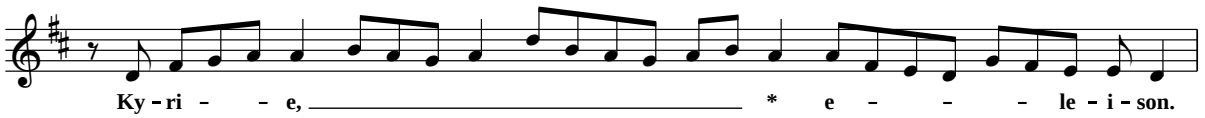
```
X: 1
K: F
C> C | D2C2F2 | E2-E z C> C | D2C2G2 | F2-F z C> C |
w: tan- ti~au- gu- ri a te,_ tan- ti~au- gu- ri a te,
+: * tan- ti~au-
c2A2F2 | E2D z B> B | A2F2G2 | F6 |]
w: gu- ri fe- li- ci, tan- ti~au- gu- ri a te!
%
W: Tanti auguri a te, tanti auguri a te,
W: tanti auguri felici, tanti auguri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



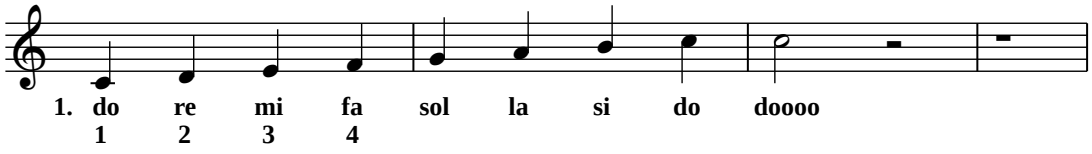
Tanti auguri a te, tanti auguri a te,
tanti auguri felici, tanti auguri a te!

Note that 8th and 16th notes are not beamed; this is common practice when the notes are sung in different syllables.

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 begins with 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 one or more `~` characters 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 digit.

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

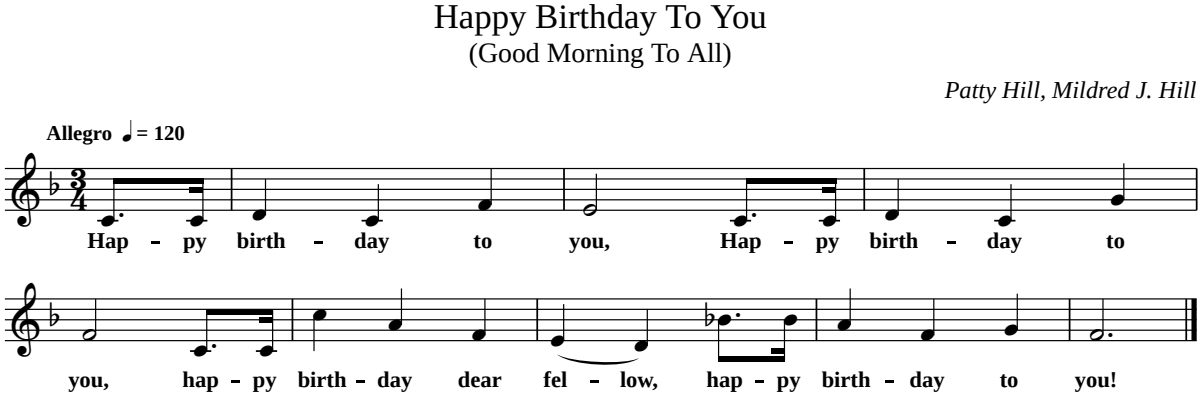


Take special care to write a number of syllables that matches the number of notes! Mismatch between notes and syllables is one of the most common causes of error. Besides, please bear in mind that lyrics may alter the spacing between notes. This effect is especially pronounced when a whole word, or a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



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 (page 59) we will learn how to change the title fonts.

2.2.6 Text and 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 weird 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 modified. For example, to get the "e circumflex" character `ê` we type `\^e`; to get the "a umlaut"character `ä` 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.

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.

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.

UTF-8 is fully supported by `abcm2ps` (with Pango support), `abc2svg`, and the ABC/MusicXML 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 typeset 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 `©`

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
```

```
%
W: A few strange characters in XML:
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)
```

XML Characters

A few strange characters in XML:
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, in addition to backslash sequences you can also use *octal codes*. 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; more precisely, abc2svg only accepts codes \201 to \205.

\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.

> ✍ **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
> ```

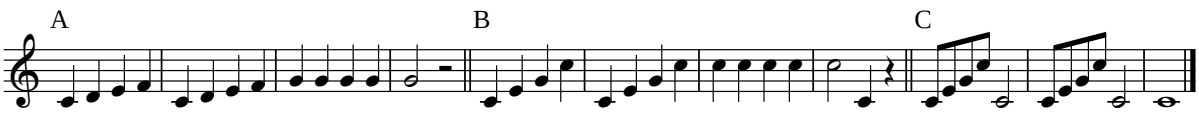
2.2.8 Parts: *P:*

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 |]
```

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.

Type	Meaning
`m` or `min`	minor
`maj`	major (also Δ)
`dim`	diminished (○)
`+` or `aug`	augmented (+)
`sus`	suspended
7, 9, …	seventh, ninth, etc.

Table 2.4: Types of accompaniment chords.

2.2.9 Accompaniment Chords: `""`

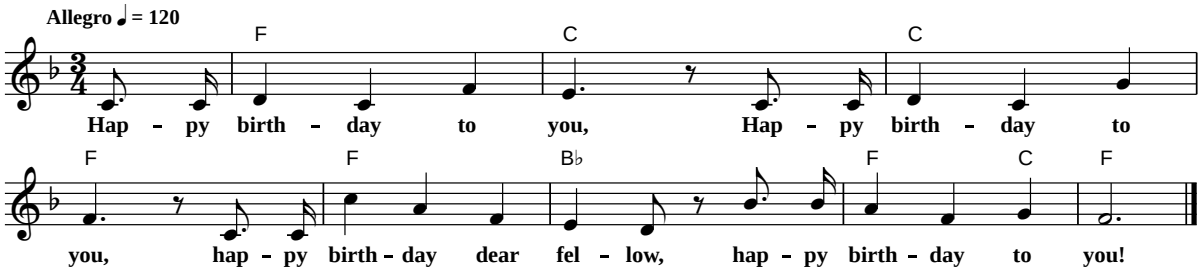
In many songbooks, accompaniment chords (say, for the guitar) are notated as "A", "C7", "Dm", "F♯" etc. above the staff. These chords are notated in ABC by writing the chord name between double quotes `"` immediately before the note, using this format:

⟨*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 7.1.10 (page 111) 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

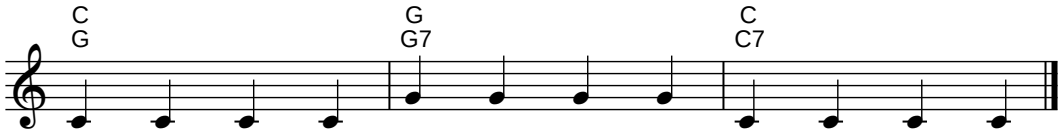


Avoid mistakes!

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, there's a way to typeset accompaniment chords using Italian notes: please refer to Section 8.5 (page 127).

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:

- `^` 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 typeset 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.

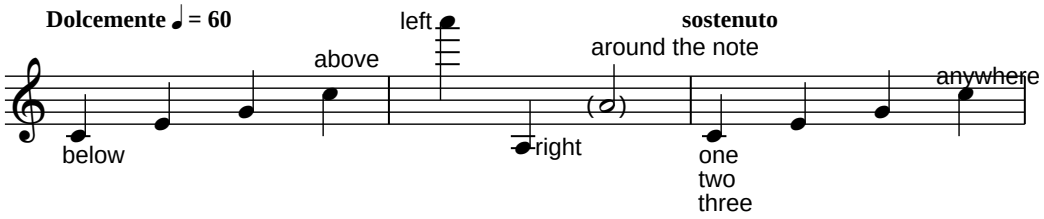
When we use `@`, it is useful to know that the distance between staff lines is 5 pt.

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 (page 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 `^_ <>@` 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.

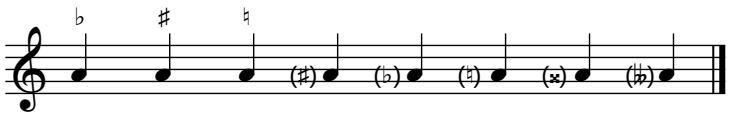
Since accidentals and many more musical symbols are defined as UTF-8 characters, we can insert them in XML syntax (see Section 2.2.7, page 34) if `abcm2ps` has Pango support. In this case, we can use all musical characters defined in block `U+1D100–U+1D1FF`. Accidentals codes are:

- `♯` for sharp;
- `♭` for flat;
- `♮` for natural;
- `𝄪` for double sharp;
- `𝄫` for double flat.

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

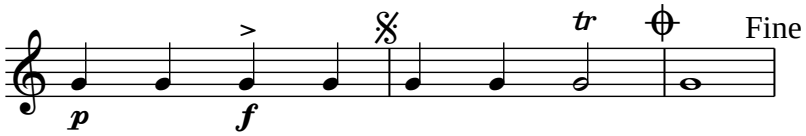
- strings `"\b" "\#" "\="` typeset flat, sharp, and natural above the staff. Other characters in the string are not allowed;
- codes `\001 \002 \003 \004 \005` typeset sharp, flat, natural, double sharp, and double flat;
- codes `\201–\205` are equivalent to `\001–\005`.

```
X: 1
M: none
L: 1/4
```

```
K: C
%
"\b"A "\#"A "\="A
"<(\001)"A "<(\002)"A "<(\003)"A "<(\004)"A "<(\005)"A|]
```

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 typeset above the staff by default, while others are typeset under the staff. Manual positioning is possible, as explained in Section 4.1.13 (page 76).

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 8 (page 121).

Some symbols have synonims:

- `!>!`: `!accent!` or `!emphasis!`
- `!+!`: `!plus!`
- `!mordent!`: `!lowermordent!`
- `!uppermordent!`: `!pralltriller!`
- `!<(!` and `!<)!`: `!crescendo(!` `!crescendo)!`
- `!>(!` and `!>)!`: `!diminuendo(!` `!diminuendo)!`
- `!rbstop!`: `!rbend!`

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

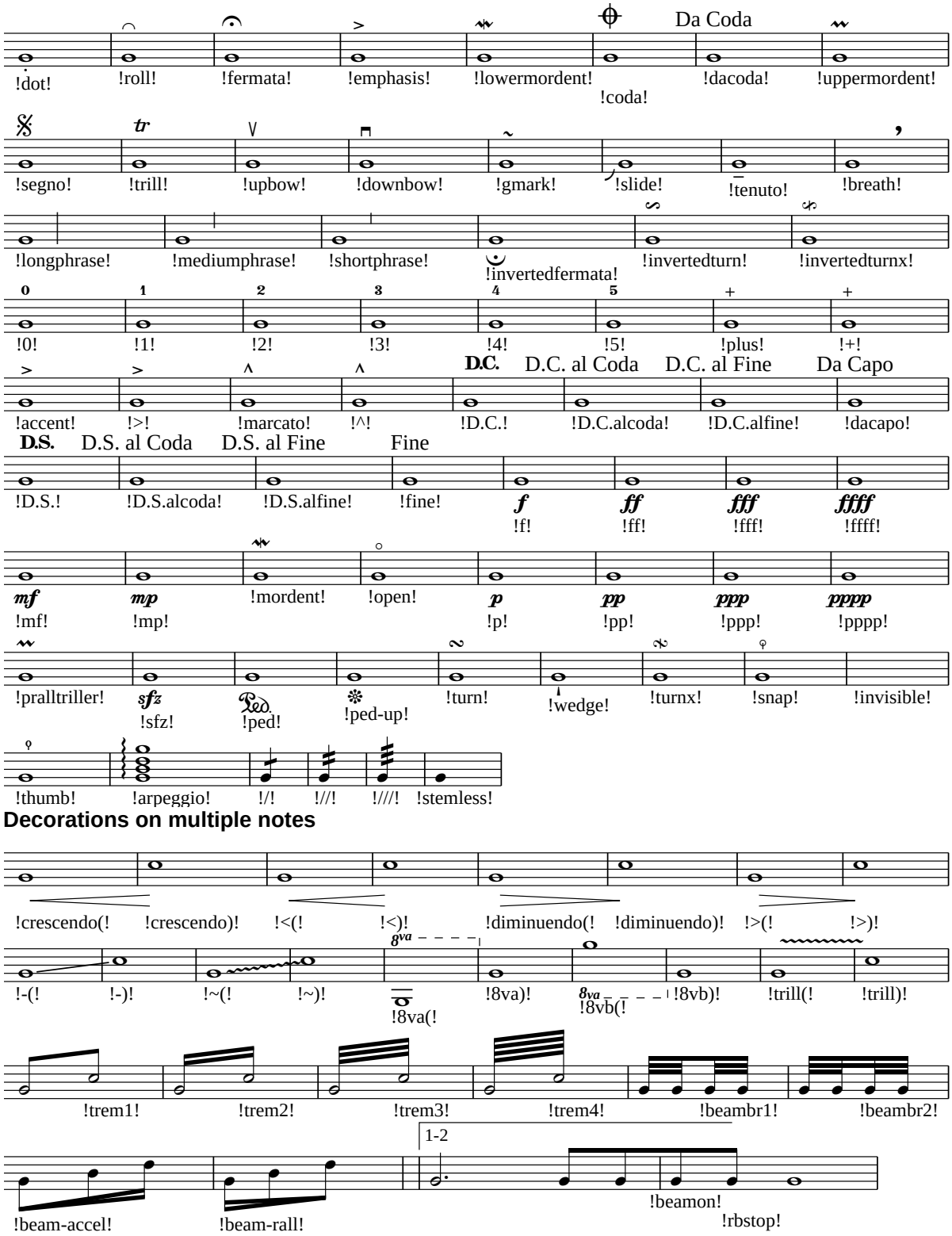
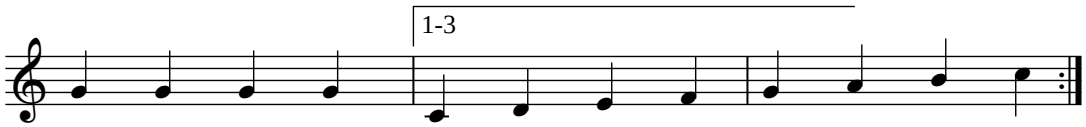



Figure 2.1: Standard decorations.

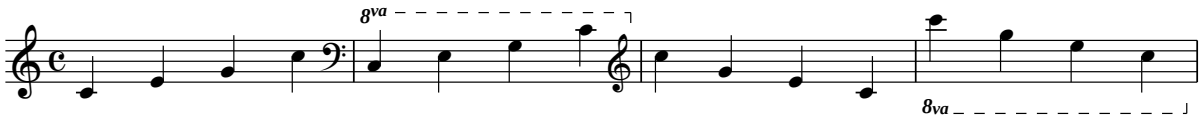
- symbols that include an open parenthesis `(` in their definition start a multi-note decoration, which is ended by the respective symbol with a closed parenthesis `)`. For example, `!<(!` starts a crescendo hairpin, `!<)!` ends it. The same holds for `!>(!`, `!>)!`, and their long-name counterparts;
- `!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 4.1.11 (page 73).

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:|
```

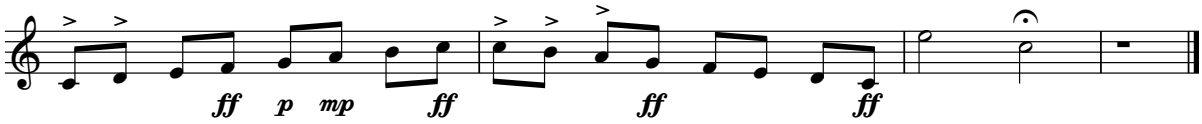


Piano symbols for “ottava alta” and “ottava bassa” indications automatically transpose the typeset notes one octave lower or higher:

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

2.2.12 Symbol Lines: *s:*

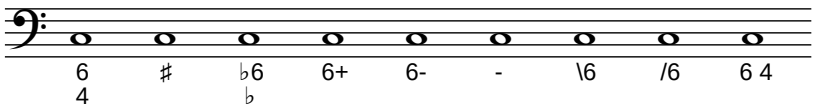
If the tune contains many chords and decorations, an alternative notation may be handy to improve legibility. After a line of music, we write a line that starts with the `s:` field. This line will contain only symbols.

Rules for matching notes and symbols are the same as those explained in Section 2.2.4 (page 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! |
```

Figured Bass

Some 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 typeset them below the staff.

2.2.13 Redefinable Symbols: *U:*

Most symbol names are quite verbose, and they could make the source difficult to read. To solve this snag, we can define a shortcut using the `U:` field.

The `U:` field is followed by a single letter (`~`, A–Z, a–z), followed by `=`, then by the symbol to be shortened. For example, the following `U:` fields define shortcuts for `!trill!`, `!fermata!`, `!tenuto!`:

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

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

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

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

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

♫ ♫ ♫ ♫ ♫

Chapter 3

Harmony

3.1 Polyphony in ABC 2

In 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 ABC could do, and it is actually a lot; many folk musicians do not need anything more.

We now turn our attention to ABC 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*.

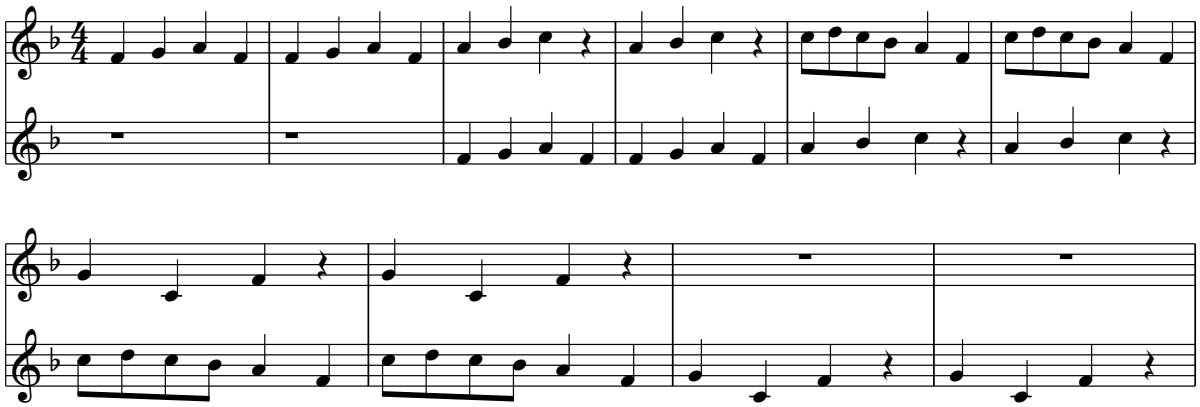
`abcm2ps` allows to typeset music for up to 32 voices, but this limitation can be easily overcome modifying and recompiling the program sources. The number of voices that `abc2svg` can handle only depends on available memory. At most four voices can be typeset on the same staff.

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 can 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

Traditional

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 typeset 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 optional *voice specifiers* to the `V:` field to set voice properties. The list of available voice specifiers depends on the application, which is free to implement its own; further, applications can silently ignore specifiers they don't implement.

`abcm2ps` supports `auto`, `down`, `dyn=`, `gchord=`, `gstem=`, `lyrics=`, `merge`, `name=`, `nm=`, `scale=`, `sname=`, `snm=`, `stafflines=`, `cmdstaffscale=`, `stem=`, and `up`.

`abc2svg` supports `clef=`, `clefpitch=`, `cue=`, `instrument=`, `map=`, `name=`, `nm=`, `octave=`, `pos=`, `scale=`, `score=`, `shift=`, `sname=`, `snm=`, `sound=`, `stafflines=`, `staffnonote=`, `staffscale=`, `stem=`, `subname=`, and `transpose=`.

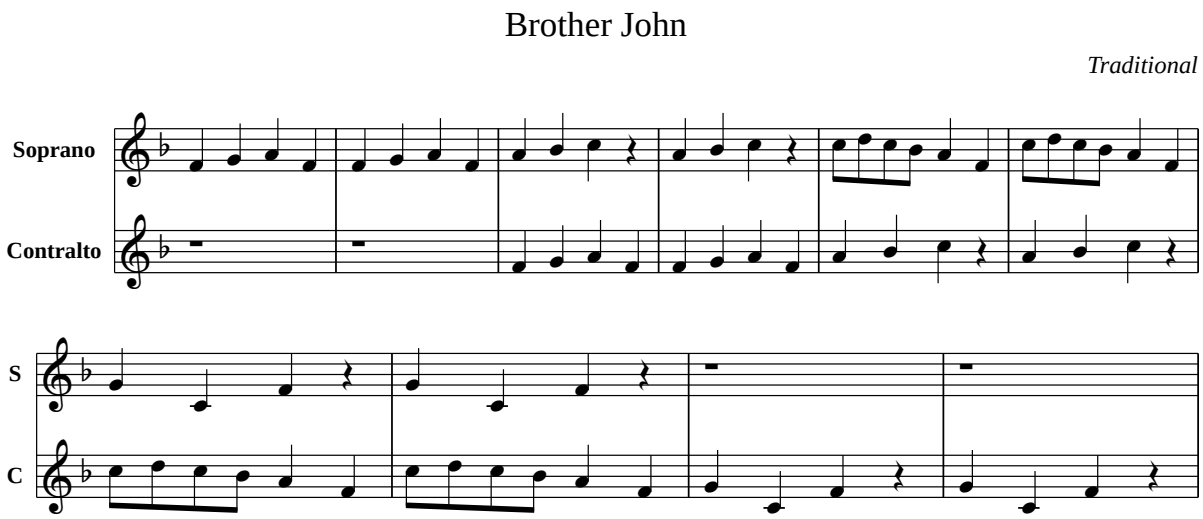
For the moment, let's examine only a few voice specifiers; more will be explained further on.

- `clef=`⟨*clef type*⟩ specifies the voice clef; clefs are the same we examined in Section 2.2.1 (page 24);
- `cue=`⟨*on*⟩ or ⟨*off*⟩ sets the current voice as "cue" (i.e. a voice for hints, which are typeset smaller than other voices) or normal voice;
- `down` forces note stems downwards;
- `dyn=`⟨*up*⟩, ⟨*down*⟩ or ⟨*auto*⟩ forces the placement of dynamics decorations;

- `gchord=`⟨*up*⟩ or ⟨*down*⟩ forces the placement of accompaniment chords;
- `gstem=`⟨*up*⟩, ⟨*down*⟩ or ⟨*auto*⟩ forces the grace note stem direction;
- `lyrics=`⟨*up*⟩, ⟨*down*⟩ or ⟨*auto*⟩ forces the placement of lyrics lines;
- `merge` indicates that this voice belongs to the same staff as the previous voice;
- `name=`⟨*name*⟩ or `nm=`⟨*name*⟩ specifies the name that appears at the left of the first staff;
- `pos=`⟨*type*⟩ ⟨*position*⟩ sets the position of music elements (more information in Section 4.1.13, page 76);
- `scale=`⟨*n*⟩ sets the scale of the voice. Default is 1, maximum value is 2, minimum is 0.5;
- `sname=`⟨*name*⟩ or `snm=`⟨*name*⟩ specifies the name that appears at the left of all the staves after the first one;
- `stafflines=`⟨*n*⟩ sets the number of lines of the associated staff;
- `staffnonote=`⟨*n*⟩: hides the staff associated to the voice (more information in Section 4.1.12, page 73);
- `staffscale=`⟨*n*⟩ sets the scale of the voice and the associated staff. Default is 1, maximum value is 3, minimum is 0.5;
- `stem=` ⟨*up*⟩, ⟨*down*⟩, or ⟨*auto*⟩ forces the note stem direction;
- up forces note stems upwards.

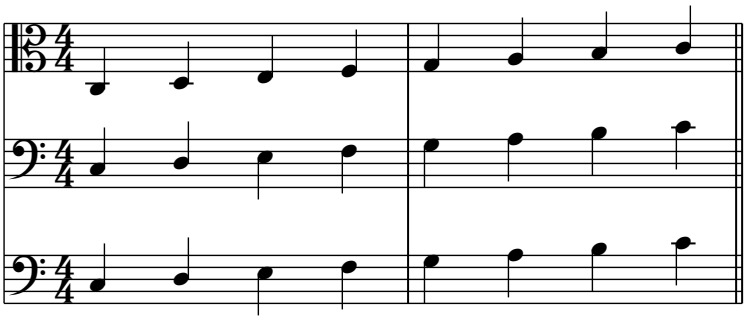
Here is the same tune with some improvements:

```
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 |
```



`V:` can also define the properties of many voices at once. `V:*`, written in the tune header and followed by voice parameters, sets the properties of all voices in the tune; parameters can be redefined by successive `V:` fields. To write the same notes to several voices at once, we can specify the voices in the same `V:` field, separated by commas:

```
X: 1
M: 4/4
L: 1/4
V: * clef=bass  % applies to all voices
K: C
%
V: 1 clef=alto  % overrides previous setting
V: 1,2,3        % specify 3 identical voices
C, D, E, F, | G, A, B, C ||
```

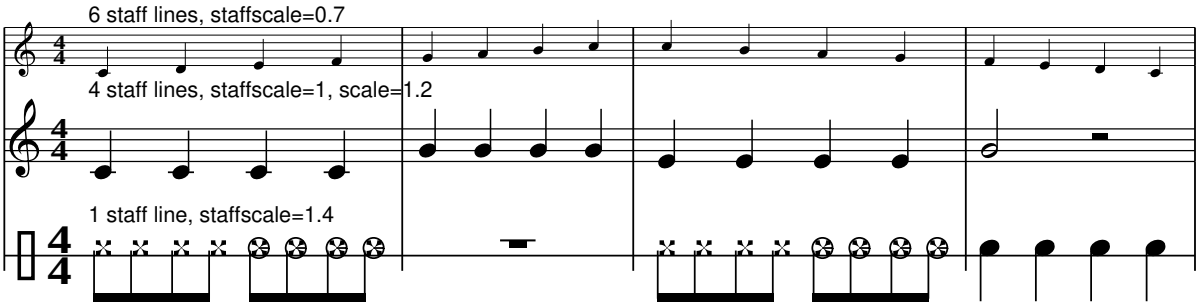


Fancy staves can be obtained using `staffscale` and `stafflines`. Percussion notes are notated using sharps for x-shaped note heads, and flats for circle-x-shaped note heads:

```
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

The above score is typeset with `abcm2ps`, since percussion notes are not supported in `abc2svg`. To be more precise, `abc2svg` implements percussion notes using an extension called *note mapping* that will be explained in Section 8.3 (page 125).

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 typeset 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 typeset on the same staff. For example: `%%score [(SA) (TB)]`

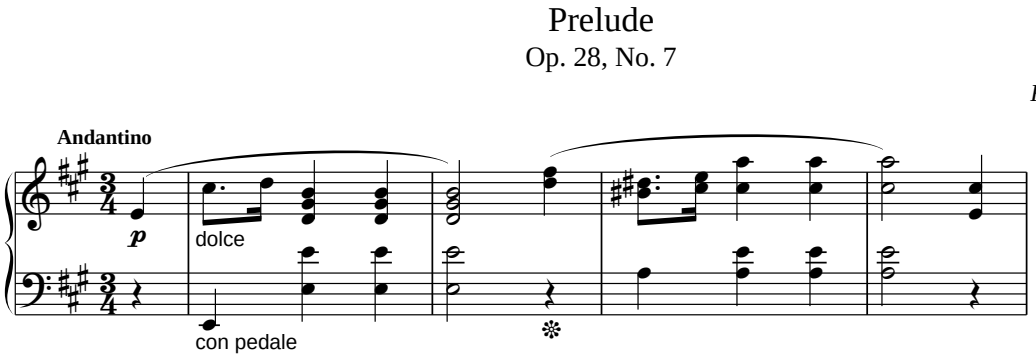
- to add measure bars that cross the staves, use the character | between voice names: %%score [S|A|T|B]

When two voices are typeset 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 right hand. When one of these voices is silent, normal rests are replaced by invisible rests we studied in Section 2.1.4 (page 18).

```
X: 1
T: Prelude
T: Op. 28, No. 7
C: Fryderyk Chopin
L: 1/4
M: 3/4
Q: "Andantino"
%%score (V1 V2) | V3
V: V1 clef=treble
V: V2 clef=treble
V: V3 clef=bass
K: Amaj
%
% 1 - 2
[V:V1] (!p!E | "_dolce"c/>d/ [DGB] [DGB]    | [DGB]2) ([df] |
[V:V2] x     | x3                           | x3            |
[V:V3] z     | "_con pedale"E,, [E,E] [E,E] | [E,E]2 z      |
% 3 - 4
[V:V1] [^B/^d/]>[c/e/] [ca] [ca] | [ca]2) ([Ec] | % slur not ended!
[V:V2] x3                        | x3           |
[V:V3] A, [A,E] [A,E]            | [A,E]2 z     |
```



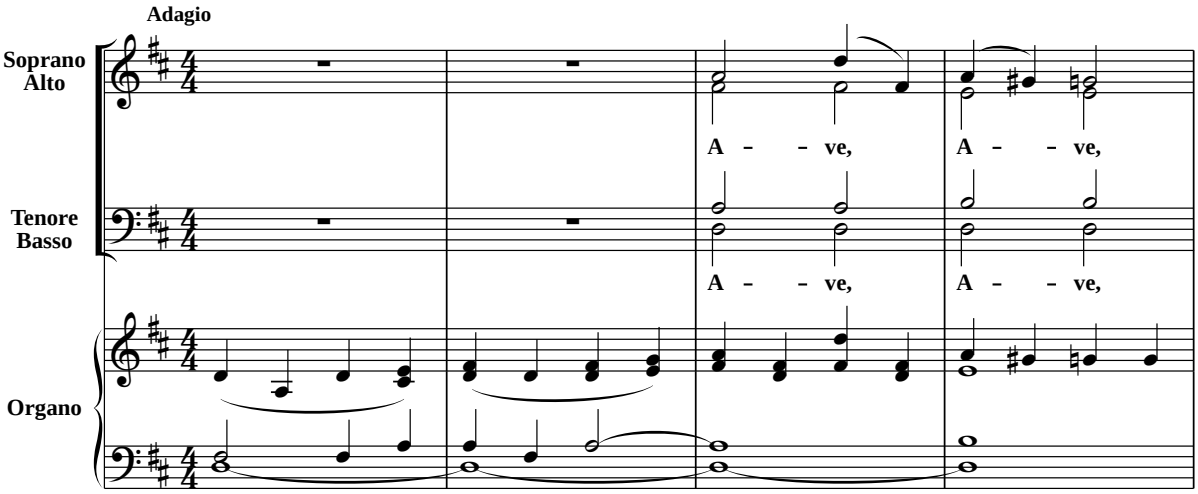
Let's now try a more complex piece. These are the first four measures of Mozart's famous "Ave Verum Corpus", for organ and SATB:

```
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,
```

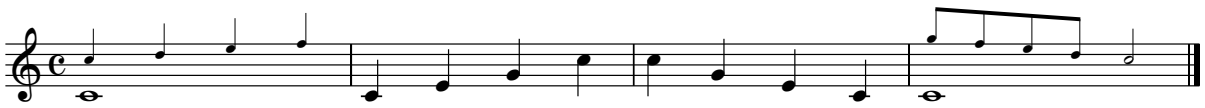
Ave Verum Corpus

W. A. Mozart

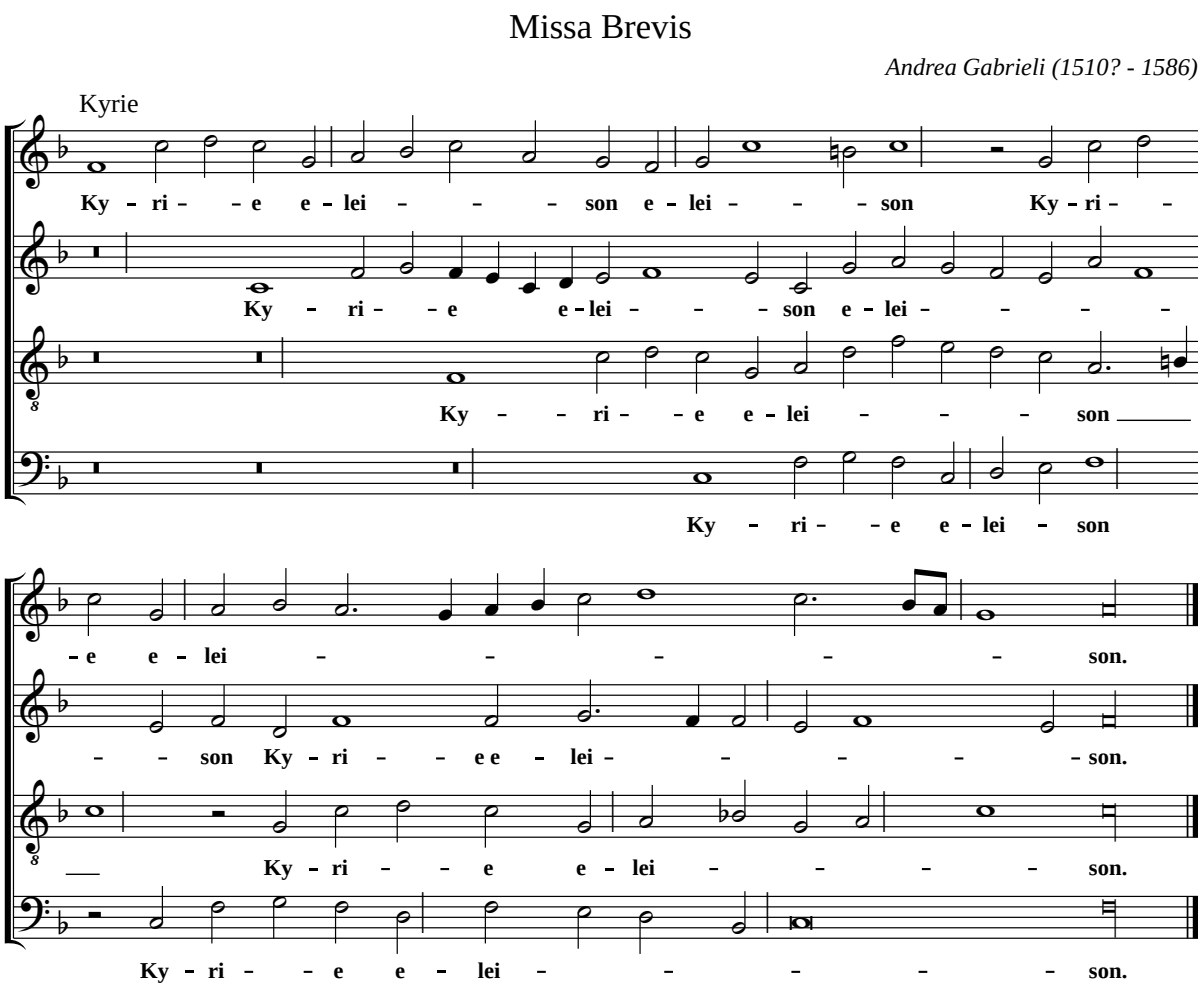


Note that the voices were intentionally written in reverse order, but `%%score` rearranged them as expected. Normally, for clarity sake voices should be written in the same order as specified in `%%score`.

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.
```



3.1.3 Typesetting 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 typeset 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 typeset 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:

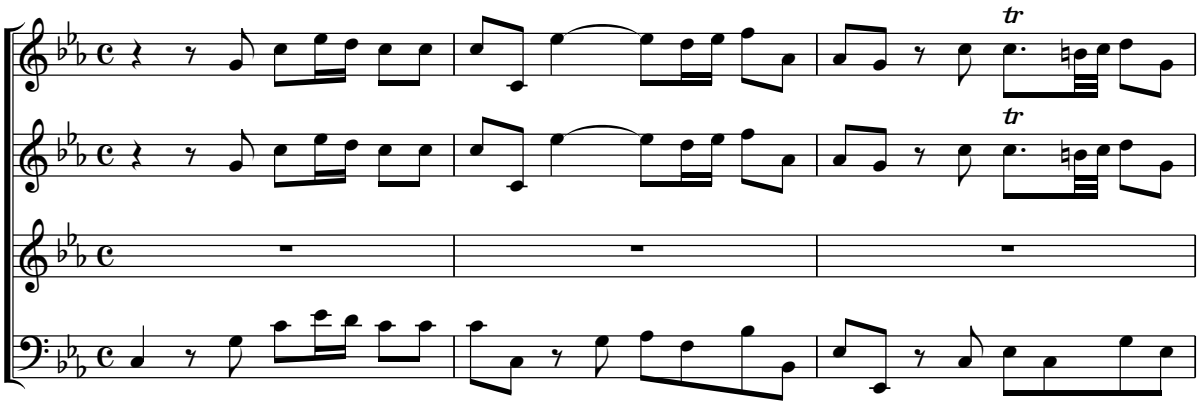
```
T: Full score
%%score [V1 V2 Sesto Bassi]
```

are uncommented. The resulting full score is:

Giulio Cesare in Egitto

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

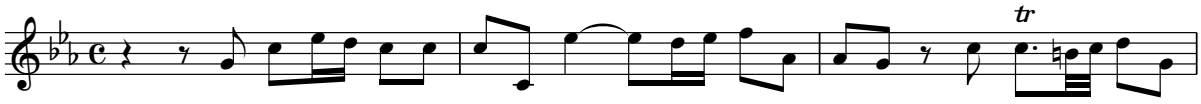
Full score

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:

Giulio Cesare in Egitto
Sesto: Svegliatevi nel core, furie d'un alma offesa (excerpt)
First violin only

G.F. Händel

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

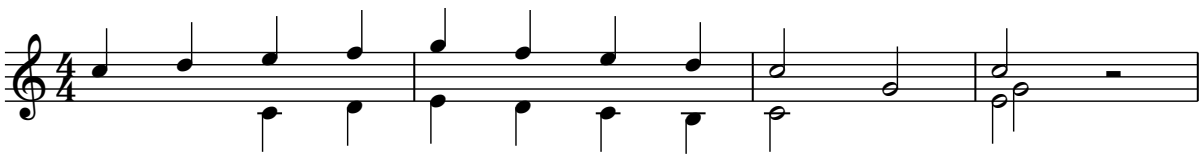
```
%  T: Full score
%  %%score [V1 V2 Sesto Bassi]
T: Vocal part and keyboard reduction
%%score Sesto {V1 Bassi}
%  T: First violin only
%  %%score V1
```

3.1.4 Voice Overlay: &

In some pieces of music, we occasionally need isolated notes (courtesy notes) or short runs of notes belonging to a temporary extra voice. To avoid introducing an additional voice that would be mostly void, we can use the `&` symbol that starts a new temporary 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 `&`. Unfortunately, there is no way to control stem direction in overlay notes.

```
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;
- use spacing to make bars stand out.

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

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 `%%linebreak` or `%%score`. There are several types of directives: some specify the page layout, others the fonts, the spacing, many types of customisation, etc. We will see in Chapter 7 (page 105) that `abc-2midi` 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; for more details, please see Section A.5, page 165.

Unfortunately, at this point Abc applications start to become incompatible with one another. In particular:

- `abc2svg` and `abcm2ps` share most directives, but some directives are specific to each program;
- `abc2svg` and `abcm2ps` use different fonts;
- The Abc/MusicXML translators only support a small subset of the directives supported by `abc2svg`, `abcm2ps`, and `abcMIDI`;
- other Abc applications may ignore directives altogether, or support their own.

As a result, *this chapter is specific to `abc2svg`* and, in some cases, to `abcm2ps`. 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.

4.1.1 How Directives Work

First of all, we examine an important difference between `abcm2ps` and `abc2svg` concerning their output type.

`abcm2ps` produces *page-oriented output*; that is, music elements are written to a page that has definite geometry and layout (width, height, spacings, headers, footers, etc.). This is called *page mode*. `abc2svg`, by default, typesets music elements as SVG images embedded in HTML pages; this is called *block mode*. To enable page mode in `abc2svg`, we must must specify the page height as shown below. Some directives work in page mode, others in block mode.

Let's now see how directives work. Many directives accept a *parameter*, optionally followed by a type specifier, of one of these types:

- a *unit of length* in centimeters (cm), inches (in), PostScript points (pt), or pixels (no suffix): for instance, `30pt`, `1cm`, `0.3in`, `20`. A PostScript point is 1/72 inch or 0.353 mm; 1 pixel is 1/96 inch or 0.265 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-Bold 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.)

Most directives have a default value. For instance, `abcm2ps` uses ANSI letter paper size by default; page width is 8.5 inches, page height is 11 inches. In `abc2svg`, only the page width is predefined, and it defaults to 21 cm (A4 paper width). For reference: A4 paper is 210×297 mm, 595×842 points; letter paper: 8.50×11.00 inches, 612×792 points.

Both programs 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 directives:

```
% 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      % typeset breve note as square glyph
```

As you can see, many directives have a self-explanatory name such as `%%pagewidth` or `%%leftmargin`, while others need explanations. The most important directives will be described in the next few sections.

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`.

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 `abc2svg` and `abcm2ps` options in the command line. For instance, the following command line typesets music with `abcm2ps`, 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.

4.1.2 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` in the browser editor, 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, in the web editor). 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 8 (page 121). Format files containing libraries of symbols can thus be employed when needed.

4.1.3 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.

4.1.4 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:

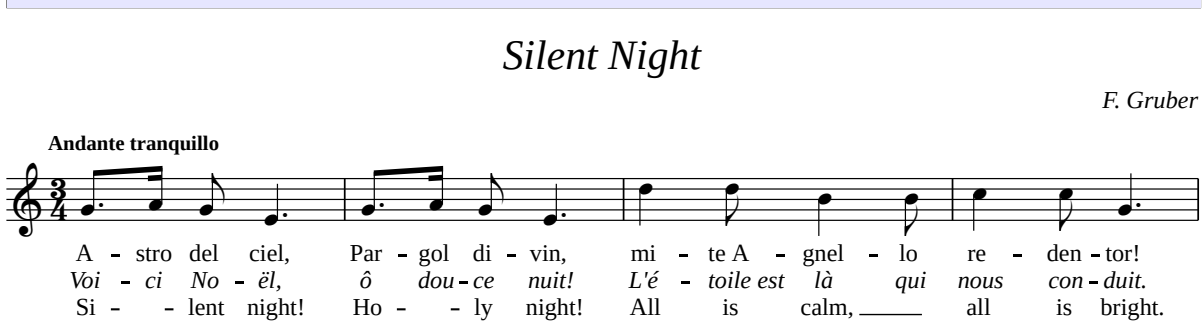
```
I:vocalfont Times-Roman 12
```

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

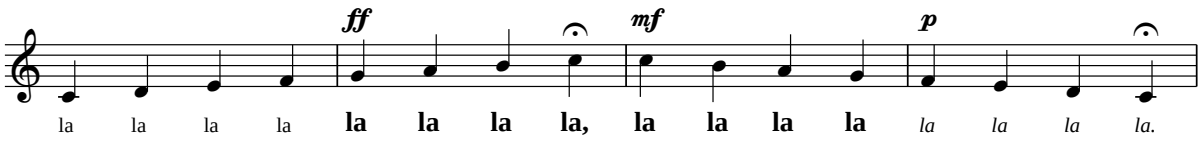
A parameter change can be followed by the `lock` keyword. This means that any following change of the same parameter will not take effect, unless the `lock` is specified again.

> ✍ **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` directive in a tune; this directive changes the font of lyrics in `w:` lines. Since it has a "generation" scope, the change is immediate:

```
%%titlefont Times-Italic 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 14
w: A- stro del ciel, Par- gol di- vin,
+: mi- te~A- gnel- lo re- den- tor!
I: vocalfont Times-Italic 14
w: Voi- ci No- \"el, \^o dou- ce nuit!
+: L'\'e- toile~est l\`a qui nous con- duit.
%%vocalfont Times-Roman 14
w: Si - lent night! Ho - ly night! All is calm,_ all is bright.
```

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

```
X: 1
L: 1/4
K: C
CDEF | !ff!GAB!fermata!c | !mf!cBAG | !p!FED!fermata!C |
%%vocalfont Times-Roman 12
w: la la la la\
I: vocalfont Times-Bold 16
+: la la la la, la la la la\
%%vocalfont Times-Italic 12
+: la la la la.
```



A better way to change fonts will be explained in the following sections.

4.1.5 Using Fonts and Text

Let me stress once again that *text and fonts are a source of major incompatibility between `abcm2ps` and `abc2svg`*. These programs use different fonts by default, and typeset text differently.

Table 4.1 shows a subset of standard fonts that are shared by both `abcm2ps` and `abc2svg`. These fonts are functionally equivalent, even though their names and overall looks are different. These are the so-called "web-safe fonts".

`abc2svg` understands PostScript font names; that is, if you select "Times-Roman", `abc2svg` will use the corresponding SVG font, "serif". `abcm2ps`, however, does not understand all SVG font names. Hence, using PostScript font names is preferable, as both programs understand them.

`abcm2ps` also supports all others PostScript fonts, which are listed in Appendix A.6 (page 179). More fonts can be added to both programs, as we will see later on.

PostScript fonts (`abcm2ps`)	SVG fonts (`abc2svg`)
Times-Roman	serif
Times-Bold	**serifBold**
Times-Italic	*serifItalic*
Times-BoldItalic	***serifBoldItalic***
Helvetica	sans-serif
Helvetica-Bold	**sans-serifBold**
Helvetica-Oblique	*sans-serifItalic*
Helvetica-BoldOblique	***sans-serifBoldItalic***
`Courier`	`monospace`
`Courier-Bold`	**`monospaceBold`**
`Courier-Oblique`	*`monospaceItalic`*
`Courier-BoldOblique`	***`monospaceBoldItalic`***

Table 4.1: Default fonts for `abcm2ps` and `abc2svg`.

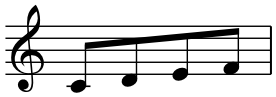
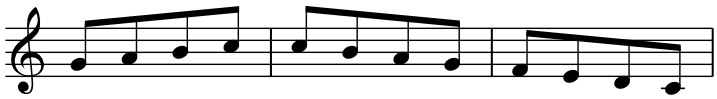
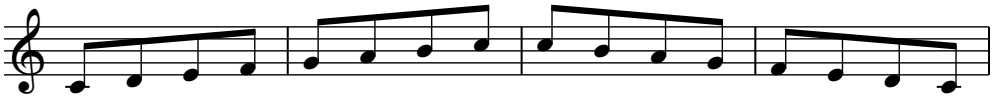
These are predefined fonts, which we can use anytime with any font directive. To use other fonts, we have to *declare* them inserting the `%%font` directive at the top of the source. Text and font related directives are:

- `%%font` *declares* a font, i.e. it tells the applications that the font will be used. If the font name contains spaces, then it must be enclosed in double quotes;
- `%%textfont` sets a font. If the font is not one of the web-safe fonts, it must be declared beforehand with `%%font`;

- %%text typesets a line of text;
- %%begintext...%%endtext define a range of text lines to be typeset;
- %%center centers the following text;
- %%vskip inserts the specified vertical space;
- %%sep inserts a short separator.

%%text lines are treated differently by abcm2ps and abc2svg. In abcm2ps, a %%text line stops the music generation, and the text is typeset immediately. In abc2svg, the text is typeset after the current music line. So, the following source produces different output:

```
X:1
%%linebreak <none>
%%textfont Times-Bold 14
M:none
K:C
CDEF |
%%text Text within music lines
GABc | cBAG | FEDC |
```

Typesetting with abcm2ps:

Text within music lines

Typesetting with abc2svg:

Text within music lines

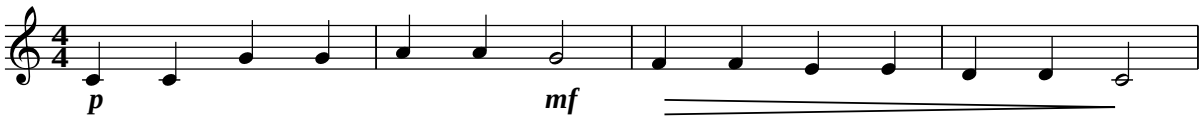
Here is an example that shows abcm2ps's capability of alternating text in different fonts with pieces of music, using one of its additional PostScript fonts (AvantGarde-Book). We can obtain similar (but not identical!) results using abc2svg and other fonts:

```
% declare non-predefined fonts
%%font AvantGarde-Book
%%musicspace -1cm
%%textfont Helvetica 18
%%center Typesetting example
%%vskip 0.4cm
%%textfont Times-Roman 16
%%begintext justify
This is an example of text inserted into an ABC file. This abcm2ps
```

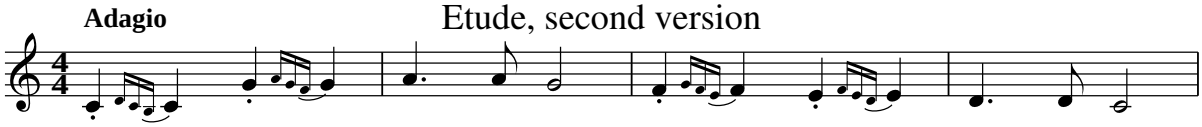
```
feature allows for the writing of songbooks, music collections or other
publications without having to resort to a word processor. Not bad, is
it? Now let's write a brief musical example.
%%endtext
X: 1
M: 4/4
L: 1/4
K: C
%
!p!CCGG | AA!mf!G2 | !diminuendo(!FFEE | DD!diminuendo)!C2 |

%%vskip 0.4cm
%%textfont AvantGarde-Book 16
%%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, 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, no font change
%%textfont * 18
%%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.



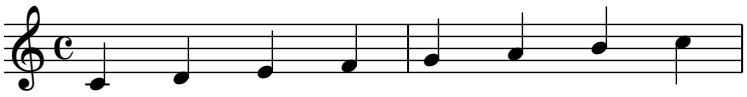
Now we'll have a look at something more lively. To start with, let's switch fonts: from Times-Roman to AvantGarde-Book. Here is the same Etude with a few small variations to make it more interesting:



End of the example.

To use different fonts in strings, 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. $0 reverts to the default font.

```
%%setfont-1 Helvetica      24
%%setfont-2 Times-Italic   22
%%setfont-3 Helvetica-Bold 18
%%setfont-4 Courier        20
X: 1
M: C
L: 1/4
K: C
%%begintext
Hello. This is the default font, $1this is font 1, $2this is font 2,
$3this is font 3, $4this is font 4,$0 and now back to the default font.
%%endtext
CDEF | GABc |
W: It also works in $3W: fields!
W: $0It's useful to $3emphasise $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 back to the default font.

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

All font directives and their default values are listed in Section A.5.3, page 168.

> ✍ **Note**
> A couple of notes:
>
> - if you use a font that is not recognised, it will be silently replaced by Courier (abcm2ps) or serif (abc2svg);
> - font setting directives have an extended syntax that will be explained later on;

4.1.6 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.

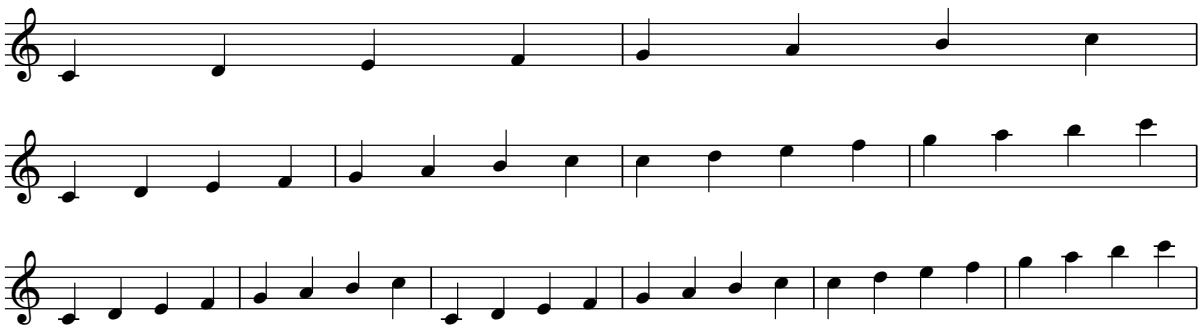
If we don't use the -c option of abcm2ps, we can force line breaks defining one or more line-breaking characters and keywords using the %%linebreak directive:

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

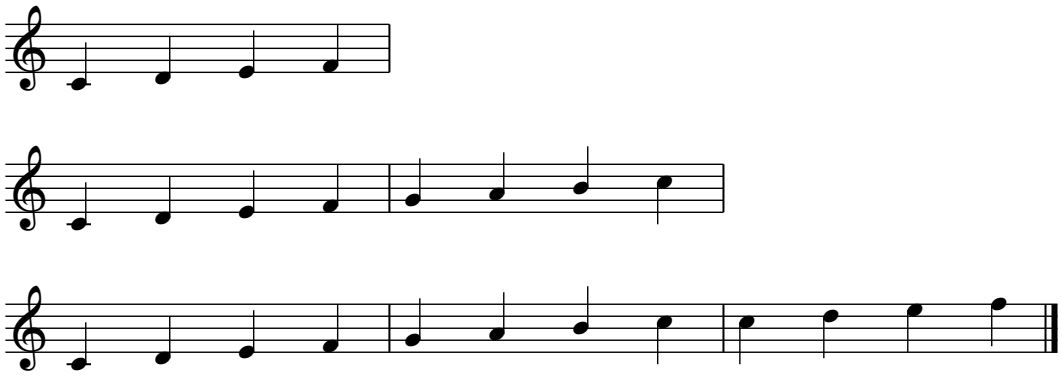
- the <none> keyword causes line breaks to occur automatically, and makes all line-breaking characters invalid;
- the <EOL> keyword causes line breaks to occur at end of lines, and makes all line-breaking characters invalid except `!`;
- %%linebreak with no parameters corresponds to %%linebreak <none>.

> **Avoid mistakes!**
> The `!` line-breaking character was widely used in older ABC releases, but its use is now discouraged.

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|]
```



Another way to insert a line break is the %%vskip directive, which is normally used to insert some vertical space. The above tune could be written as follows:

```
X: 1
L: 1/4
K: C
CDEF|
% 0-height vertical skip
%%vskip 0
CDEF|GABc|
%%vskip 0
CDEF|GABc|cdef|]
```

To avoid ugly results, forcing line and staff breaks should be done with special care.

If a tune doesn't fit in the current page, it is typeset 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.

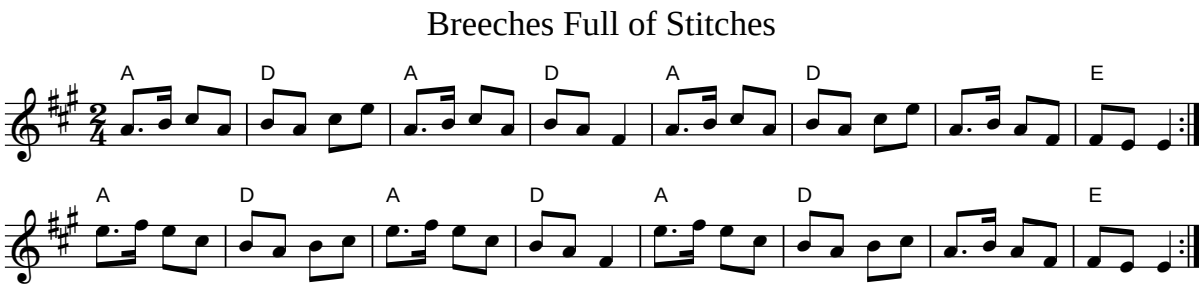
Finally, to force a new page we use the aptly named %%newpage directive.

4.1.7 Avoiding Line Breaks

By default, $\langle n \rangle$ measures in a source line produce $\langle n \rangle$ 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". \ should be preceded by one or more spaces.

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 :|
```

To avoid line breaks entirely, even when line breaking characters are written in the source, we use the `%%linebreak <none>` command. It is the equivalent of the `-c` option; it also works in the command line with `abc2svg` (please see Section 10.2, page 148.)

To typeset the above tune without line breakes, we will the following command:

```
$ abcm2ps --linebreak "<none>" -O= breeches.abc
```

Just like the `-c` option, `%%linebreak <none>` overrides `\` and other line-breaking characters.

4.1.8 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, page 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`;
- `%%equalbars` ⟨*bool*⟩ makes the width of measures proportional to their duration; only available in `abc2svg`.

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.

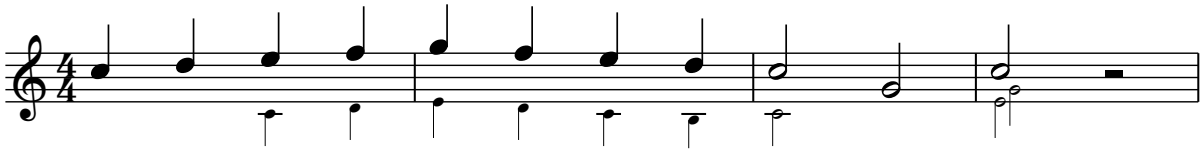
If you decide to set the number of measures yourself, be careful not to write too many or too few per line! If you write too few, the score will look ugly; if you write too many, `abcm2ps` or `abc2svg` will rework the lines at their discretion.

4.1.9 Voice Size

In addition to the `scale` parameter in `V:` definitions (Section 3.1.1, page 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 |
```



4.1.10 Repeated Sequences (*Simile*)

Sometimes, a tune contains repeated runs of notes or measures. Instead of writing the same sequence multiple times, we can use 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 $\langle n \rangle$ and $\langle k \rangle$: 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:

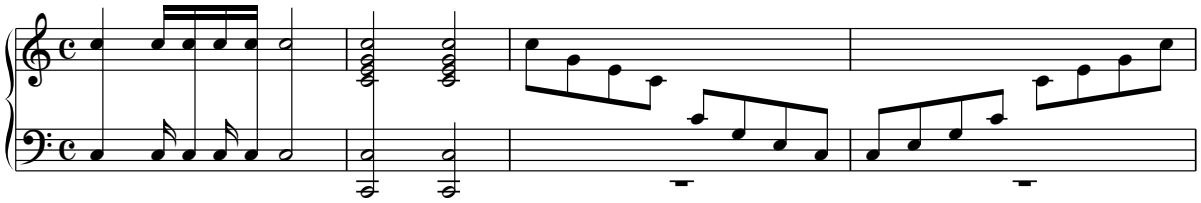
```
X: 1
M: 4/4
L: 1/4
K: C
%
% let's repeat a single note three times
C [I:repeat 1 3] C C C |
% let's repeat a sequence of four notes twice
C//E//G//E// [I:repeat 4 2] C//E//G//E// C//E//G//E// C |
% let's repeat a whole measure once
CEGc | [I:repeat] CEGc |]
```

4.1.11 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`.

4.1.12 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 typeset 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*⟩ typesets or hides voices according to its parameter:

- if ⟨*n*⟩ = 0, only staves that contain visible notes are typeset;
- if ⟨*n*⟩ = 1, only staves that contain visible noted or visible rests are typeset;
- if ⟨*n*⟩ = 2, all staves are typeset.

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 typeset the active voice only, and hide voices that contain only 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 |
```

Blank Sheet Music

We can use %%staffnonote to typeset sheet music (i.e. only blank staves) using this code:

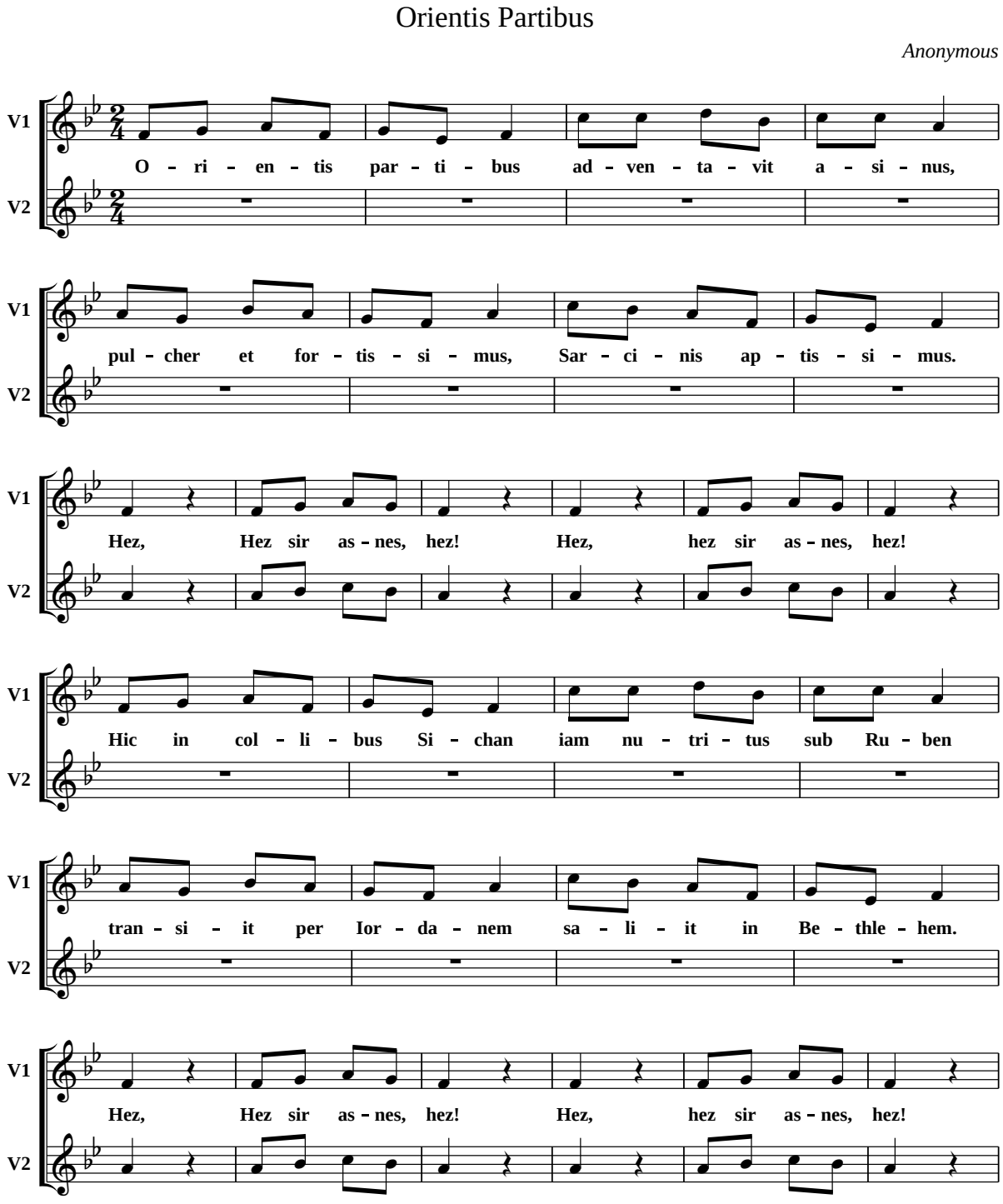



Figure 4.1: A piece with a variable number of systems.

```
X: 1
M: none
L: 1/4
K: C clef=none
%
%%staffnonote   2
%%stretchstaff  1
%%stretchlast   1
%%linebreak     *
%%linewarn      0
% 12 blank staves
x4 [] * x4 [] * x4 [] * x4 [] * x4 [] * x4 [] *
x4 [] * x4 [] * x4 [] * x4 [] * x4 [] * x4 [] *
```

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

4.1.13 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.);
- `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.).

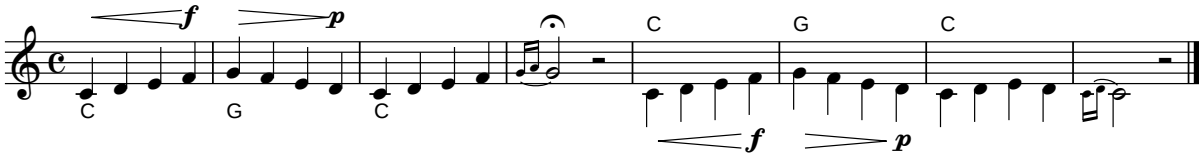
Parameter ⟨*position*⟩ can be one of the following:

- `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 |]
```

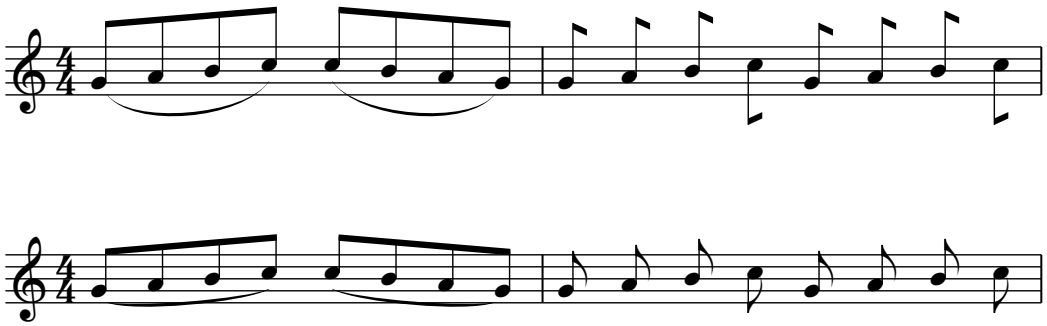


4.1.14 Customising Slurs, Beams, Stems, and Flags

The appearance of slurs, note beams, stems, and flags can be further customised. The `%%flatbeams` 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 |
```

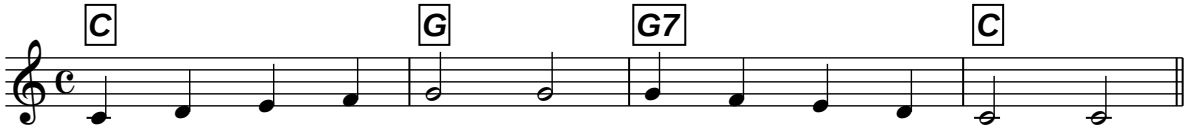


These parameters are applied once, and cannot change midtune. If we want to change their value, we will have to start a new tune, or insert a %%vskip command.

4.1.15 Customising Accompaniment Chords

To make accompaniment chords more visible, you may want to change their font and/or highlight the chords with a box. The relevant directives are %%gchordfont and %%gchordbox:

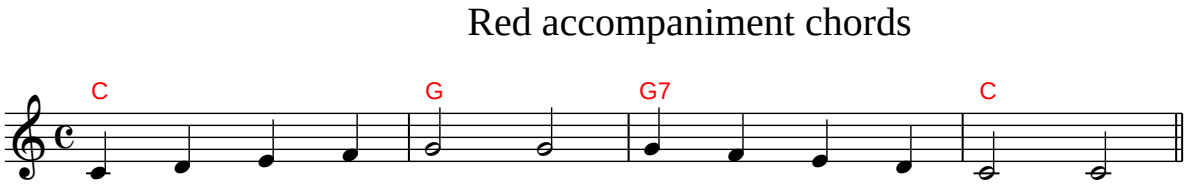
```
%%gchordbox 1
%%gchordfont sans-serifBoldItalic 16

X: 1
L: 1/4
M: C
K: C
%
"C"CDEF | "G"G2 G2 | "G7"GFED | "C"C2 C2 ||
```



Another way to highlight accompaniment chords is typeset them in a different colour. The following tune uses low-level code in PostScript and SVG that will be explained in Section 8.2, page 122:

```
% postscript code for abcm2ps
%%ps /gcshow { 1 0 0 setrgbcolor show 0 0 0 setrgbcolor } !

% SVG code for abc2svg
%%beginsvg
<style>
.red {color:red}
</style>
%%endsvg
% use the "red" class in accompaniment chords
%%gchordfont * * class=red
```

```
X:1
T: Red accompaniment chords
L:1/4
M:C
K:C
%
"C"CDEF | "G"G2 G2 | "G7"GFED | "C"C2 C2 ||
```



The %%gchordfont accepts an additional argument that refers to the red class we defined as SVG code.

In PostScript code, colours can be specified as RGB components (real numbers between 0.0 and 1.0) before the setrgbcolor command. In SVG code, you may use HTML colour names or the rgb(R, G, B) command; RGB components are numbers between 0 and 255.

4.1.16 Customising Tuplets

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

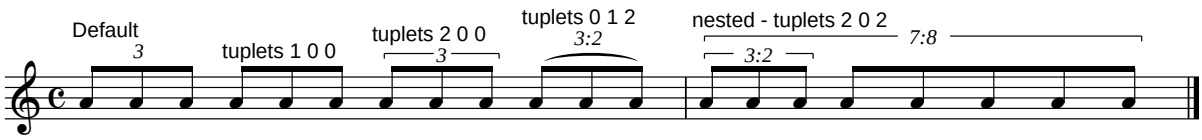
%%tuplets ⟨*where*⟩ ⟨*what*⟩ ⟨*value*⟩

Parameters are:

- where is a number that indicates where to typeset 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 typeset. 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:

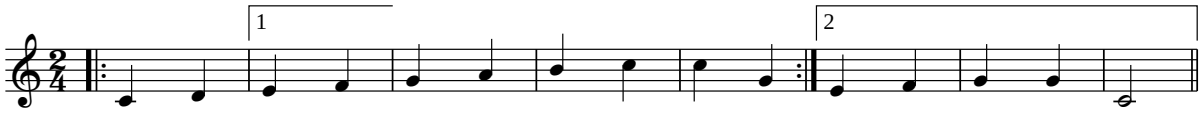
```
X: 1
M: C
L: 1/4
K: none
%
"^Default"(3A/A/A/ "^tuplets 1 0 0"[I:tuplets 1 0 0](3A/A/A/ \
"^tuplets 2 0 0"[I:tuplets 2 0 0](3A/A/A/ \
"^tuplets 0 1 2"[I:tuplets 0 1 2](3A/A/A/ |\
"^nested - tuplets 2 0 2"[I:tuplets 2 0 2](7:8:8(3A/A/A/ A/A/A/A/A/ | z4]
```



4.1.17 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:

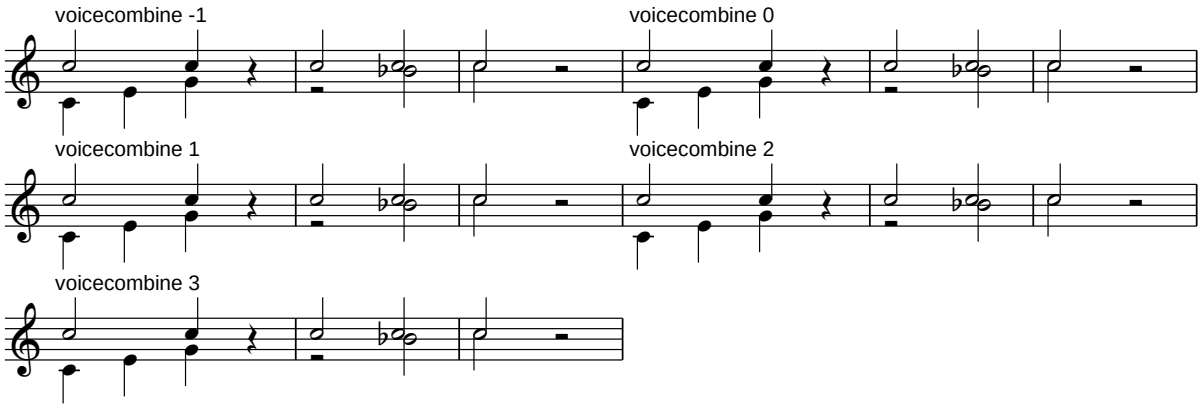
```
X: 1
M: 2/4
L: 1/4
K: C
%%rbmax 3
%%rbmin 1
|:CD |1 EF | GA | Bc | cG :|2 EF | GG | C2 ||
```

4.1.18 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 typeset. The %%voicecombine directive is followed by a parameter that produces different effects:

- -1: notes and rests are typeset 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 |
```



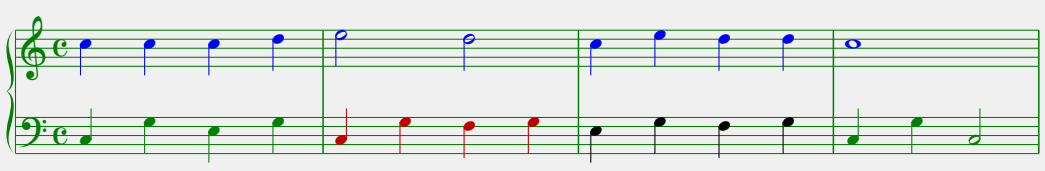
4.1.19 Colours

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. Alternatively, the colour can be specified using a valid HTML 4 colour name: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

In addition to voice colours, `abc2svg` also provides the `%%bgcolor` and `%%fgcolor` directives to set the page background and foreground colours.

```
%%bgcolor #f0f0f0   % light gray
%%fgcolor #008000   % dark green
X:1
M:C
L:1/4
K:C
%%score {1|2}
%
[V:1] [I:voicecolor blue]cccd | e2d2 | cedd | c4 |
[V:2] C,G,E,G, | [I:voicecolor #c00000] C,G,F,G, \
[I:voicecolor black] | E,G,F,G, | [I:voicecolor #008000] C,G,C,2 |
```



4.1.20 Multi-column Output

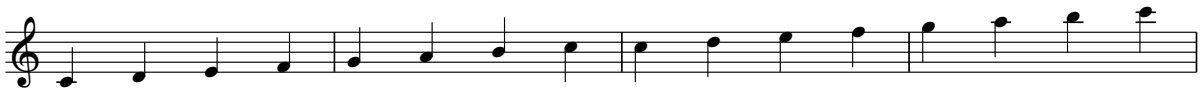
Text and music can be laid out in multiple columns that are defined using the %%multicol start, %%multicol new and %%multicol end directives:

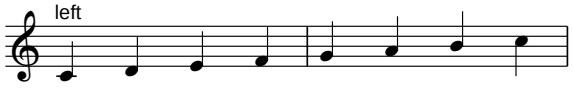
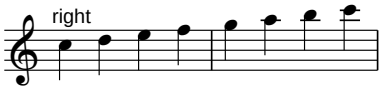
%%multicol start saves the current page margins and sets the vertical position for the beginning of a column. At this point we can change the margins and typeset the material in the first column.

%%multicol new moves the vertical position to the beginning of a new column, resetting the margins. Change the margins again and typeset 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 typeset.

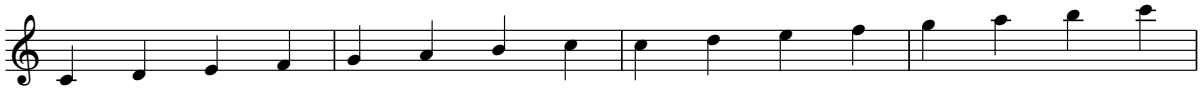
It sounds difficult, but it's actually quite easy as shown in the following example:

```
%%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

4.1.21 Customising Titles

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

Complex titles are specified using `%%titleformat`, followed by a set of letters, digits, and commas. Letters refer to ABC fields, and may be anything in the range `A` … `Z`; the corresponding field will be typeset. Digits may follow a letter, meaning "0" for centre, "1" for right align, or "-1" for left align. A comma `,` forces a newline; unrecognized characters are ignored.

```
%%writefields X
%%titleformat R-1 T0 C1, Q-1 T0 O1, T0 H1
X: 123
T: Main Title
T: Second Title
T: Last Title, The
C: Anonymous
H: Written many many years ago...
L: 1/4
M: 4/4
R: march
O: Scotland
Q: 1/4 = 108
K: C
CDEF | GABc | CDEF | GABc |]
```

march

♩ = **108**

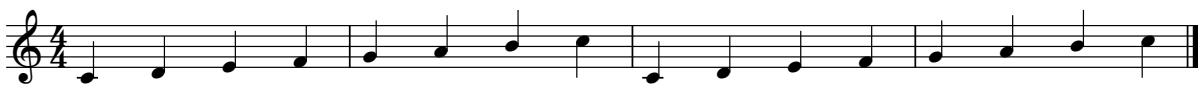
123. Main Title

Second Title

The Last Title

Anonymous

Scotland

Written many many years ago...

First of all: the `%%writefields` command, followed by `X`, typesets the tune index number before the first title.

Let's examine the %%titleformat line. The first part, R-1 T0 C1, typesets the rhythm (R:) to the left, the main title at the centre, the composer to the right, and starts a new line. Q-1 T0 O1, typesets the tempo to the left, the second title at the centre, the origin to the right, and starts a new line. Finally, T0 H1 typesets the third title at the centre and the history (H:) to the right.

Note the strange third title. As you can see, it was rearranged to read "The Last Title". Whenever abcm2ps or abc2svg find 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.

Avoid mistakes!

Do not write the X: number in the tune title; it's a very bad practice. Sooner or later, you may need to rearrange the index numbers, and you would have to edit the tune(s) manually to fix the title(s)!

4.1.22 Headers and Footers

The following directives define text that should be typeset automatically on all pages: %%header for the page header, and %%footer for the page footer. These directives, followed by text, will typeset it centred on the page by default.

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 typesets date and time of the last modification of the current input file;
- $D typesets the current date and time;
- $F typesets the name of the current file;
- $I⟨*x*⟩ typesets the contents of a field (⟨*x*⟩ is capital letter, from A to Z);
- $P typesets the page number;
- $P0 and $P1 typesets the page number, but only if it is even or odd;
- $T typesets the title of the current tune;
- $V typesets 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, page 7.)

Here is an example of the %%footer directive used to typeset 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!

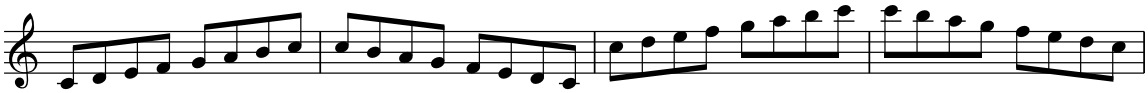
```
%%footer "$P0   $T      $P1"
```

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

4.1.23 Inserting Graphics Files (abcm2ps)

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 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.

To include graphics files with `abc2svg`, please refer to Section 4.1.27 (page 87).

4.1.24 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 typeset a number every ⟨*n*⟩ measures, we use `%%measurenb` ⟨*n*⟩. 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.

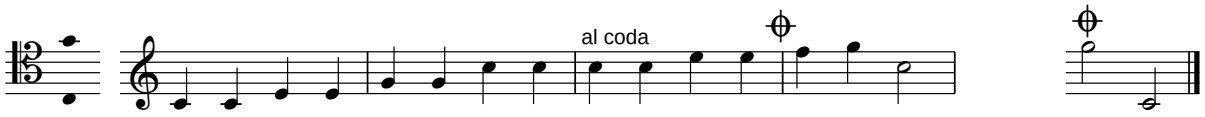
To set the initial bar number as ⟨*n*⟩, use `%%setbarnb` ⟨*n*⟩.

In addition to header and footers (Section 4.1.22, page 84), page numbers can be specified with the option `-N` ⟨*number*⟩ from the `abcm2ps` command line. Possible values are:

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

4.1.25 Staff Gaps

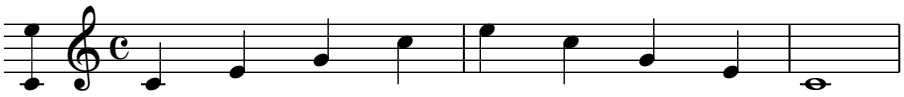
In some pieces, we may want to specify the original key signature and voice extension (*ambitus*) at the beginning; or we may want to write a coda. The staff (or the staves) can be interrupted with the `%%staffbreak` directive. This directive doesn't actually "break" the staff; rather, it makes a gap:

```
X: 1
L: 1/4
K: C alto4
%%setbarnb 0
%
!stemless![C,G]\
%%staffbreak 0.3cm
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 instead of 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.

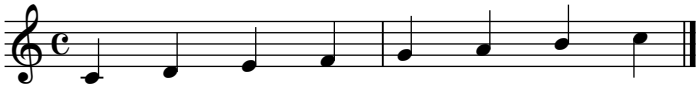
4.1.26 Ambitus (abc2svg)

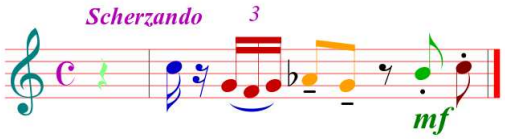
Using abc2svg, writing an ambitus is much simpler. All we have to do is insert the %%ambitus 1 directive, which will automatically establish the ambitus and typeset it before the clef:

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

4.1.27 Embedding XHTML (abc2svg)

When %%pageheight is not set, we can use the %%beginml…%%endml directives to include XHTML code in ABC sources. For example, we can specify web links and images as in the following example:

```
X: 1
T: Using web links
L: 1/4
M: C
K: C
%
CDEF | GABc |]
%%beginml
<style>
.center text-align: center; vertical-align: middle;
</style>
<div class="center">
Please <a href="http://abcplus.sourceforge.net/">follow this link</a>
to find ABC software and documentation.
<br>
It's the ABC Plus site:
<img class="center" src="http://abcplus.sourceforge.net/Scherzando.png"
width=300>
</div>
%%endml
```

Using web links

Please follow this link to find ABC software and documentation.

It's the ABC Plus site: 

Please note that cropping the resulting PDF file with PDFCrop will disable the link, while `pdfcrop.sh` will keep it active.

4.1.28 Saving Space

Quite often, one wants to typeset 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 and ties, use the `%%slurheight` and `%%tieheight` and 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: typesetting 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 Transposition

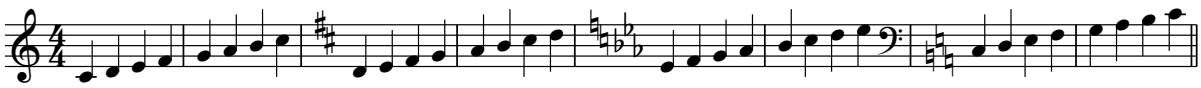
Transposition is a complex subject, and it can be accomplished in different ways. Unfortunately, applications do not use the same syntax; nevertheless, we can transpose both typeset and playback output.

4.2.1 Using %%transpose

The %%transpose directive is supported by abcm2ps and abc2svg, but (as of this writing) not by abc2midi. This directive *transposes typeset output*; MIDI transposition uses other commands, described in Section 7.1.8 (page 110). %%transpose 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 tonic 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
M: 4/4
K: C
%
CDEF | GABc | \ % up two half-tones: from C to D
%%transpose 2
CDEF | GABc | \ % from C to E flat
%%transpose C_E
CDEF | GABc | \ % down one octave
%%transpose cC
CDEF | GABc ||
```

%%transpose is useful to transpose music from concert pitch to another key for a transposing instrument; e.g. for the B♭ clarinet. For example, a tune written in the key of C that includes a %%transpose 2 directive will be typeset as if it were written in the key of D, but the resulting MIDI will sound in C.

4.2.2 K: and V: Modifiers (abc2svg)

abc2svg supports K: or V: transposition modifiers:

- score=⟨*from*⟩⟨*to*⟩ transposes the typeset output only. ⟨*from*⟩ is the original note, ⟨*to*⟩ is the transposed note;
- sound=⟨*from*⟩⟨*to*⟩ transposes the playback only;
- shift=⟨*from*⟩⟨*to*⟩ transposes both the typeset and playback.

These modifiers can also be applied globally when they are written in a V:* field at the top of a file. Our previous example can be rewritten as:

```
X: 1
L: 1/4
M: 4/4
K: C shift=CD
%
CDEF | GABc | CDEF | GABc ||
```

> ✍ **Note**
>
> To transpose an ABC file to a new, transposed ABC file, we use `abc2abc` (Section 10.6, page 155).

Chapter 5

Tablatures and Fingering Charts

abc2svg provides several directives that typeset tabulatures and chord grids for the guitar and other string instruments. abcm2ps does not support these directives, but it can typeset fingering charts for a few wind instruments.

5.1 Tablatures

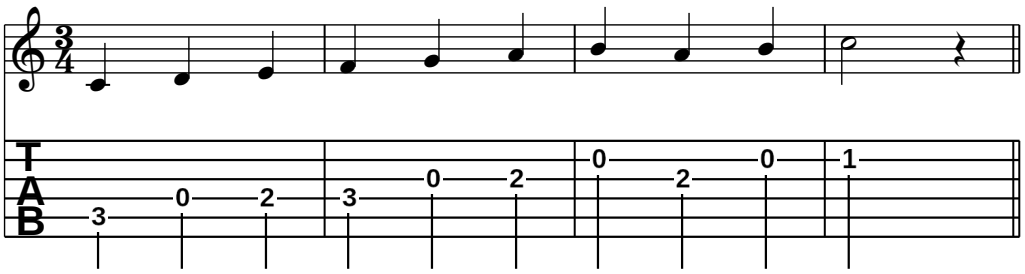
The %%strtab directive is used to typeset a voice as tablature for the guitar or other string instruments. The syntax is:

%%strtab ⟨*string notes*⟩ *[diafret]*

The first parameter is a list of notes that define the instrument tuning; for example, standard guitar tuning is E,A,DGBe. The optional string parameter diafret indicates that the fretboard is diatonic, as in the dulcimer.

Sometimes, tablatures consist of two staves; the top staff is in standard notation, the stave below is the corresponding tablature. We could write two identical voices, but there's a handy shortcut. We can specify two voices in the V: field; notes that follow will be typeset on two staves. Adding %%strtab to the definition of the second voice makes it a tablature:

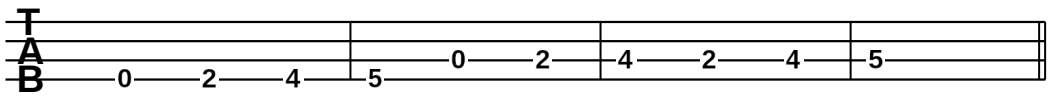
```
X: 1
M: 3/4
L: 1/4
%%score 1 T          % voice order
V: T                 % the second voice is the tablature
%%strtab E,A,DGBe    % standard guitar tuning
K: C
%
V: 1,T               % same notes on both voices
C D E | F G A | B A B | c2 z ||
```



> ✍ **Note**
> Specifying multiple voices in `V:` produces no output in `abcm2ps` and `abc2midi`, and is not supported by `abc2xml.py`.

Let's now write a tablature for the mandolin in standard tuning GDAE; more precisely, `Gdae'`. In this example we typeset only the tablature, and hide note stems using `%%pos`:

```
X:1
M:3/4
L:1/4
V:T
%%strtab Gdae'    % standard mandolin tuning
%%pos stem hidden
K:G
%
V: T
G A B | c d e | f e f | g2 z ||
```



5.2 Chord Grids

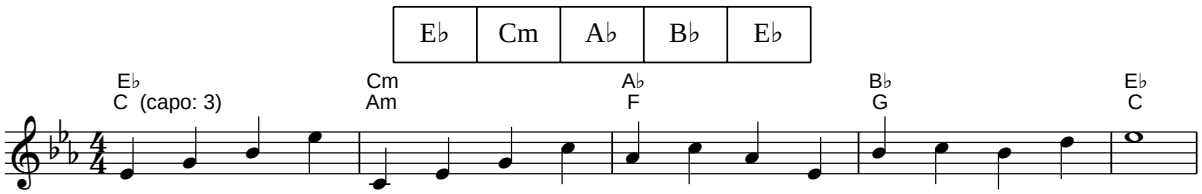
Chord grids are used to typeset a summary of the chords that are played in a song. Two different formats are available:

- `%%grid` ⟨*n*⟩ typeset 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 typeset;

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 the key of E♭, which is difficult to play on the guitar. To play the tune as if it were written in C major, we specify a capo on the third fret. We also typeset the chord grid and the chord diagrams:

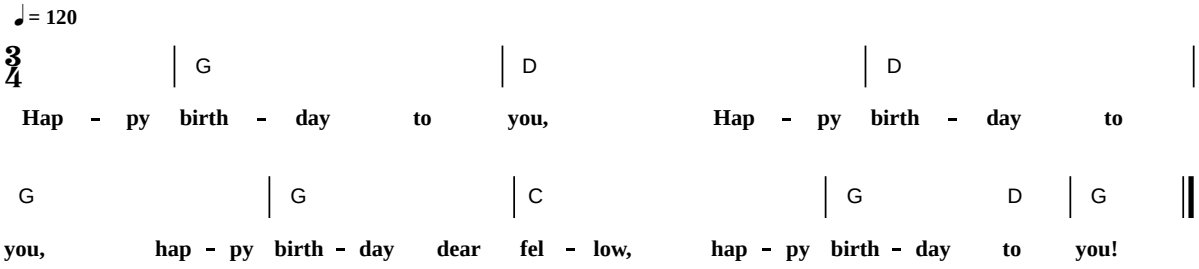
```
X:1
L:1/4
M:4/4
K:Ebmaj
%
%%capo 3
%%grid 1
"Eb"EGBe | "Cm"CEGc | "Ab"AcAE | "Bb"BcBd | "Eb"e4 |
```



The following tune shows %%grid2 in action:

```
X: 1
T: Happy Birthday with %%grid2
M: 3/4
Q: 1/4=120
K: Gmaj
V: 1
%%grid2 1
D> D|  "G"E2D2G2 | "D"F3 z D> D  | "D"E2D2A2    | "G"G3 z D> D |
w:Hap-py birth-day to you, Hap-py birth-day to you, hap-py
"G"d2B2G2 | "C"F2E z c> c | "G"B2G2"D"A2 | "G"G6 |]
w:birth-day dear fel-low, hap-py birth-day to you!
```

Happy Birthday with %%grid2



5.3 Chord Diagrams

Chord diagrams show how to finger a chord on the fretboard; they are commonly used in guitar songbooks. Predefined chord diagrams are available for notes `C, D, E, F, G, A B` (no sharps and flats); supported chord types are major (no suffix), minor (`m`), seventh (`7`), minor seventh (`m7`), major seventh (`M7`), and suspended fourth (`sus4`). Other chord types can be added.

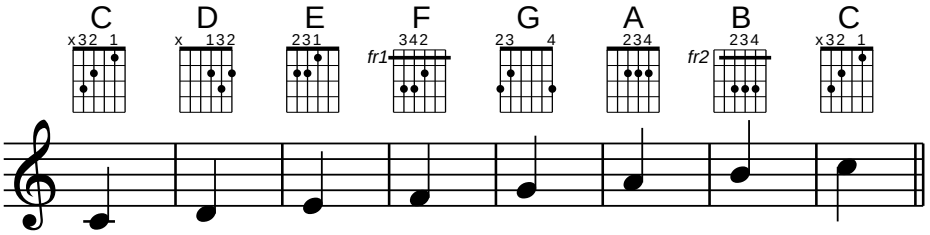
Chord diagrams are defined and typeset with the following directives:

- %%diagram typesets guitar chord diagrams above the notes, in addition to chord names;
- %%setdiag allows to define or redefine guitar chord diagrams.

These are the predefined chord diagrams for `C`:

```
%%pagescale 1.3
%%diagram 6 % show diagrams
X: 1
```

```
M: none
L: 1/4
K: C
%
"C"C | "D"D | "E"E | "F"F | "G"G | "A"A | "B"B | "C"c ||
```



To redefine existing diagrams or add diagrams for other keys or shapes, we use the `%%setdiag` command as follows:

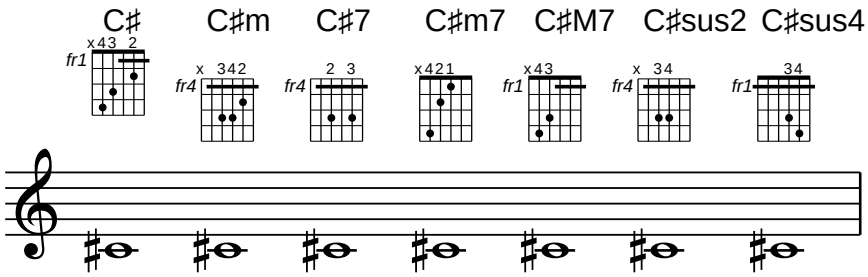
`%%setdiag` ⟨*name*⟩ ⟨*dots*⟩ ⟨*label[,pos]*⟩ ⟨*fingering*⟩ *[barre=⟨n1⟩-⟨n2⟩]*

where:

- ⟨*name*⟩ is the chord name;
- ⟨*dots*⟩ is a list of six fret positions: a digit ⟨*n*⟩ plots a dot on the ⟨*n*⟩th fret, '0' or 'x' means "no dot";
- ⟨*label*⟩ is a text label to be typeset at fret *[pos]*; `,0` means "no label";
- ⟨*fingering*⟩ indicates which finger to use on each fret. 'x' means "mute string", '0' or 'y' means "open string";
- *[barre=⟨n1⟩-⟨n2⟩]* draws a barre between strings ⟨*n1*⟩ and ⟨*n2*⟩. Strings are numbered 654321, from `E,` to `e`.

The following source defines new chord diagrams for the key of C♯:

```
%%pagescale 1.3
%%diagram 1

X: 1
L: 1/1
M: none
K: none
%%setdiag C# 043020 fr1,1 x43020 barre=3-1
%%setdiag C#m x03320 fr4,1 x03420 barre=5-1
%%setdiag C#7 x03030 fr4,1 002030 barre=5-1
%%setdiag C#m7 x42100 ,0 x42100
%%setdiag C#M7 x43000 fr1,1 x43000 barre=3-1
%%setdiag C#sus2 x03300 fr4 x03400 barre=5-1
%%setdiag C#sus4 000340 fr1,1 000340 barre=6-1
"C#"^C "C#m"^C "C#7"^C "C#m7"^C "C#M7"^C "C#sus2"^C "C#sus4"^C |
```



Let's examine how we defined `C♯`. Guitar strings are numbered, right to left, from 1 to 6: 1 is the high `e`, 6 is the low `E,`.

- `%%setdiag C#` defines the chord name;
- `043020`: typesets a dot on string 5 at the 4th fret, on string 4 at the 3rd fret, on string 2 at the 2nd fret. The other strings are not fingered, so 0 means "no dots";
- `fr1,1` typesets the string `fr1` at the 1st fret;
- `x43020` typesets `x` above the 6th string, 4 above the 5th string, 2 above the 2nd string, nothing above the other strings;
- `barre=3-1` draws a barre from the 3rd to the 1st string.

Just as we defined new tablatures with `%%strtab`, we can also define chords for other string instruments with `%%setdiag` specifying fewer or more strings.

5.4 Fingering Charts (`abcm2ps`)

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` provides predefined fingering charts and provides support for user-defined charts. The format file `flute.fmt` contains fingerings definitions for tin whistles in several keys; for the *galoubet*, a French three-hole fipple flute; and of other three-hole instruments like the English pipe (as in "pipe and tabor") and the 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*⟩ typeset under the staff. The parameter ⟨*key*⟩ is a digit corresponding to a key, as outlined in Table 5.1.

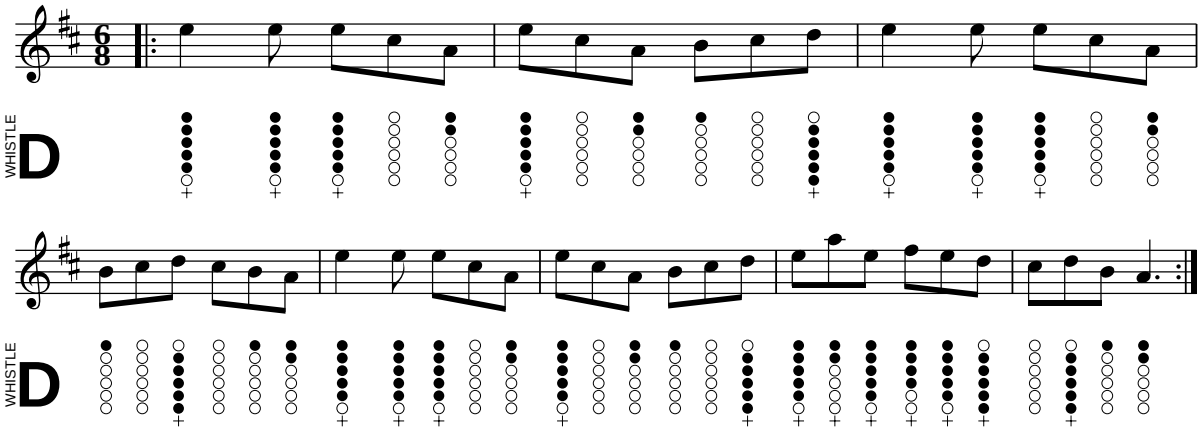
Digit	1	2	3	4	5	6	7	8
Key	D	C	E♭	B♭	F	G	A	C (galoubet)

Table 5.1: 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 6

Tune Collections

We learnt in Section 1.2.4 (page 8) that Abc files may contain several tunes. This feature, along with the ease of use of Abc, spurred the creation of many Abc *tune collections* on the Internet.

The ability to create tune collections is one of the strongest points of Abc, and is not replicated by any other music notation format or application.

6.1 Index Number: *X:*

In the examples examined so far, we dealt with Abc 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 7.1.2, page 106) 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. Please note the `%%writefields` line.

6.2 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/, I suggest that you include at least the `O:`, `R:`, and `D:` fields. Besides, including the `Q:` field to give a speed indication is highly recommended!

6.3 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 typeset 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
```

6.4 Songbooks

The body of a tune may consist of text 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:

```
%%textfont      Courier 14
%%titlefont     Times-Bold 18
%%composerfont  Times-Italic 16
%%titleleft     1
%%aligncomposer -1

X: 131
T: Hey Jude
C: The Beatles
K: Fmaj
%%begintext
    F                   C
Hey Jude don't make it bad
       C7                    F
take a sad song and make it better
Bb                             F
Remember to let her into your heart
```

```
                C               F
and then you can start to make it better
...
%%endtext

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

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

To make sure that chords and lyrics lines are properly aligned, we must set the text font as Courier (monospace). If we want to make the chords more visible, we can typeset them using Courier Bold, which preserves the alignment:

```
%%setfont-1       Courier-Bold 14
%%setfont-2       Courier      14
%%titlefont       Times-Bold 18
%%composerfont    Times-Italic 16
%%titleleft       1
%%aligncomposer   -1

X: 131
T: Hey Jude
C: The Beatles
```

```
K: Fmaj
%
%%begintext
$1    F                  C
$2Hey Jude don't make it bad
$1       C7                   F
$2take a sad song and make it better
$1Bb                            F
$2Remember to let her into your heart
$1                 C               F
$2and then you can start to make it better
...
%%endtext
```

Each song in the songbook will look like this:

Hey Jude

The Beatles

```
    F                   C
Hey Jude don't make it bad
       C7                    F
take a sad song and make it better
Bb                             F
Remember to let her into your heart
                  C                F
and then you can start to make it better
...
```

You may wonder why one should write songbooks using ABC instead of plain text or ad-hoc software; the following section explains why.

6.5 Selecting Tunes

An incredibly useful feature of `abcm2ps` and `abc2svg` 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 typeset:

```
%%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-8.14.12 (2021-07-14)
```

```
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-8.14.12 (2021-07-14)
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!

6.6 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 typeset.

So, if we want to typeset 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 typeset only the measures included between the first and the fifth measure bars.

To typeset 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 7

Playing

7.1 MIDI Conversion

In addition to printable output, Abc can also produce playback files; 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 Abc 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 Abc source files;
- `midi2abc`: conversion of MIDI files to Abc;
- `yaps`: a command-line formatter analogous to `abcm2ps`, but less powerful.

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

7.1.1 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.

7.1.2 Our First Midi

Do you remember the first scale and the command-line instructions we examined in Section 1.2.4 (page 8)? We wrote a very simple ABC source and turned it into PostScript using `abcm2ps` in 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 in the command line:

```
timidity scale11.mid
```

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

7.1.4 Supported Decorations

Some of the decorations described in Section 2.2.11 (page 40) are supported by `abc2midi` and produce playback:

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

Default volume is equivalent to `!f!`.

7.1.5 `%%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.

7.1.6 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 4.1.5, page 65). 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 typeset by abcm2ps,
  but will not be converted to MIDI.
%%endtext
%%MidiOn
...
```

7.1.7 Voices and Instruments

Let's consider "Ave Verum Corpus", which we examined in Section 3.1.2 (page 49). 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.8 (page 183). 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.

7.1.8 MIDI Transposition

`abc2midi` does not support the transposition syntax we studied in Section 4.2 (page 88), and provides its own:

- the `transpose=`⟨*n*⟩ modifier, applicable to `K:` and `V:` fields;
- the `%%MIDI transpose`⟨*n*⟩ and `%%MIDI rtranspose`⟨*n*⟩ directives.

The following tune uses `shift` to let `abc2svg` transpose its output by two semitones, and adds `%%MIDI transpose 2` to transpose the resulting MIDI file:

```
X: 1
L: 1/4
M: 4/4
K: C shift=CD
%%MIDI transpose 2
CDEF | GABc | CDEF | GABc ||
```

`%%MIDI transpose`⟨*n1*⟩ followed by `%%MIDI rtranspose`⟨*n2*⟩ will perform transposition of ⟨*n1+n2*⟩ half-tones.

7.1.9 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.

7.1.10 Accompaniment Chords

The accompaniment chords described in Section 2.2.9 (page 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
```

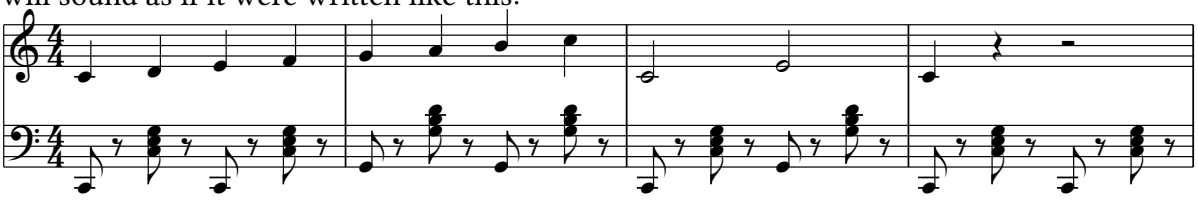
Additional chords can be defined with `%%MIDI chordname`, explained in Section 7.1.11 (page 113).

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: garota.mp3.

7.1.11 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 7.1.10 (page 111), ⟨*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.

7.1.12 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`.

7.1.13 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 $\langle n \rangle$.

7.1.14 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.

> ✍ **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.

7.1.15 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.8.2, page 184) 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]|
```

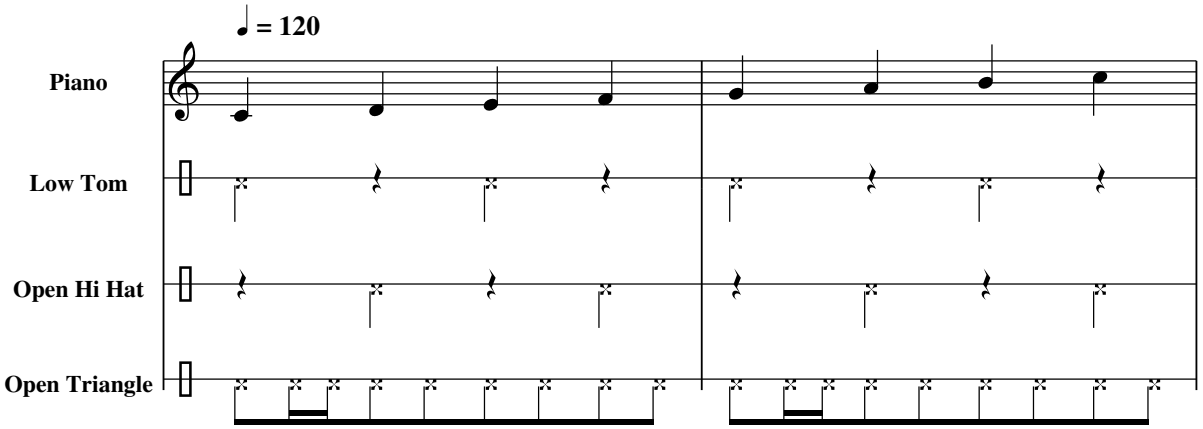
7.1.16 Percussion Instruments and Drum Mapping

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.8.2 (page 184).

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):

```
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.

7.1.17 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):

```
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.

7.1.18 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 7.1. To obtain notes higher or lower than the octave shown in the table, simply add or subtract 12 to the note.

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

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

Table 7.1: 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.

7.1.19 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.

7.2 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 Abc 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, page 23) 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 4.1.10, page 72) 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 10.5 (page 153).

♫ ♫ ♫ ♫ ♫

Chapter 8

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!

8.1 Font Directives, Revisited (`abc2svg`)

In Section 4.1.5 (page 65), we learnt how to change fonts. Actually, font directives support an extended syntax:

`%%*font` ⟨*font name*⟩ *[style/weight] [encoding] [size] [arguments]*

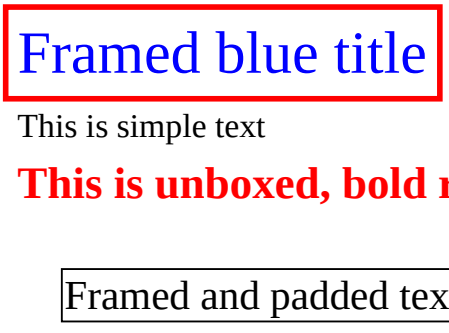
where:

- `%%*font` is one of the several font changing directives like `%%titlefont`, `%%textfont`, etc.;
- ⟨*font name*⟩ can also be an URL, which can point to a locally installed font or to a font available on the web. For example:
  `%%textfont url(http://example.org/fonts/coolfont.woff)`
  `%%textfont url(/usr/share/fonts/coolfont.otf)`
- *[style/weight]* can be one of the strings `italic`, `bold`, or `normal`. These strings can be separated from the font name by a hyphen or a space. The string `normal` removes both style and weight;
- *[encoding]* is not used by `abc2svg`;
- *[size]* is the font size in pixels, or `*` that means “keep the previous font size”.
- *[arguments]* can be one of the following strings:
  - `box`, which draws a box around the text, or `nobox` to stop drawing text boxes;

- `class=`⟨*class name*⟩ sets the text using a CSS class defined with `%%beginml` … `%%endml`;
- `padding=`⟨*int*⟩ sets the text padding in pixels;
- `wadj=`⟨*adjustment*⟩ sets the text width adjustment; the parameter can be `none`, `spacing`, or `glyph`. This parameter only works when using `abc2svg` in the command line.

It looks quite complicated, so let's see a pretty complete example:

```
% definition of 'redtext' and 'bluetext' classes
%%beginsvg
<style>
.redtext  color:red;
.bluetext color:blue; outline: 3px solid;
  outline-color:red; outline-offset: 5px;
</style>
%%endsvg

%%titleleft 1
%%titlefont * 32 class=bluetext

X: 1
T: Framed blue title
K: C
%%textfont serif 18
%%text This is simple text
%%textfont serif bold 24 nobox class=redtext
%%text This is unboxed, bold red text
%%textfont * normal 22 box padding=25
%%text Framed and padded text
%%textfont url(/usr/share/fonts/MuseJazzText.otf) nobox 20
%%text MuseJazz is a nice font
```



8.2 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`' 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.

8.2.1 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:

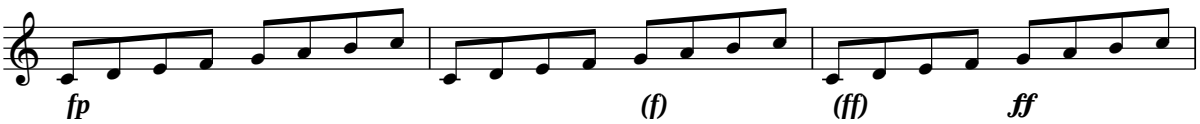
%%deco ⟨*name*⟩ ⟨*type*⟩ ⟨*function*⟩ ⟨*height*⟩ ⟨*wl*⟩ ⟨*wr*⟩ ⟨*string*⟩

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.

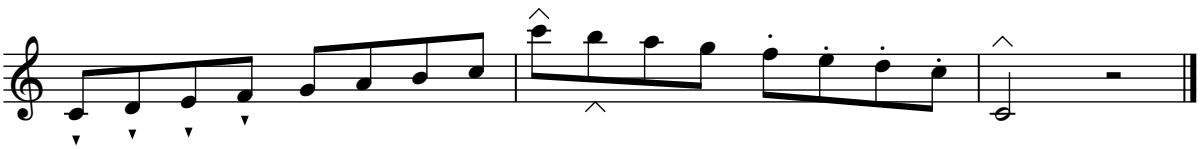
Let's have a look at an example taken from the file deco.abc supplied with abcm2ps. We will add a few new symbols for dynamics using the predefined pf routine:

```
%      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|]
```



8.3 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 typeset using the new shapes and/or positions.

The syntax is:

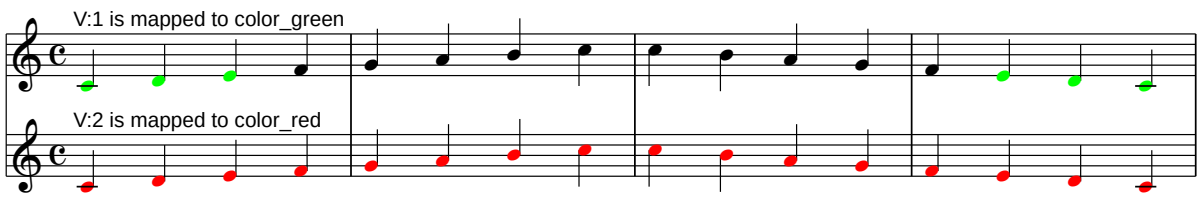
%%map ⟨*map_name*⟩ ⟨*mapped_note*⟩ *[heads=]*⟨*glyph names*⟩ *[print=]*⟨*note*⟩ *[color=]*⟨*#rrggbb*⟩

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 typeset as ⟨*note*⟩ (that is, on a different line or space);
- *[color=]*⟨*#rrggbb*⟩ defines note colours.

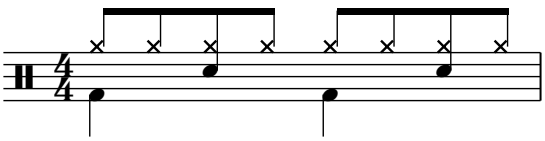
It sounds complicated, but some examples will hopefully clarify how %%map works. Let's define two maps; one that typeset notes heads C D E in green, and another that typeset all note heads in red:

```
X:1
M:C
L:1/4
K:C
%  first map: only C D E note heads typeset 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 typeset as 'g'
%%map drum ^A, print=g heads=x_head
%  Note D, (snare) will be typeset as 'c'
%%map drum  D, print=c
%  Note C, (bass drum) will be typeset 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 typeset 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 7.1.16, page 117).

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

8.4 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.

8.5 Accompaniment Chords in Italian Notation

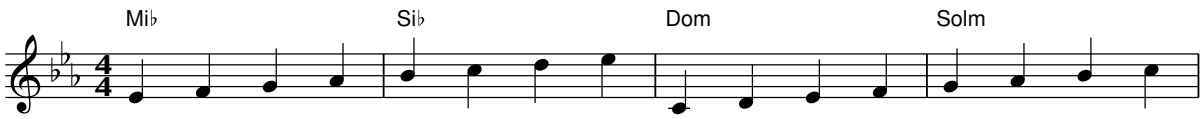
By default, accompaniment chords are typeset using English note names. Chord names can be redefined using the PostScript commands (`abcm2ps`) or the `%%chordnames` directive (`abc-2svg`).

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 typeset 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
```

8.6 Adding Text Fonts (`abcm2ps`)

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 9.26), Windows 7 and Windows 10 (Ghostscript 9.55 installed in `C:\gs\gs9.54`). You will need root/administrator privileges.

An excellent site boasting a wide collection of free and high-quality PostScript and TrueType fonts is http://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 9.55) and its fonts in default locations, such directory is:

- Windows: `C:\gs\gs9.54\lib`
- GNU/Linux: `/usr/share/fonts/`

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

```
gs -h
GPL Ghostscript 9.26 (2018-11-20)
...
Search path:
   /usr/share/ghostscript/9.26/Resource/Init :
   /usr/share/ghostscript/9.26/lib :
   /usr/share/ghostscript/9.26/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\gs9.54\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) ;
```

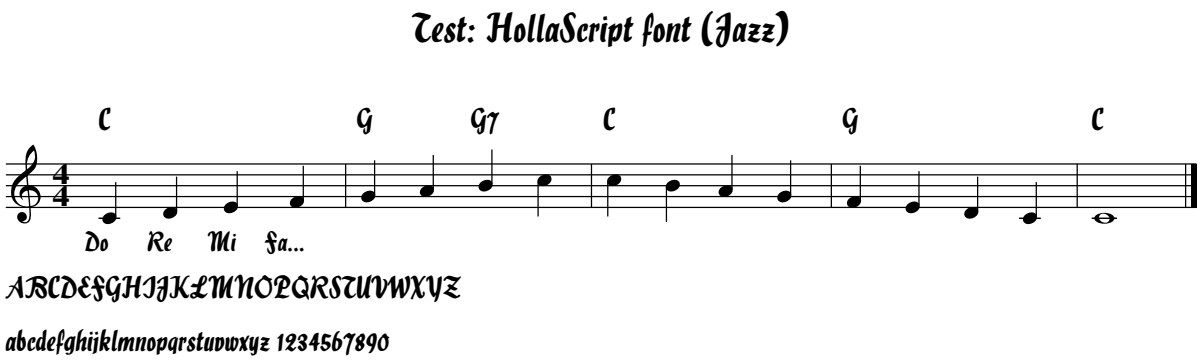
which defines a new font called `HollaScript`. If you wish, you can also define an *alias*, i.e. an alternate name for the same font adding a new line:

```
/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.

8.7 Adding Text Fonts (abc2svg)

Adding text fonts for abc2svg is much simpler; besides, most fonts will automatically work for abcm2ps too, without the complex installation procedure explained in the previous section. Only OpenType .otf fonts apply.

Let's download and install the following fonts:

- Gadelica, http://www.leabhair.ie/Gadelica.zip
- QT Arabian, https://www.ctan.org/tex-archive/fonts/qualitype/
- Urbanist Light Italic, https://github.com/coreyhu/Urbanist

Install the fonts as usual (GNU/Linux: just copy them to /usr/share/fonts), then declare their names in in %%font lines. Beware: you must specify the real name, as it appears in the font manager of your operating system, not just the file name. If the font name is made of more than one word, it must be enclosed in quotes:

```
% Fonts Gadelica.otf, QTArabian.otf, and
% Urbanist-LightItalic.otf are installed in /usr/share/fonts

%%font      Gadelica
```

```
%%font      QTArabian
%%font      "Urbanist Light Italic"

%%textfont  Gadelica 24
%%text      This is Gadelica
%%textfont  QTArabian 24
%%text      This is QTArabian
%%textfont  Urbanist Light Italic 24
%%text      This is Urbanist Light Italic
```

A useful font set is TeX Gyre, https://ctan.org/pkg/tex-gyre. It's an implementation of most standard PostScript fonts in OpenType format, compatible with both `abcm2ps` and `abc2svg`. In particular, the TeX Gyre implementation of web-safe fonts lets `abc2svg` and `abcm2ps` typeset identical output.

8.8 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 a SMuFL-compliant music font. SMuFL stands for Standard Music Font Layout; it is a specification designed to provide a standard way of mapping musical symbols to characters. Please note the difference between a *text font*, such as HollaScript in the previous section, and a *music font* that changes glyphs.

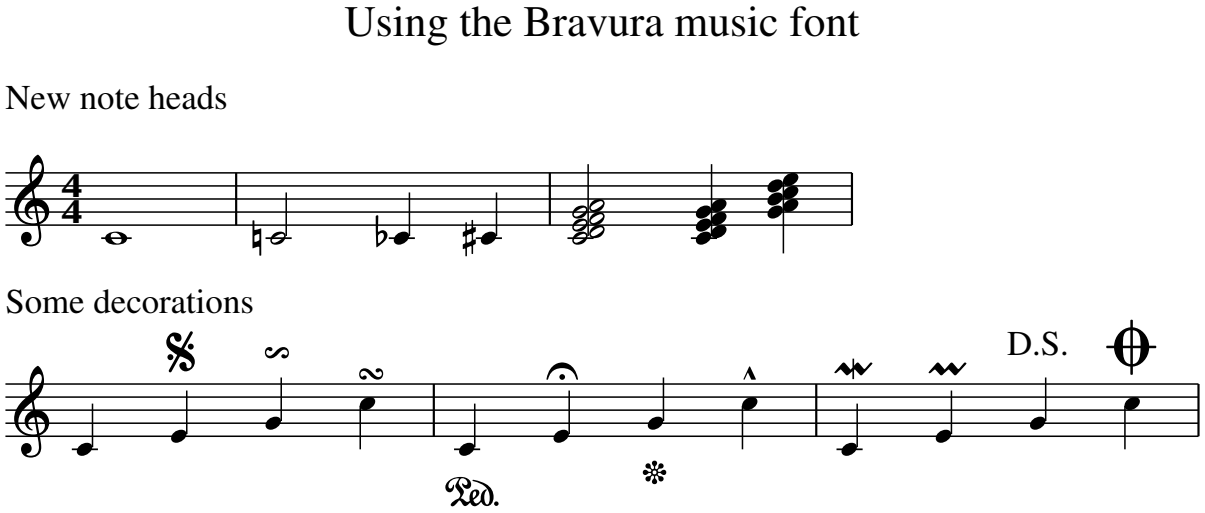
Some free and open source SMuFL compliant fonts are:

- Bravura, https://github.com/steinbergmedia/bravura
- Petaluma, https://github.com/steinbergmedia/petaluma; a Jazz style font
- Sebastian, https://github.com/fkretlow/sebastian
- Leland, https://github.com/MuseScoreFonts/Leland

Some of these fonts are available in two versions: one is used to typeset music glyphs in the score, the other is used to typeset music glyphs in text. For example, we have Bravura and BravuraText, Sebastian and SebastianText. These fonts are released in OpenType format; file extension is `.otf`.

Let's try Bravura; the procedure is the same for other fonts. Download `Bravura.otf` and install it in one of Ghostscript search directories, as explained in Section 8.6; adding the font in the `Fontmap` file is not required. The `%%musicfont` directive instructs `abcm2ps` and `abc2svg` to use the new glyphs provided by Bravura:

```
%%pagescale 1.2
%%musicfont Bravura

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  |
```



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

Using `abcm2ps`, not all symbols are replaced by the SMuFL font; for example, dynamics decorations like `!p!` are normally implemented as strings. To redefine these missing glyphs, we can use the `smufl.fmt` format file, available at the Abc 2 home page. To use `smufl.fmt`, we declare a SMuFL-compliant font before loading it:

```
%%musicfont  Bravura
%%format     smufl
```

then all new glyphs will be available using the specified SMuFL font.

♫ ♫ ♫ ♫ ♫

Chapter 9

Abc and MusicXML

9.1 Introducing MusicXML

Let'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 `abcm2ps` 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.

9.2 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`, 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 experiment with `xml2abc-js`' translation options.

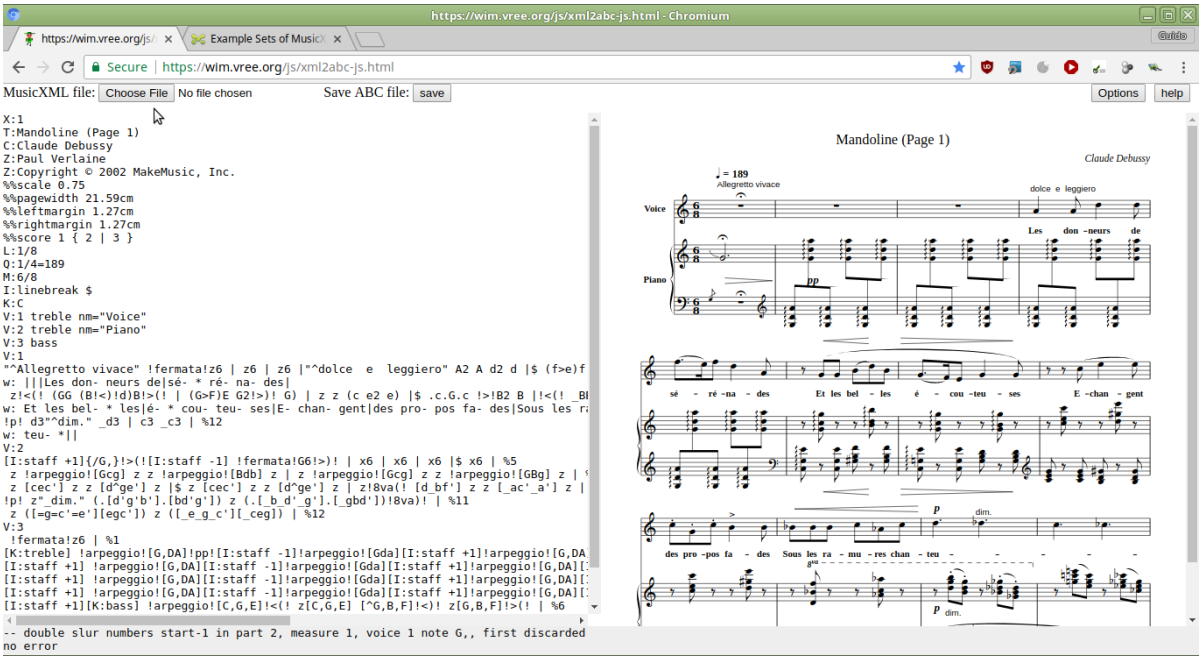


Figure 9.1: Importing and displaying MusicXML files in `xml2abc-js`.

9.3 Using `abc2xml.py` and `xml2abc.py`

`abc2xml.py` and `xml2abc.py` are command line programs that require Python; Windows users have a `.exe` version of these programs. 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 directory 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.

9.3.1 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.

9.4 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 in 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.

MuseScore can also convert to other formats: `png`, `svg`, `wav`, `flac`, `ogg`, `mp3`, `mid`, `xml`, and `mxl`.

9.5 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/.

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 multivoice files.

In this case, using MusicXML as an intermediate format produces better results. We can choose between two MusicXML to Lilypond converters: `musicxml2ly`, included in the Lilypond package; and `xml2ly`, part of the libmusicxml project, http://libmusicxml.sourceforge.net. `xml2ly` is currently the best translator, but you may have to compile it yourself.

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)
-- done in 0.01 secs
$ 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'
...lots of messages...
Converting to `./scale1.pdf'...
Success: compilation successfully completed
$ _
```

To convert ABC to MusicXML to LilyPond using `xml2ly`, the commands are:

```
$ abc2xml.py scale1.abc > scale1.xml
-- decoded from utf-8
-- skipped header: (field X,1)
-- done in 0.01 secs
$ xml2ly -aofn -q -lpv "2.18.2" scale1.xml
$ lilypond scale1.ly
GNU LilyPond 2.18.2
Processing `scale1.ly'
...lots of messages...
Converting to `./scale1.pdf'...
Success: compilation successfully completed
$ _
```

9.6 Converting Abc to Verovio

Verovio is a free and open source library for engraving music coded in MEI format. MEI stands for Music Encoding Initiative; it's an alternative to MusicXML. More details at the Verovio home page, https://www.verovio.org.

Verovio can be used in several ways; we only need its command line interface. Verovio accepts MusicXML input; basic Abc input is also supported, but many features are still missing. Output is in SVG format, so we will also need the `rsvg-convert` program that turns SVG files to PDF. `rsvg-convert` is part of the libRSVG library, included in most GNU/Linux distributions.

To turn an Abc file (single tune only) to PDF using Verovio, the commands are:

```
$ abc2xml.py file.abc > file.xml
$ verovio -a -f xml --footer none file.xml -o file
Output written to file_001.svg.
Output written to file_002.svg.
Output written to file_003.svg.
$ rsvg-convert -f pdf file*.svg -o file.pdf
$ _
```

A simpler way to convert MusicXML to PDF is the Verovio web converter https://www.verovio.org/musicxml.html. Load a MusicXML file, press the "Convert to MEI" button, then save the PDF from the "PDF" button in the left-hand menu.

> ✍ **Note**
> Remember, Abc is just a notation and is not tied to a specific application. We can choose between five different programs to typeset our music!

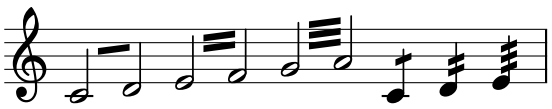
9.7 `abc2xml.py` Extensions

Since MusicXML supports more musical features than Abc, `abc2xml.py` provides a few extensions that MusicXML applications can exploit. The following Abc sources must be converted to MusicXML and fed to applications like MuseScore to typeset the final output.

9.7.1 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 or Verovio to convert the corresponding MusicXML file, we obtain the following score:

Unfortunately, `abc2midi` does not support any of the tremolo decorations. MIDI output, however, can be made by MuseScore.

9.7.2 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]*

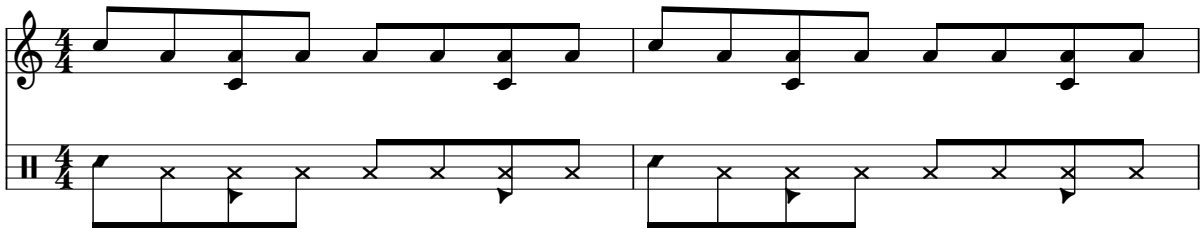
where:

- ⟨*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 typeset as *[mapped_note]*. Use `*` to typeset ⟨*note*⟩ on its normal position;
- ⟨*percussion*⟩ is the MIDI code of a percussion instrument, or its standard name (please see Section A.8.2, page 184). 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 |
```



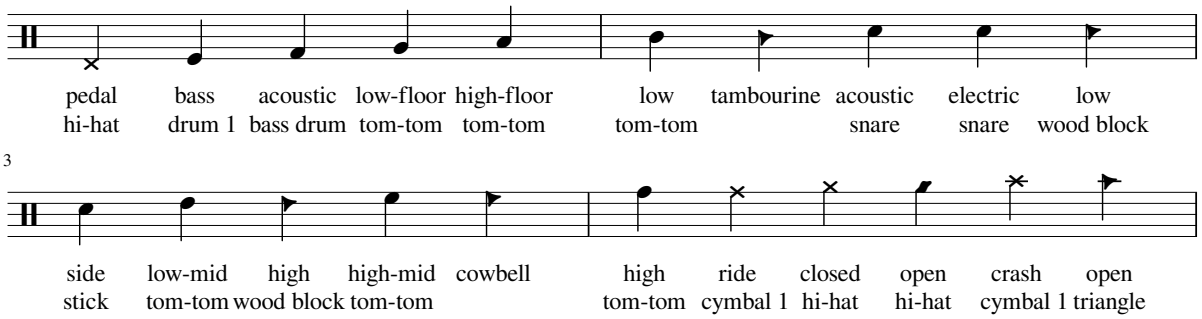
Let's now examine a more complete example that implements all percussion instruments described in the "Percussion notation" Wikipedia page. To play the file correctly with `abc-2midi`, we also specify the MIDI channel 10 (percussion instruments):

```
%%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



The above examples can only be typeset by MuseScore. Fortunately, we learnt in Section 8.3 (page 125) 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 |
```

9.7.3 Tablatures, Revisited

We learnt in Section 5.1 (page 91) how to define tablatures with `%%strtab`. Since `abc2xml.py` does not support this directive, tablatures that use the `%%strtab` syntax cannot be exported to MusicXML.

Fortunately, `abc2xml.py` supports its own extensions to the `K:` or `V:` fields to implement tablatures that can be exported. These extensions can be used alongside `%%strtab`, making the tablatures compatible with `abc2svg` too.

The syntax is:

`K:` or `V:` ⟨*clef=tab*⟩ *[strings=]*⟨*tuning*⟩ *[capo=*⟨*n*⟩*]* *[nostems]*

Parameters are:

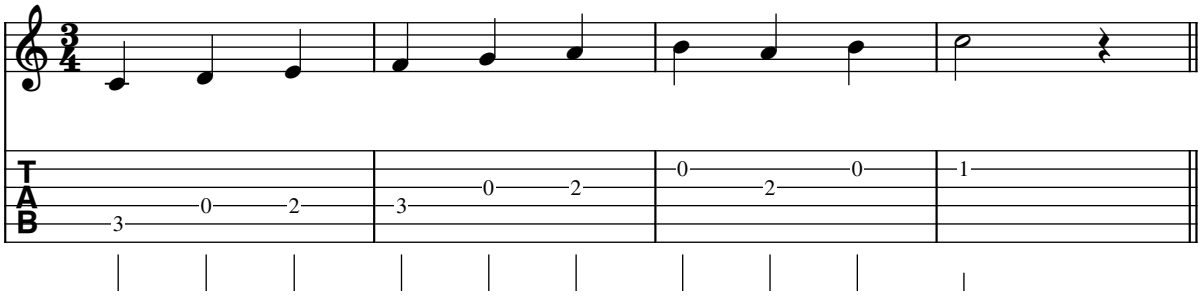
- *[strings=]*⟨*tuning*⟩ specifies an alternate strings tuning. ⟨*tuning*⟩ is a text string that defaults to `E2,A2,D3,G3,B3,E4`, i.e. the default guitar tuning. These are notes in International Pitch Notation, not ABC notes. Spaces between the notes are not allowed.
- *[capo=]*⟨*n*⟩ indicates a capo on fret ⟨*n*⟩; tuning is changed accordingly.

- *[nostems]* typesets tab notes with no stems.

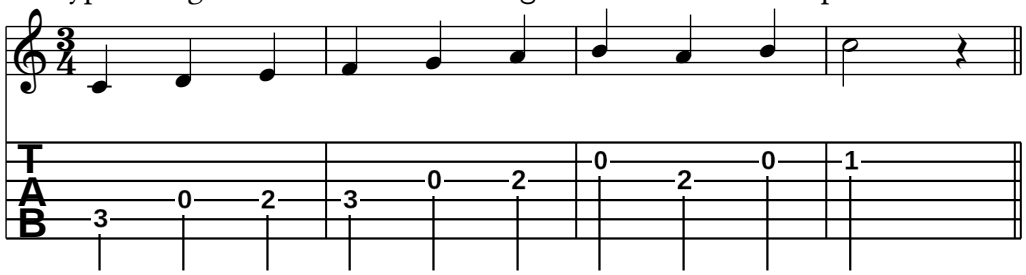
If we want to make the tablature compatible with abc2svg, we just insert a %%strtab line with no parameters. This directive tells abc2svg to use the abc2xml.py syntax.

As a first example, we rewrite the simple tablature we already used. abc2xml.py does not support multiple voice definitions in V:, so we will write the same voice twice:

```
%%strtab  % for abc2svg

X: 1
M: 3/4
L: 1/4
%%score 1 T
V: T clef=tab octave=-1 % use standard guitar tuning
K: C
%
V: 1
C D E | F G A | B A B | c2 z ||
V: T
C D E | F G A | B A B | c2 z ||
```



Typesetting the source with abc2svg we obtain similar output:

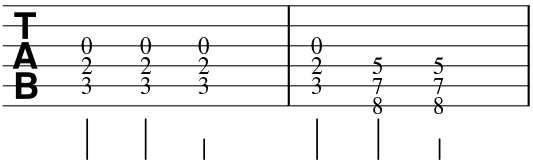


Alternate tunings are notated specifying the right notes in the strings= option. Guitar DADGAD tuning, for example, is defined as D2,A2,D3,G3,A3,D4; standard Irish bouzouki as G2,D2,A3,D3; bass guitar as E1,A1,D2,G2.

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 typeset a tablature for the C chord using default strings in the first measure, then specifying alternate string positions:

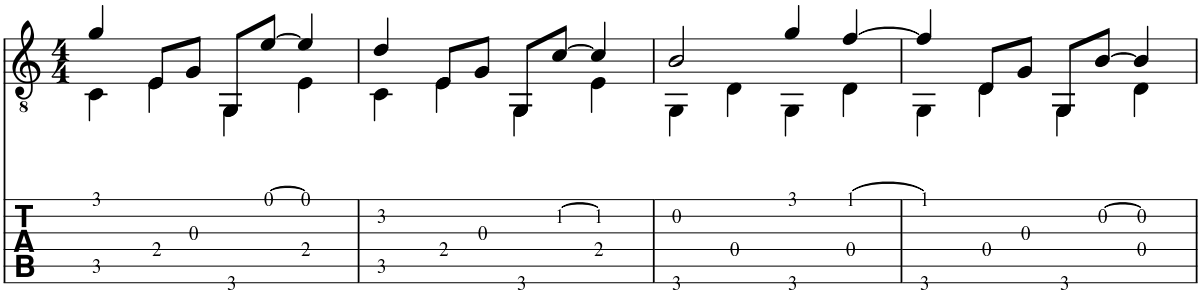
```
X: 1
L: 1/4
M: 2/4
K: C
%
V:1 clef=tab octave=-1
%   standard positions
[CEG] [CEG] [CEG]2 |
%   alternate fingering
[CEG] [!6!C!5!E!4!G] [CE!4!G]2 |
```



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 typeset 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:

```
%%strtab  % for abc2svg

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

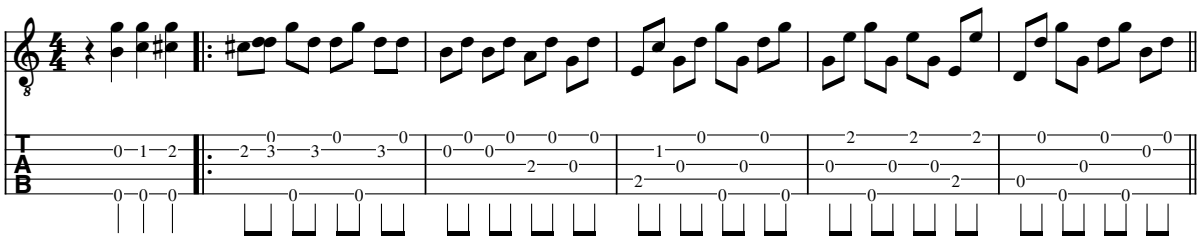
issue with MuseScore's handling of unisons in tablatures. In fact, when MuseScore finds a unison like E/ and E in voices 1 and 2, it puts the notes on two different strings: this may lead to unplayable positions. Other programs handle unisons correctly.

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`, needed by Musescore to make the tablature fit in one line.

```
%%linebreak <none>

%%scale 0.65  % for Musescore
%%strtab      % for abc2svg

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



9.8 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 param-

eters; 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 10

Advanced Usage

This chapter is written for expert users who can work comfortably with the Bash shell; many useful programs and scripts only work in this environment.

The following information *only applies to GNU/Linux* and possibly to macOS (not tested). Regrettably, many commands and scripts that work in Unix-like systems don't work in Windows, even in POSIX-like environments like MSYS2. Therefore, Windows power users will have to install Windows Subsystem for Linux, or a virtual machine running GNU/Linux.

10.1 Cropping PDF Files

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 you want to include the music in a document, whole pages are definitely bad.

To crop a PostScript or PDF file to its actual contents, we can use the following Bash script, `pdfcrop.sh`, which crops single-page PDF files using low-level Ghostscript commands:

```
#!/bin/sh

# pdfcrop.sh
# Guido Gonzato, PhD. GPL 2 or later.

MYSELF=$(basename $0)

if [ $# -eq 0 ] ; then
  printf "Usage: ${MYSELF} <file.pdf>\n"
  printf "This script uses ghostscript to crop a one-page pdf file.\n"
  exit 1
fi

GS=gs
INPUT=$1
PDF=$(basename $1 .pdf)
OUTPUT=$PDF-cropped.pdf
GSOPTS="-q -sDEVICE=bbox -dBATCH -dNOPAUSE"

# find the bounding box
$GS $GSOPTS $INPUT 2>&1 | grep "%%B" > $PDF.bbox

# read bbox coordinates in variables
```

```
read tmp X1 Y1 X2 Y2 < $PDF.bbox

# write the output, cropped to bbox
$GS -q -o $OUTPUT \
  -sDEVICE=pdfwrite \
  -c "[ /CropBox [$X1 $Y1 $X2 $Y2] /PAGES pdfmark" \
  -f $INPUT

/bin/rm -f $PDF.bbox
/bin/mv -f $OUTPUT $INPUT

echo "$INPUT cropped."
```

To crop a PostScript file, we will simply do:

```
$ ps2pdf scale1.ps       # scale1.pdf is created...
$ pdfcrop.sh scale1.pdf  # ... then cropped.
scale1.pdf cropped.
$ _
```

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:

```
$ 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.

10.2 Using abc2svg in the Command Line

We normally use abc2svg in its browser-based editor, but in this environment abc2svg cannot manage large ABC files; say, tune collections with more than 300 tunes. It may become too slow or even crash.

Luckily, abc2svg can also be used directly in the command line; in this case, it can manage much larger ABC files. abc2svg is actually a set of Javascript programs, so we will need a runtime such as Node.js®, which is freely available (often as a native package) for GNU/Linux, macOS, and Windows. Another Javascript runtime is QuickJS; it's very fast, and is probably the best choice for abc2svg in the command line.

10.2.1 Running abc2svg Scripts

The ABC 2 page provides packages for both QuickJS and abc2svg, which includes the abcqjs and abctopdf scripts. Alternatively, abc2svg can be installed using the Node.js® package manager npm:

```
# npm -g install abc2svg
# _
```

which installs the abc2svg, abc2odt, and abctopdf scripts. abcqjs is equivalent to abc2svg, but it's much faster.

abcqjs translates an Abc source to an html file:

```
$ abcqjs file.abc > file.html
$ _
```

file.html can then be loaded into any web browser; we can just display it or print it to PDF. As stated earlier, Chromium is the recommended browser.

Other actions are possible. abcqjs can load other scripts that convert the Abc sources in several ways:

- toabc.js performs tune selection and transposition, as explained in Section 6.5;
- toabw.js converts the input to Abiword ABW format;
- tomei.js converts the input (single tune only) to the MEI music notation format (https://music-encoding.org/about);
- toodt.js converts the input to OpenDocument ODT format.

For example, this is how to use abcqjs to select Abc files by K: field:

```
$ abcqjs toabc.js tunes.abc --select "K:D" > tunes_in_D.abc
$ _
```

10.2.2 Converting html to PDF

Output produced by abc2svg in html format is meant to be displayed in computer screens or tablets. However, html can also be converted to PDF: in other words, abc2svg can replace abcm2ps, in nearly all cases. The author of both programs recommends that users switch to abc2svg, since development of abcm2ps has stopped.

The abc2svg package provides a script, abctopdf, which converts an Abc source file to PDF. It uses the fastest Javascript engine it finds to convert Abc to html, then it looks for an html to PDF converter: a Chromium-based browser, or weasyprint, or wkhtmltopdf, or rsvg-convert. I have run many experiments using all available converters; in my tests, the most reliable converter is Chromium.

The command is:

```
$ abctopdf file.abc -o file.pdf
Using abcqjs
and chromium-browser
Using PPAPI flash.
... Written to file file.pdf.
```

If we omit the -o file.pdf option, the output file will be named abc.pdf.

The following script (abc2pdf.sh) is a simplified version of abctopdf that turns an Abc file to a single-page, cropped PDF file:

```
#!/bin/bash

# abc2pdf.sh
# Guido Gonzato, PhD. GPL 2 or later.

MYSELF=$(basename $0)

if [ $# -eq 0 ] ; then
  printf "Usage: ${MYSELF} <file.abc>\n"
  printf "abc -> cropped pdf conversion.\n"
  exit 1
fi

# the ABC file must be the last argument
FILE=$(basename "${BASH_ARGV}" .abc)
ABC2SVG=abcqjs
CHROME=chromium-browser

echo "Converting to $FILE.html..."
$ABC2SVG $@ > $FILE.html
echo "Converting to $FILE.pdf..."
$CHROME -headless --print-to-pdf=$FILE.pdf \
  --print-to-pdf-no-header $FILE.html
echo "Cropping $FILE.pdf..."
pdfcrop.sh $FILE.pdf
```

10.2.3 Inserting Music in Word Processors

To include music snippets in word processors like LibreOffice Writer, the simplest solution employs the `toabw.js` script:

```
$ abcqjs toabw.js music.abc > music.abw
$ _
```

`music.abw` can then be opened with LibreOffice and converted to other formats, if needed. The `toodt.js` script could also be used, but it needs the `jszip` package.

10.2.4 Inserting Music in LaTeX

This guide is written in LaTeX, which tech people typically 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 above. 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}
```

10.3 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.
```

10.4 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.

10.5 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*.

10.5.1 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.

10.5.2 Redefining Symbols

In Section 8.4 (page 126), 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
$ _
```

10.6 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 11

The End

11.1 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!

11.2 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!

11.3 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 threw 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
- Abc 2 standard, version 2.1:
  http://abcnotation.com/wiki/abc:standard:v2.1.
- 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/abcm2ps-doc/index.html
- The `abcMIDI` home page and user guide:
  http://ifdo.ca/~seymour/runabc/top.html
  https://abcmidi.sourceforge.io/
- TradMusician home page:
  http://nicolasjouve.fr/tradmusician/
- 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 PostScript 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

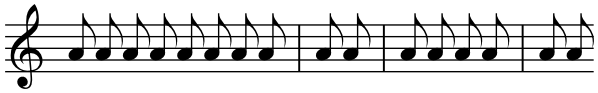
- MuseScore is a multiplatform, free and open source scorewriter:
  https://musescore.org
- GNU LilyPond is a multiplatform, free and open source text-based scorewriter:
  http://lilypond.org/web/
- The `abc` package for LaTeX:
  http://www.ctan.org/tex-archive/macros/latex/contrib/abc/
- The Abc Users mailing list:
  https://groups.io/g/abcusers
- Node.js® Javascript engine:
  https://nodejs.org
- QuickJS Javascript engine:
  https://bellard.org/quickjs/

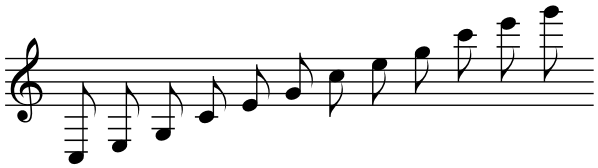
A.2 Abc Fields

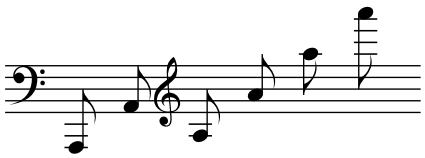
Field	Where	Notes and Example
A:	header	Area. `A:Liverpool`
B:	header	Book. `B:Groovy Songs`
C:	header	Composer. `C:The Beatles`
D:	header	Discography. `D:The Beatles Complete Collection`
F:	header	File name. `F:http://www.beatles.org/help.abc`
G:	header	Group. `G:guitar`
H:	header	History. `H:This song 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`

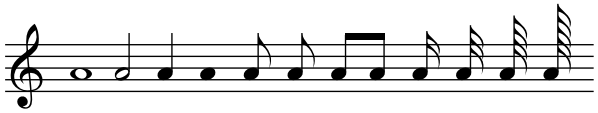
A.3 ABC Summary

Minimal example

```
X:1  % first header field, mandatory
K:C  % last header field, mandatory
% defaults: free meter, note = 1/8,
% bars can be inserted anywhere
A A A A A A A A | A A | A A A A | A A
```

Pitch, absolute

```
X:1
K:C clef=treble
C, E, G, C E G c e g c' e' g'
```

Pitch, relative

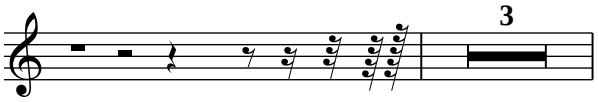
```
X:1
K:C
[K:octave=-3] A [K:octave=-2] A \
[K:octave=-1] A [K:octave=0] A \
[K:octave=1] A [K:octave=2] A
```

Note duration, absolute

```
X:1
K:C
A8 A4 A2 A0 A A AA A/ A// A/// A////
```

Note duration, relative

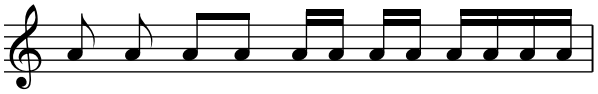
```
X:1
L:1/4
K:C
A4 A2 A A/ [L:1/8] A4 A2 A A/ \
[L:1/16] A4 A2 A A/
```

Rests

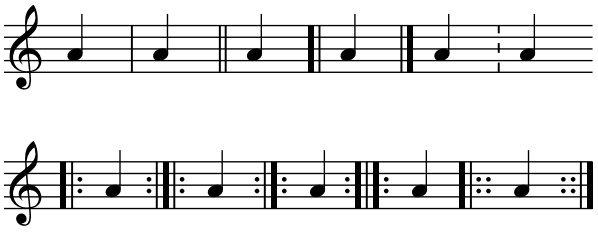
```
X:1
K:C
z8 z4 z2 y20 z z/ z// z/// z//// | Z3 |
```



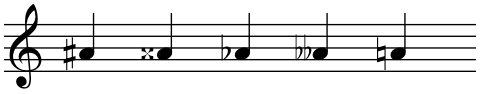
Beaming

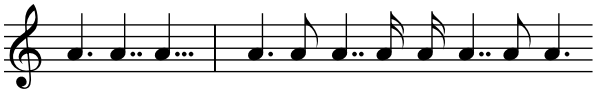
```
X:1
K:C
A A AA A/A/ A/A/ A/A/A/A/ |
```

Measure bars

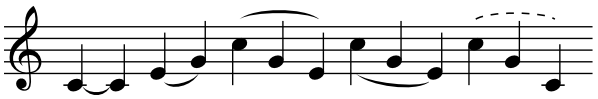
```
X:1
K:C
A2 | A2 || A2 [| A2 |] A2 .| A2 []
[|: A2 :|]|: A2 :|]: A2 :]|[: A2 \
|:: A2 ::|]
```

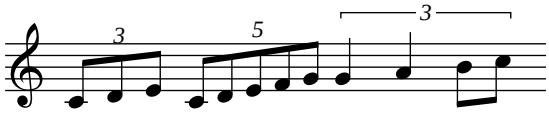


Accidentals

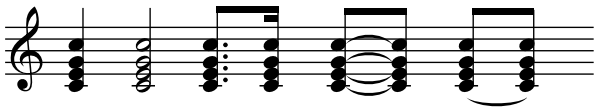
```
X: 1
K: C
^A2 ^^A2  _A2 __A2 =A2
```

Dotted notes

```
X:1
L:1/4
K:C
A3/2 A7/4 A15/8 | A>A A>>A A<<A A<A
```

Ties and slurs

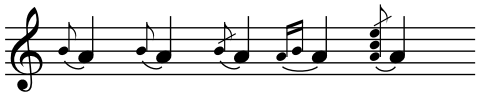
```
X:1
K:C
C2-C2 (E2 G2) (c2 G2 E2) \
(,c2 G2 E2) .(c2 G2 C2)
```

Tuplets

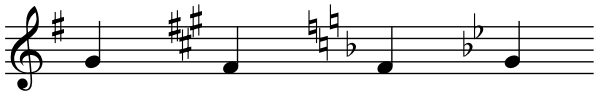
```
X:1
K:C
(3CDE (5CDEFG (3:2:4G2A2Bc
```

Chords

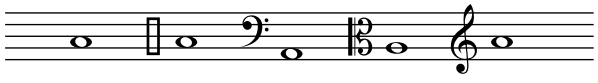
```
X:1
K:C
[CEGc]2 [CEGc]4 [CEGc]>[CEGc] \
[CEGc]-[CEGc] ([CEGc][CEGc])
```



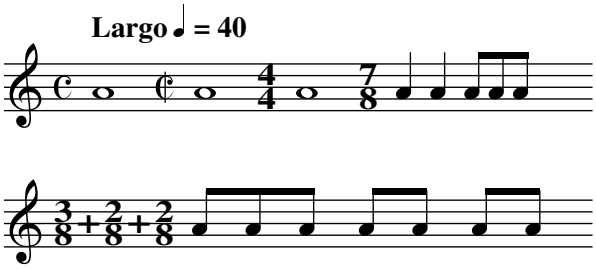
Grace notes

```
X:1
K:C
{B}A2 {B}A2 {/B}A2 {AB}A2 {/[Ace]}A2
```

Keys

```
X:1
K:C
[K:Gmaj] G2 [K:F#m] F2 \
[K:F] F2 [K:Gm] G2
```

Clefs

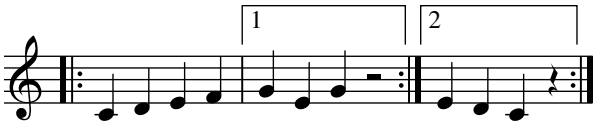
```
X: 1
K: C
[K:clef=none] A8 [K:clef=perc] A8 \
[K:clef=F] A,,8 [K:clef=C] A,8 \
[K:clef=G] A8
```

Time signatures

```
X:1
K:C
[M:C][Q: "Largo" 1/4=40] A8 [M:C|] A8 \
[M:4/4] A8 [M:7/8] A2 A2 AAA
[M:3/8+2/8+2/8] AAA AA AA
```



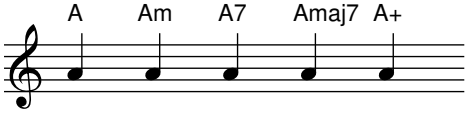
Repeats and endings

```
X:1
K:C
|: C2D2E2F2 |1 G2E2G2z4 :|2 E2D2C2z2 :|
```



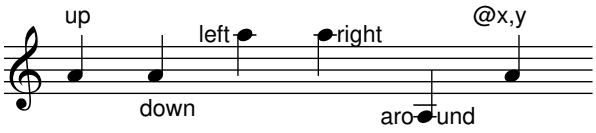
Accompaniment chords

```
X:1
K:C
"A"A2 "Am"A2 "A7"A2 "Amaj7"A2 "A+"A2
```

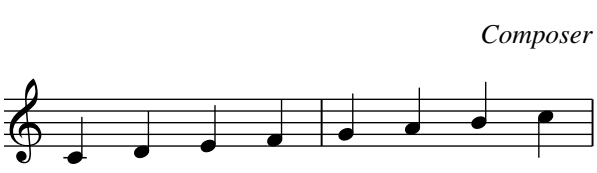


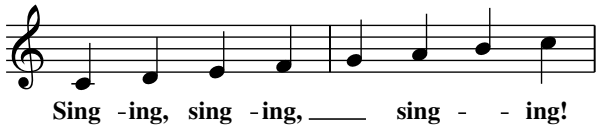
Text annotations

```
X:1
K:C
"^up"A2 "_down"A2 "<left"a2 ">right"a2\
"<aro" ">und"A,2 "@-20,25@x,y"A2
```

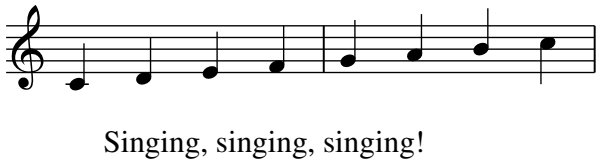


Title and composer

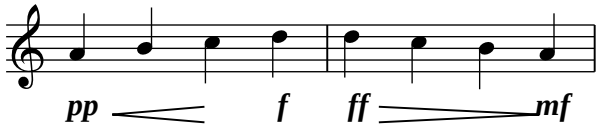
```
X: 1
T: Title
T: Subtitle
C: Composer
K: C
C2 D2 E2 F2 | G2 A2 B2 c2 |
```

Lyrics

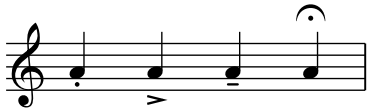
```
X: 1
K: C
C2 D2 E2 F2 | G2 A2 B2 c2 |
w: Sing-ing, sing-ing,_ sing -ing!_
```

Lyrics at end

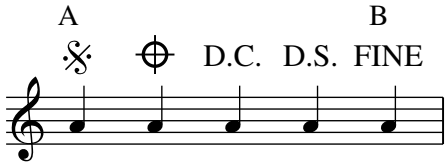
```
X: 1
K: C
C2 D2 E2 F2 | G2 A2 B2 c2 |
W: Singing, singing, singing!
```



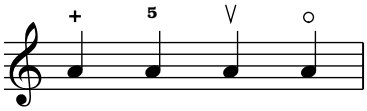
Dynamics

```
X:1
K:C
!pp!!<(!A2 B2 !<)!c2 !f!d2 | \
!ff!!>(!d2 c2 B2 !mf!!>)!A2 |
```

Articulations

```
X:1
K:C
.A2 !>!A2 !tenuto!A2 HA2 |
```

Position markings

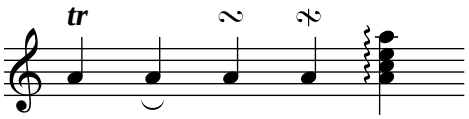
```
X:1
K:C
[P:A] !segno!A2 !coda!A2 \
!D.C.!A2 !D.S.!A2 [P:B] !fine!A2 |
```

Fingerings

```
X:1
K:C
!+!A2 !5!A2 !upbow!A2 !open!A2 |
```



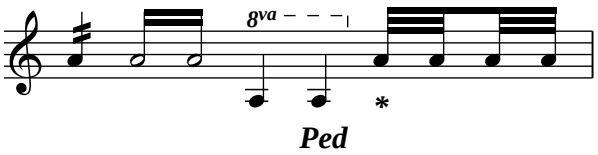
Ornaments

```
X:1
K:C
!trill!A2 !roll!A2 !turn!A2 \
!turnx!A2 !arpeggio![Acea]2
```



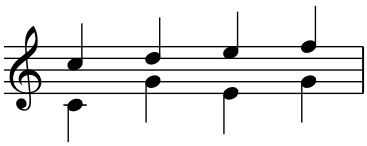
Other decorations

```
X:1
K:C
!//!A2 A2!trem2!A2 !8va(!A2 !ped!A2 \
!8va)!!ped-up!A//A//!beambr1!A//A//|
```



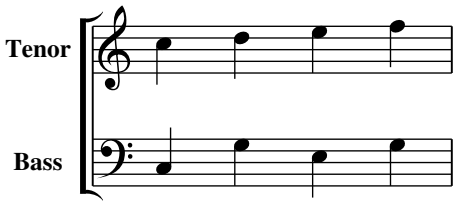
Voices

```
X:1
%%score (1 2)
K:C
[V:1] c2 d2 e2 f2 |
[V:2] C2 G2 E2 G2 |
```



Staves

```
X:1
%%score [1 2]
V:1 name="Tenor"
V:2 name="Bass" octave=-1
K:C
[V:1] c2 d2 e2 f2 |
[V:2] C2 G2 E2 G2 |
```



A.4 Common Errors and Bad Practices

This is a list of the most common errors that people make writing ABC files. I selected these errors from several tune collections available on the net.

A.4.1 Syntax Errors

This is quite obvious: please make sure that you write ABC sources that can be read and/or translated by software! Always double check.

A.4.2 Character Encoding

ABC is simple text, but text encoding is not necessarily simple. Please make sure that your source is either pure ASCII or UTF-8; other encodings will probably lead to wrong characters in output.

The `file` command on GNU/Linux and macOS is useful to check the encoding of ABC files:

```
$ file *abc
england.abc:       ISO-8859 text, with CRLF line terminators
ireland.abc:       data
klezmer.abc:       Non-ISO extended-ASCII text, with very long lines
O_Neills_1001.abc: data
O_Neills_1850.abc: ASCII text
oneills.abc:       ASCII text
scotland.abc:      ASCII text
session.abc:       UTF-8 Unicode text
tunes.abc:         ASCII text
```

In the above list, the only correct files are those described as "ASCII text" or "UTF-8 Unicode text". To fix an ISO-8859 encoded file, run this command:

```
$ iconv -f ISO-8859-15 -t UTF-8 < england.abc > england_utf8.txt
```

Files classified as "data" need manual corrections.

A.4.3 Lines Too Long

As I stressed in Section 3.1.5 (page 56), one of the main advantages af the ABC notation is readability. Please preserve it! Do do not write music lines that are longer than, say, 70 characters. You'll want your ABC source to fit the screen width on most devices; besides, long lines are notoriously harder to read.

A.4.4 `X:` Number in Titles

This is a very annoying bad practice. You get a tune collection, you want to select some tunes to include them in your own tune collection, and you have to edit the titles to remove an index number…

Please use `%%writefields X`, as explained in Section 4.1.21 (page 83)!

A.4.5 Ties vs. Slurs

Although ties and slurs look the same in printed music, they have a completely different musical meaning. Regrettably, some users abused the tie notation to connect notes of different pitch. This is a non-fatal error, but all ABC software will issue a warning.

Please use ties and slurs appropriately!

A.4.6 Outdated Syntax

The current ABC standard defines a set of outdated elements that one should avoid. The complete list is available here: http://abcnotation.com/wiki/abc:standard:v2.1#outdated_syntax

`Q:` Syntax

The `Q:` field used to accept a simplified syntax, like `Q: 108`. This is deprecated; please specify the tempo indicating the note length, like `Q: 1/4=108`.

`!` as End of Line

As we learnt in Section 4.1.6 (page 68), there are several ways to force line breaks. If your sources use `!` as an end of line marker, the easiest way to fix them is including the `%%I:linebreak !` command.

`+` for Decorations and Chords

In early version of ABC, decorations could be written using `+` as delimiters instead of `!`. Similarly, chords could once be written using `+` as delimiters instead of `[` and `]`. Please fix your old sources!

\ in `w:` Lines

Using \ to split a `w:` line into two `w:` lines is now deprecated: please use + instead.

`middle=` Specifier

The `K:` or `V:` `middle=`⟨*note*⟩ specifier was once used to specify the note that corresponds to the third line in the score. This syntax is now deprecated.

A.5 Formatting Directives

Directives supported by `abc2svg` and `abcm2ps` are listed and briefly described at this page: http://moinejf.free.fr/abcm2ps-doc/index.html

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.5.1 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 typeset as footer on each page. Default: none; scope: page; not available in abc2svg.

%%header ⟨***text***⟩**:** sets the text to be typeset 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.5.2 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 that will be typeset.

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

Nice tune indeed.
%%endtext
```

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

%%begintext obeylines typesets 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.5.3 Fonts

These directives specify the character fonts used in various parts of a score. Please note that the common True Type fonts used by the operating system *are not the same fonts* used by abcm2ps. In fact, abcm2ps uses the PostScript fonts, provided for and managed by Ghostscript. Standard PostScript fonts are shown in Appendix A.6 (page 179). I remind you that indications for adding new fonts are given in Section 8.6 (page 128).

%%annotationfont ⟨***string***⟩**:** font of annotations. Default: sans-serif 12; scope: immediate.

%%capofont ⟨***string***⟩**:** font of %%capo indications. Default: sans-serif 12; scope: immediate.

%%composerfont ⟨***string***⟩**:** font of C: field(s). Default: serifItalic 14; scope: tune.

%%footerfont ⟨***string***⟩**:** font of %%footer lines. Default: serif 16; 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: sans-serif 12; scope: immediate.

%%gridfont ⟨***string***⟩**:** font of chord grids. lines. Default: serif 16; scope: tune.

%%headerfont ⟨***string***⟩**:** font of %%header lines. Default: serif 16; scope: page.

%%historyfont ⟨***string***⟩**:** font of H: field. Default: serif 16; scope: tune.

%%infofont ⟨***string***⟩**:** font of I: field. Default: 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: serifItalic 10; scope: generation.

%%musicfont ⟨***string***⟩**:** defines a music font that replaces PostScript and/or SVG glyphs.

%%partsfont ⟨***string***⟩**:** font of P: field. Default: serif 15; scope: generation.

%%repeatfont ⟨***string***⟩**:** font of repeat numbers or text. Default: 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. Default size 12; scope: generation.

%%subtitlefont ⟨***string***⟩**:** font of the second T: field. Default: serif 16; scope: generation.

%%tempofont ⟨***string***⟩**:** font of Q: field. Default: serifBold 15; scope: tune.

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

%%tupletfont ⟨***string***⟩**:** font of tuplet indications. Default: serifItalic 12; scope: tune.

%%vocalfont ⟨***string***⟩**:** font of the text in w: lines. Default: serifBold 13; scope: generation.

%%voicefont ⟨***string***⟩**:** font of voice names. Default: serifBold 13; scope: tune.

%%wordsfont ⟨***string***⟩**:** font of the text in W: lines. Default: serif 16; scope: tune.

A.5.4 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***⟩**:** typesets a centered line long ⟨*length3*⟩, with ⟨*length1*⟩ space above and ⟨*length2*⟩ space below. Default; none; scope: immediate, restart.

%%sep**:** typesets 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.

%%tieheight ⟨***float***⟩**:** sets the tie height factor; lesser than 1 flattens the tie, greater than 1 expands it. Default; 1.0; 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.5.5 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***⟩**:** typesets an ambitus before the clef. Scope: generation. Not available in abcm-2ps.

%%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 markup language (XHTML) commands. Scope: immediate. Not available in abcm2ps.

%%beginps–%%endps**:** starts/ends a sequence of PostScript commands; please see Section 8.2 (page 122). Default: none; scope: immediate.

%%beginsvg–%%endsvg**:** starts/ends a sequence of SVG commands. Default: none; scope: immediate.

%%bgcolor ⟨***string***⟩**:** defines the background colour of HTML output. 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 typesets 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, typesets 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.

%%equalbars ⟨***bool***⟩**:** makes the width of measures proportional to their duration. Default: none; scope: generation; not available in abcm2ps.

%%fgcolor ⟨***string***⟩**:** defines the foreground colour of HTML output. The parameter can be a colour name, like 'green', or an RGB colour value, like #00ff00. Default: none; scope: page.

%%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***⟩**:** typesets 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, typesets measure numbers every ⟨*int*⟩ bars; if it is 0, typesets 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. PostScript 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, typesets straight flags on note stems in bagpipe tunes. Default: 0; scope: generation.

%%staff ⟨***int***⟩**:** typesets 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.5.6 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.

%%transpose: replaced by shift=.

%%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.6 PostScript Fonts

These are the 35 standard PostScript fonts, supported by `abcm2ps`:

AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Light
Bookman-LightItalic
Bookman-Demi
Bookman-DemiItalic
`Courier`
`Courier-Bold`
`Courier-Oblique`
`Courier-BoldOblique`
Helvetica
Helvetica-Bold
Helvetica-Oblique
Helvetica-BoldOblique
Helvetica-Narrow
Helvetica-Narrow-Bold
Helvetica-Narrow-Oblique
Helvetica-Narrow-BoldOblique
NewCenturySchlbk-Roman
NewCenturySchlbk-Bold
NewCenturySchlbk-Italic
NewCenturySchlbk-BoldItalic
Palatino-Roman
Palatino-Bold
Palatino-Italic
Palatino-BoldItalic
Times-Roman
Times-Bold
Times-Italic
Times-BoldItalic
Σψμβολ (Symbol)
ZapfChancery-MediumItalic
✺❁❒❄⌘❇■✳✡❁▼▲ (ZapfDingbats)

A.7 `abcMIDI` Directives

Most of these directives are also listed and explained at the `abcMIDI` page, https://abcmidi.sourceforge.io/

`%%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 nobeataccents`.

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

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

`%%MIDI 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 redefines existing ones as was seen in Section 7.1.11 (page 113).

`%%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 development.

`%%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 7.1.15 (page 116).

%%MIDI drummap ⟨***str***⟩ ⟨***int***⟩: associates the note ⟨*str*⟩ (in ABC notation) to the a percussion instrument, as listed in Section A.8.2 (page 184).

%%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 7.1.10 (page 111).

%%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.8 MIDI Instruments

A.8.1 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

Chromatic Percussion
9. Celesta
10. Glockenspiel
11. Music Box
12. Vibraphone
13. Marimba
14. Xylophone
15. Tubular Bells
16. Dulcimer

Organ
17. Drawbar Organ
18. Percussive Organ
19. Rock Organ
20. Church Organ
21. Reed Organ
22. Accordion
23. Harmonica
24. Tango Accordion

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

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

Solo Strings
41. Violin
42. Viola
43. Cello
44. Contrabass
45. Tremolo Strings
46. Pizzicato Strings
47. Orchestral Strings
48. Timpani

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

Brass
57. Trumpet
58. Trombone
59. Tuba
60. Muted Trumpet
61. French Horn
62. Brass Section
63. SynthBrass 1
64. SynthBrass 2

Reed
65. Soprano Sax
66. Alto Sax
67. Tenor Sax
68. Baritone Sax
69. Oboe
70. English Horn
71. Bassoon
72. Clarinet

Pipe
73. Piccolo
74. Flute
75. Recorder
76. Pan Flute
77. Blown Bottle
78. Skakuhachi
79. Whistle
80. Ocarina

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)

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)

Synth Effects
97. FX 1 (rain)
98. FX 2 (soundtrack)

Ethnic
105. Sitar
106. Banjo

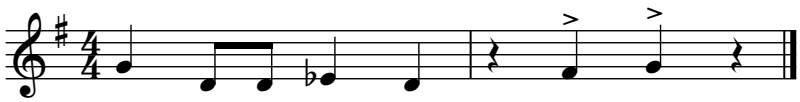
Percussive
113. Tinkle Bell
114. Agogo

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)
107. Shamisen
108. Koto
109. Kalimba
110. Bagpipe
111. Fiddle
112. Shanai
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.8.2 Percussion Instruments

These instruments can be used with `%%MIDI drum`, or using the corresponding note in the MIDI channel 10.

35. `B,,` Acoustic Bass Drum	**36**. `C,` Bass Drum 1	**37**. `^C,` Side Stick
38. `D,` Acoustic Snare	**39**. `^D,` Hand Clap	**40**. `E,` Electric Snare
41. `F,` Low Floor Tom	**42**. `^F,` Closed Hi Hat	**43**. `G,` High Floor Tom
44. `^G,` Pedal Hi-Hat	**45**. `A,` Low Tom	**46**. `^A,` Open Hi-Hat
47. `B,` Low-Mid Tom	**48**. `C` Hi Mid Tom	**49**. `^C` Crash Cymbal 1
50. `D` High Tom	**51**. `^D` Ride Cymbal 1	**52**. `E` Chinese Cymbal
53. `F` Ride Bell	**54**. `^F` Tambourine	**55**. `G` Splash Cymbal
56. `^G` Cowbell	**57**. `A` Crash Cymbal 2	**58**. `^A` Vibraslap
59. `B` Ride Cymbal 2	**60**. `c` Hi Bongo	**61**. `^c` Low Bongo
62. `d` Mute Hi Conga	**63**. `^d` Open Hi Conga	**64**. `e` Low Conga
65. `f` High Timbale	**66**. `^f` Low Timbale	**67**. `g` High Agogo
68. `^g` Low Agogo	**69**. `a` Cabasa	**70**. `^a` Maracas
71. `b` Short Whistle	**72**. `c'` Long Whistle	**73**. `^c'` Short Guiro
74. `d'` Long Guiro	**75**. `^d'` Claves	**76**. `e'` Hi Wood Block
77. `f'` Low Wood Block	**78**. `^f'` Mute Cuica	**79**. `g'` Open Cuica
80. `^g'` Mute Triangle	**81**. `a'` Open Triangle	

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.