ABC2MTEX

An easy way of transcribing folk and traditional music

Version 1.6.1 – January '97

Chris Walshaw
School of Maths,
University of Greenwich,
London, U.K.
email: `C.Walshaw@gre.ac.uk`

Contents

1 Introduction

Welcome to ABC2MTEX!

This is a package written to typeset tunes stored in an ascii format or `abc` notation. It was designed primarily for folk and traditional tunes of Western European origin (such as Irish, English and Scottish) which can be written on one stave in standard classical notation. However, it can act as a fast preprocessor for multi-stave music (see §4.2) and should be extendible to many other types of music.

The package is small, easy to use and features, amongst other things, the ability to transpose both the music and the `abc` notation. It will also create an index of all the tunes you have transcribed.

The ability to write tunes in `abc` notation means that they can be easily and portably stored or transported electronically. This package allows you to typeset them easily too. In addition the notation is not specifically tied to MusicTEX and similar packages have been written to translate to other formats such as postscript and MIDI.

This package has, however, been written on top of MusicTEX, Daniel Taupin's music typesetting package, itself written on top of TEX, Donald Knuth's typesetting package. To run it you will need TEX, MusicTEX (both freely available by ftp from a number of sources – see the `README` file), a C compiler (although executables are available for PCs and Macs) and access to a postscript printer.

1.1 Other `abc` packages

Since its introduction, `abc` has become a popular method for notating, playing and discussing music. There are several other packages available including typesetters which don't require the installation of TEX and MusicTEX, players (to play `abc` tunes through the speakers of most types of computer) and translators (e.g. to convert `abc` to MIDI and vice-versa). An up to date list is maintained at `http://www.gre.ac.uk/~c.walshaw/abc/`.

1.2 The ABC2MTEX mailing list

An abc users mailing list has just been set up – subscribe by sending the message `subscribe abc-users` to `majordomo@ecs.soton.ac.uk`. In addition, I keep a list of anyone who has mailed me to ask for a copy of the code and automatically notify them of new releases. If you have got the code by `ftp` and want to be added to the list just mail me `C.Walshaw@gre.ac.uk`. Traffic is *low* – maybe one message every couple of months, if that.

1.3 Acknowledgements

Thanks to everyone who has helped to improve this package: John Walsh, Don Ward, Steve Allen, David West, Roger Negaret, Hugh Stewart, Rich Holmes, Olivier Clary, Jeroen Nijhof (for the transposition of accidentals algorithm), Paul Anderson (for the Mac version), Michael Methfessel and everyone else.

1.4 Copyright disclaimer

The author does not accept responsibility for any copyright infringement which occurs through the use of this package. It is the responsibility of the user to ensure that tunes and/or arrangements typeset with the help of ABC2MTEX lie within the public domain. Happily this is the case with most traditional music.

2 Tune files

The music can either be stored one tune per file with a name that reflects the contents, e.g. the tune 'Paddy O'Rafferty' in the file `PaddyORafferty.abc`, or grouped together in files according to some common theme, e.g. `Reels.abc` contains Irish reels and English.abc has English tunes of all types. If the tunes are grouped then an index is useful and indeed indexing facilities are provided here (see the document index.tex). The example files, in particular English.abc, demonstrate typical file structures.

Each tune consists of a header and a body. The header, which is composed of information fields, should start with an X (reference number) field followed by a T (title) field and finish with a K (key) field. The body of the tune in abc notation should follow immediately after. Tunes are separated by blank lines.

2.1 Information fields

The information fields are used to notate things such as composer, meter, etc. ... in fact anything that isn't music. Most of the information fields are for use within a tune header but in addition some may be used in the tune body, or elsewhere in the tune file. Those which are allowed elsewhere can be used to set up a default for the whole or part of a file. For example, in exactly the same way that tunebooks are organised, a file might start with M:6/8 and R:Jigs, followed by some jigs, followed by M:4/4 and R:Reels, followed by some reels. Tunes within each section then inherit the M: and R: fields automatically, although they can be overridden inside a tune header. Finally note that any line beginning with a letter in the range A–Z and immediately followed by a : is interpreted as a field (so that line like E:|, which could be regarded as an E followed by a right repeat symbol, will cause an error).

By far the best way to find out how to use the fields is to look at the example files (in particular English.abc) and try out some examples. Thus rather than describing them in detail, they are summarised in the following table. The second, third and fourth columns specify respectively how the field should be used in the header and whether it may used in tune body or elsewhere in the file. Certain fields do not affect the typeset music but are there for other reasons, and the fifth column reflects this; index fields only affect the index (see index.tex) while archive fields do not affect the output at all, but are just provided to put in information that one might find in, say, a conventional tunebook.

	Can be used in:				
Field name	header	tune	elsewhere	Used by	Examples and notes
A:area	yes				A:Donegal, A:Bampton
B:book	yes		yes	archive	B:O'Neill's
C:composer	yes				C:Trad.
D:discography	yes			archive	D:Chieftains IV
E:elemskip	yes	yes			see §2.3.3
F:file name			yes		see index.tex
G:group	yes		yes	archive	G:flute
H:history	yes		yes	archive	H:This tune is said to ...
I:information	yes	yes	yes	playabc	see the playabc user manual
K:key	last	yes			K:G, K:Dm, K:AMix
L:default note length	yes	yes			L:1/4, L:1/8
M:meter	yes	yes	yes		M:3/4, M:4/4
N:notes	yes				N:see also O'Neill's - 234
O:origin	yes		yes	index	O:I, O:Irish, O:English
P:parts	yes	yes			P:ABAC, P:A, P:B
Q:tempo	yes	yes			Q:200, Q:C2=200
R:rhythm	yes		yes	index	R:R, R:reel
S:source	yes				S:collected in Brittany
T:title	second	yes			T:Paddy O'Rafferty
W:words		yes			W:Hey, the dusty miller
X:reference number	first				X:1, X:2
Z:transcription note	yes				Z:transcribed from photocopy

Some additional notes on certain of the fields:–

T – tune title. Some tunes have more than one title and so this field can be used more than once per tune – the first time will generate the title whilst subsequent usage will generate the alternatives in small print. The normal TEX accents (such as in caf\'e for producing café or in makazi\u ce for producing makaziče) can be used but will be stripped out of the index. The T: field can also be used within a tune to name parts of a tune – in this case it should come before any key or meter changes.

`K` – key; the key signature should be specified with a capital letter which may be followed by a `#` or `b` for sharp or flat respectively. In addition, different scales or modes can be specified and, for example, `K:F lydian`, `K:C`, `K:C major`, `K:C ionian`, `K:G mixolydian`, `K:D dorian`, `K:A minor`, `K:Am`, `K:A aeolian`, `K:E phrygian` and `K:B locrian` would all produce a staff with no sharps or flats. The spaces can be left out, capitalisation is ignored for the modes and in fact only the first three letters of each mode are parsed so that, for example, `K:F# mixolydian` is the same as `K:F#Mix` or even `K:F#MIX`. There are two additional keys specifically for notating highland bagpipe tunes; `K:HP` doesn't put a key signature on the music, as is common with many tune books of this music, while `K:Hp` marks the stave with F♯, C♯ and G♮. Both force all the beams and staffs to go downwards.

Finally, global accidentals can also be set in this field so that, for example, `K:D =c` would write the key signature as two sharps (key of D) but then mark every c as natural (which is conceptually the same as D mixolydian). Note that the there can be several global accidentals, separated by spaces and each specified with an accidental, `__`, `_`, `=`, `^` or `^^`, (see below) followed by a letter in lower case. Global accidentals are overridden by accidentals attached to notes within the body of the abc tune and are reset by each change of signature. See §2.3 for further notes on their use with ABC2MTeX.

`L` – default note length; i.e. `L:1/4` – quarter note, `L:1/8` – eighth note, `L:1/16` – sixteenth, `L:1/32` – thirty-second. The default note length is also set automatically by the meter field `M:` (see §2.2.3).

`M` – meter; apart from the normal meters, e.g. `M:6/8` or `M:4/4`, the symbols `M:C` and `M:C|` give common time and cut time respectively.

`P` – parts; can be used in the header to state the order in which the tune parts are played, i.e. `P:ABABCDCD`, and then inside the tune to mark each part, i.e. `P:A` or `P:B`.

`Q` – tempo; can be used to specify the notes per minute, e.g. if the default note length is an eighth note then `Q:120` or `Q:C=120` is 120 eighth notes per minute. Similarly `Q:C3=40` would be 40 dotted quarter notes per minute. An absolute tempo may also be set, e.g. `Q:1/8=120` is also 120 eighth notes per minute, irrespective of the default note length. The tempo may also be changed relative to the previous tempo; `Q:`n=m means that, from now on, a note value of n occupies the same time as a note value of m did previously. Both m and n must be prefixed by the pitch value `C` (which is ignored) to distinguish them from the earlier examples. For example, `Q:C4=C6` slows down the tempo by a factor of 3/2; a note of length 4 takes exactly the same time as did a note of length 6 previously. When the meter changes from 4/4 (8 default notes per bar) to 6/8 (6 default notes per bar) a tempo change of `Q:C3=C4` is needed to keep the number of bars per minute the same, i.e. to maintain the pulse of the music.

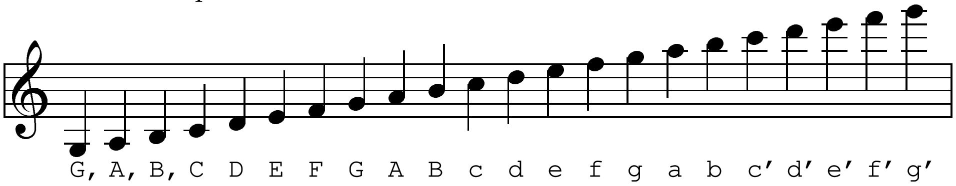
`G` – group; to group together tunes for indexing purposes.

`H` – history; can be used for multi-line stories/anecdotes, all of which will be ignored until the next field occurs.

2.2 `abc` tune notation

2.2.1 Notes

The following letters are used to represent notes:–



and by extension, the notes `C, D, E, F, a′` and `b′` are available. Notes can be modified in length (see §2.2.3).

2.2.2 Rests

Rests are generated with a `z` and can be modified in length in exactly the same way as notes can.

2.2.3 Note lengths

NB Throughout this document note lengths are referred as sixteenth, eighth, etc. The commonly used equivalents are sixteenth note = semi-quaver, eighth = quaver, quarter = crotchet and half = minim.

Each meter automatically sets a default note length and a single letter in the range `A-G`, `a-g` will generate a note of this length. For example, in 3/4 the default note length is an eighth note and so the input `DEF` represents 3 eighth notes. The default note length can be calculated by computing the meter as a decimal; if it is less than 0.75 the default is a sixteenth note, otherwise it is an eighth note. For example, 2/4 = 0.5, so the default note length is a sixteenth note, while 4/4 = 1.0 or 6/8 = 0.75, so the default is an eighth note. Common time and cut time (`M:C` and `M:C|`) have an eighth note as default.

Notes of differing lengths can be obtained by simply putting a multiplier after the letter. Thus in 2/4, `A` or `A1` is a sixteenth note, `A2` an eighth note, `A3` a dotted eighth note, `A4` a quarter note, `A6` a dotted quarter note, `A7` a double dotted quarter note, `A8` a half note, `A12` a dotted half note, `A14` a double dotted half note, `A15` a triple dotted half note and so on, whilst in 3/4, `A` is an eighth note, `A2` a quarter note, `A3` a dotted quarter note, `A4` a half note, ...

To get shorter notes, either divide them – e.g. in 3/4, `A/2` is a sixteenth note, `A/4` is a thirty-second note – or change the default note length with the `L:` field. Alternatively, if the music has a broken rhythm, e.g. dotted eighth note/sixteenth note pairs, use broken rhythm markers (see §2.2.4). Note that `A/` is shorthand for `A/2`.

2.2.4 Broken rhythms

A common occurrence in traditional music is the use of a dotted or broken rhythm. For example, hornpipes, strathspeys and certain morris jigs all have dotted eighth notes followed by sixteenth notes as well as vice-versa in the case of strathspeys. To support this `abc` notation uses a > to mean 'the previous note is dotted, the next note halved' and < to mean 'the previous note is halved, the next dotted'. Thus the following lines all mean the same thing (the third version is recommended):

```
L:1/16
a3b cd3 a2b2c2d2

L:1/8
a3/2b/2 c/2d3/2 abcd

L:1/8
a>b c<d abcd
```

As a logical extension, >> means that the first note is double dotted and the second quartered and >>> means that the first note is triple dotted and the length of the second divided by eight. Similarly for << and <<<.

2.2.5 Duplets, triplets, quadruplets, etc.

These can be simply coded with the notation `(2ab` for a duplet, `(3abc` for a triplet or `(4abcd` for a quadruplet, etc., up to `(9`. The musical meanings are:

- `(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
- `(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

If the time signature is compound (3/8, 6/8, 9/8, 3/4, etc.) then n is three, otherwise n is two.

More general tuplets can be specified using the syntax `(p:q:r` which means 'put p notes into the time of q for the next r notes'. If q is not given, it defaults as above. If r is not given, it defaults to p. For example, `(3:2:2` is equivalent to

`(3::2` and `(3:2:3` is equivalent to `(3:2 , (3` or even `(3:: .` This can be useful to include notes of different lengths within a tuplet, for example `(3:2:2G4c2` or `(3:2:4G2A2Bc` and also describes more precisely how the simple syntax works in cases like `(3D2E2F2` or even `(3D3EF2`. The number written over the tuplet is p.

2.2.6 Beams

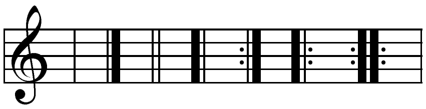
To group notes together under one beam they should be grouped together without spaces. Thus in 2/4, `A2BC` will produce an eighth note followed by two sixteenth notes under one beam whilst `A2 B C` will produce the same notes separated. The beam slopes and the choice of upper or lower staffs are generated automatically.

2.2.7 Repeat/bar symbols

Bar line symbols are generated as follows:

`|`	bar line
`|]`	thin-thick double bar line
`||`	thin-thin double bar line
`[|`	thick-thin double bar line
`:|`	left repeat
`|:`	right repeat
`::`	left-right repeat

Viz:–



2.2.8 First & second repeats

First and second repeats can be generated with the symbols `[1` and `[2`, e.g. `faf gfe|[1 dfe dBA:|[2 d2e dcB|].` When adjacent to bar lines, these can be shortened to `|1` and `:|2`, but with regard to spaces `| [1` is legal, `| 1` is not.

2.2.9 Accidentals

The symbols `^` = and _ are used (before a note) to generate respectively a sharp, natural or flat. Double sharps and flats are available with `^^` and __ respectively.

2.2.10 Changing key, meter, and default note length mid-tune

To change key, meter, or default note length, simply put in a new line with a `K:` `M:` or `L:` field, e.g.

```
ed|cecA B2ed|cAcA E2ed|cecA B2ed|c2A2 A2:|
K:G
AB|cdec BcdB|ABAF GFE2|cdec BcdB|c2A2 A2:|
```

To do this without generating a new line of music, put a \ at the end of the first line, i.e.

```
E2E EFE|E2E EFG|\
M:9/8
A2G F2E D2|]
```

2.2.11 Ties and slurs

You can tie two notes together either across or within a bar with a - symbol, e.g. abc-|cba or abc-cba. More general slurs can be put in with () symbols. Thus (DEFG) puts a slur over the four notes. Spaces within a slur are OK, e.g. (D E F G), but the open bracket should come immediately before a note (and its accents/accidentals, etc.) and the close bracket should come immediately after a note (and its octave marker or length). Thus (=b c'2) is OK but (=b c'2) is not.

The old notation, e.g. sDEFsG, can still be used by setting a user switch – see §2.3.1.

2.2.12 Gracings

Grace notes can be written by enclosing them in curly braces, {}. For example, a taorluath on the Highland pipes would be written {GdGe}. The tune 'Athol Brose' (in the file Strspys.abc) has an example of complex Highland pipe gracing in all its glory. Grace notes have no time value and so expressions such as {a2} or {a>b} are not legal.

Alternatively, the tilde symbol ~ represents the general gracing of a note which, in the context of traditional music, can mean different things for different instruments, for example a roll, cran or staccato triplet

2.2.13 Accents

Staccato marks (a small dot above or below the note head) can be generated by a dot before the note, i.e. a staccato triplet is written as (3.a.b.c

For fiddlers, the letters u & v can be used to denote up-bow & down-bow, e.g. vAuBvA

2.2.14 Chords & unisons

Chords (i.e. more than one note head on a single stem) can be coded with [] symbols around the notes, e.g. [CEGc] produces the chord of C major. They can be grouped in beams, e.g. [d2f2][ce][df] but there should be no spaces within a chord. See the tune 'Kitchen Girl' in the file Reels.abc for a simple example.

If the chord contains two notes both of the same length and pitch, such as [DD], then it is a unison (e.g. a note played on two strings of a violin simultaneously) and is shown as note-head with both upward and downward stems.

The old notation, e.g. +CEGc+, can still be used by setting a user switch – see §2.3.1.

2.2.15 Guitar chords

Guitar chords can be put in under the melody line by enclosing the chord in inverted commas, e.g. "Am7"A2D2 . See the tune 'William and Nancy' in English.abc for an example.

2.2.16 Order of symbols

The order of symbols for one note is <guitar chords>, <accents> (e.g. roll, staccato marker or up/downbow), <accidental>, <note>, <octave>, <note length>, i.e. ~^c'3 or even "Gm7"v.=G,2

Tie symbols, -, should come immediately after a note group but may be followed by a space, i.e. =G,2- . Open and close chord symbols, [], should enclose entire note sequences (except for guitar chords), i.e. "C"[CEGc] or "Gm7"[.=G,^c'] and open and close slur symbols, (), should do likewise, i.e. "Gm7"(v.=G,2~^c'2)

2.2.17 Comments

A % symbol will cause the remainder of any input line to be ignored. The file English.abc contains plenty of examples.

2.2.18 New notation

The letters `H-Z` can be used to define your own new notation within a tune. Currently the way they are implemented (if at all) is extremely package dependent and so users are advised not to rely too heavily on them to include new features. Instead, if there is a feature or symbol that you need and which is not available it is better to press for it to be included as a part of the language. See §2.3.2 for how to use these symbols with ABC2MTEX.

2.2.19 Line breaking & justification

Generally one line of `abc` notation will produce one line of music, although if the music is too long it will overflow onto the next line. This can look very effective, but it can also completely ruin ties across bar lines, for example. You can counteract this by changing either the note spacing with the `E:` field (although currently this is package dependent) or break the line of `abc` notation. If, however, you wish to use two lines of input to generate one line of music (see, for example, the 'Untitled Reel' in `Reels.abc`) then simply put a `\` at the end of the first line. This is also useful for changing meter or key in the middle of a line of music.

With most packages lines of music are right-justified. However, where this is not the case (e.g. when using MusicTEX), a `*` at the end of each line of `abc` notation will force a right-justified line-break.

2.3 ABC2MTEX extensions

The global accidentals which can be specified in the `K:` field are somewhat basic in their application. Firstly every specified note is marked with an accidental, rather than just the first of that note in each bar. Secondly, although global accidentals are overwritten by an accidental attached to a note this isn't done particularly intelligently. For example, for the key specification `K:G =f` every f is marked with a natural and if it was required to override one of these with a sharp, i.e. a `^f` within the abc of the tune, it would make sense for the output to have nothing by that f since the key signature automatically sharpens every f. However, in the current implementation, a sharp will appear.

By default, when using MusicTEX, lines of music are left-justified but not right-justified. There are two ways to overcome this; either with a `*` at the end of each line of `abc` notation as described above (§2.2.19) or by using the setting `justify` as described in §2.3.1.

Currently slurs and ties are only available with MusiXTEX (see §4.1).

The code which handles chords, e.g. `[ac]`, is a bit sensitive and you may need to fiddle around a bit with the order of the notes in the chord to get it looking right.

Within guitar chords ABC2MTEX only processes the first letter (and the second if it is a `#` or a `b`) of the character string in between the inverted commas and submits the rest to TEX as is. This means that in chords such as `"F#m"` or `"BbMaj7"` the sharp and flat symbols will be handled correctly and will even come out transposed correctly. Also, for chords with more than one or two letters, you may need to add some extra spacing (see §2.3.4) in the tune to prevent two chords from overlapping. Finally, by default the letters appear below the staff; this can be changed with the user settings, §2.3.1.

2.3.1 User settings

Various user defined switches can be set in the file `settings` to customise the behaviour of the output. To set a switch just put the name of the switch on a line in `settings`. The switches are:–

justify – when using MusicTEX this switch right-justifies every line of music.

gchords above – puts guitar chords above the staff rather than below.

autobeam – for many common meters, the code can, if required, try to ensure that beams contain the 'right' number of beats. For example, in 6/8 double jigs, each bar normally contains two beams of three eighth notes, or equivalent and so with autobeaming set, the input `abcdef` will produce exactly the same output as `abc def`; two beams of 3 eighth notes. By default, i.e. with autobeaming off, the input `abcdef` produces one beam of 6 eighth notes.

oldrepeats (for backwards compatibility) – gives the old output from `[1` & `[2` for both `[1`, `[2`, `|1` and `:|2`.

oldchords (for backwards compatibility) – forces the code to recognise the old notation `++` for chords as well as the new `[]` – e.g. `+CEGc+` will give the same output as `[CEGc]`.

oldslurs (for backwards compatibility) – forces the code to recognise the old notation `s.s` for slurs as well as the new `(.)` – e.g. `sDEFsG` will give the same output as `(DEFG)`.

2.3.2 New notation

The new notation letters, `H-Z`, have been provided because of demand by users to add extra symbols into the manuscript and each letter can generate 3 different commands, depending on how it is used.

In the simplest form, the input `S` will produce a `\userS` in the output; `\userS` should then be defined by the user either in `header.tex` or even in the `abc` file and could, say, be used to produce a horizontal space by defining `\def\userS{\qsk}`. In a similar manner, if using MusiXTEX, you could mark bars as the final repeats (in the same way as `|1` marks them as first repeats) by using `L|` and defining `\def\userL{\setvolta{last time}}`.

More complicated use occurs when the input letter is combined with a note (`ABCDEFGacbdefg`). In this case the letter `P`, say, produces either `\userPu{.}` or `\userPl{.}` depending on whether the following note is in an upper or lower beam. The argument `{.}` is the MusicTEX note pitch. An example use could be to mark a note with some accent, e.g. a V above the note head, and in this case the command would be defined

```
\def\userPl#1{\zcharnote#1{\raise5pt\hbox{$\vee$}}}%
```

N.B. To use a TEX command combined with a note, the command letter must be in the same place as the other note attributes (i.e. after a guitar chord and before an accidental – see §2.2.16). Thus, in the input `J A` or `J"Am"A`, the `J` will produce the output `\userJ`. Also it is possible to have more than one TEX command per note (e.g. `HIA` will produce two user commands combined with the note.

It is, of course, possible to write TEX commands for anything, including sequences of notes, but the author *strongly* recommends that they are not used for this purpose and, in general, used sparingly. The reasons for this are twofold; firstly one of the strengths of `abc` notation is that it can be fairly easily understood by humans – this may not be the case if it is peppered with extra letters. Secondly, the user commands will be ignored by other packages such as `playabc`, so any notes or other musical marks normally understood by `playabc` will be present in the printed output but missing from the sound output.

2.3.3 Internote spacings

The internote spacing is set by the information field `E`. As the format is currently set up, `E:8` & `E:7` can be used to squeeze long tunes up a bit and `E:10` and above to stretch short tunes. Using `E:6` really looks a bit too cramped.

2.3.4 TEX input

If there is a line in a tune file beginning with a `\`, it is put directly into the output file (`music.tex`). For example, you can use `\qsk` to get small horizontal space within a tune and `\bigskip` or `\medskip` to get vertical space. This is acceptable input anywhere except within tune headers.

2.4 Examples

Examples are provided for most of the possibilities above. They can be found in:–

Example	File & Number	Section
Information fields	`English:3`	2.1
Broken Rhythm	`Strspys:1`	2.2.4
Triplets	`Strspys:1`	2.2.5
First & Second Repeats	`Jigs:1`	2.2.8
Line Breaking	`English:1`	2.2.19
Changing key mid-tune	`Reels:1`	2.2.10
Changing meter mid-tune	`English:2`	2.2.10
Changing default note length mid-tune	`English:3`	2.2.10
Gracings	`Strspys:2`	2.2.12
Chords	`Reels:2`	2.2.14
Guitar Chords	`English:3`	2.2.15
Internote spacings	`Strspys:2`	2.3.3

3 Running `abc2mtex`

Run the program with the command `abc2mtex`. You will be prompted to choose tunes with the line

```
select tunes:
```

Typically you might respond with `English:1-3,5,9-` which will select tunes numbered 1, 2, 3, 5 and 9 onwards from the file `English`. Just entering `English` or `English:-` will select all the all the tunes from `English`. Note that there should be no spaces anywhere in this input.

Alternatively, all the input can go in the command line, e.g. `abc2mtex English:1-3,5,9- Jigs:-10` and the program will even read from the standard input, e.g. `abc2mtex -`, which is useful to pipe in the results of a search (see `index.tex`). Also, if the filename extension is `.abc` then you don't need to type that in – e.g. the input `Reels:1-10` will open the file `Reels.abc` (if the file `Reels` doesn't exist). When the program has finished formatting the chosen tunes (a matter of seconds) it will prompt you again with a `select tunes`. To quit the program simply hit return (or `q` or `quit`) at this prompt.

Finally, if you want to keep processing the same tunes over and over again you can store a list of their files and reference numbers in another file and run the code with this file as an input (to save typing it in again). There is an example in the following section.

3.1 TEX input

If at the prompt

```
select tunes:
```

you enter a line beginning with a `\`, it is put directly into the output file (`music.tex`). This is a good way of putting things like page headings directly into the document, e.g. something like

```
\headline{ENGLISH TUNES \hfil \folio}
```

will generate a header and page number for each page.

This sort of input can then be stored in a file to save the format of particular documents. For example, you could store

```
\headline{FLUTE MUSIC \hfil \folio}
\centerline{REELS}
Reels:1-9
\vfill\eject
\centerline{JIGS}
Jigs:1-4
```

in a file and then run

```
abc2mtex < file
```

Note that the \vfill\eject causes a page break and the \folio generates page numbers.

3.2 Command line options

By default, abc2mtex outputs MusicTEX to the file music.tex but this can be changed with the use of command line options. Thus abc2mtex -i creates an index of all the files in the file index and can even include the first two bars of each tune in abc notation (see index.tex), while abc2mtex -x produces MusiXTEX output in music.tex (see §4.1). The option -t will transpose some or all of the tunes, §3.3. The output file can be set with the -o option, e.g. abc2mtex -o Jigs.tex Jigs.abc. It also possible to use combinations of options, e.g. abc2mtex -x -t to produce transposed MusiXTEX output, but they must all come before the first input filename.

3.3 Transposing tunes

If you select the option -t, you will be prompted at every tune by

```
Transpose?
```

You should enter here the change in the number of sharps and flats you wish to make. For example, if the tune is in G (1 sharp) and you wish to transpose it up to A (3 sharps) you would enter +2 or just 2. Flats are counted negatively so that to transpose from G to F (1 flat or −1) you would enter -2.

If you have chosen to transpose the tune, i.e. entered anything other than 0 you will then be prompted by

```
Note offset?
```

which is just the number of line and spaces that each note must move. Thus going from G to A is just moving every note up one and so you would enter +1 or 1 whilst G to F would require -1. Note that something like changing from E$\flat$ (3 flats) to E (4 sharps) has note offset 0.

Alternatively, at the Transpose? prompt, you can enter the interval, e.g. ^5 means transpose up a fifth. Possible intervals are shown in the table below.

input	transposition	change in $\sharp/\flat$'s	note offset
^2	up tone	+2	+1
^5	up fifth	+1	+4
_4	down fourth	+1	−3
^8	up octave	0	+7
_8	down octave	0	−7
^4	up fourth	−1	+3
_5	down fifth	−1	−4
_2	down tone	−2	−1

As well as generating transposed MusicTEX output in the file music.tex, the -t option also puts transposed abc notation in the file transpose.abc. This is useful if you want to save the transposed version and abc2mtex can be run on this file.

Finally to transpose all of the selected input tunes the transposition and note offset can be entered on the command line with abc2mtex -t:<transpose>:<note offset> or abc2mtex -t:<interval>. For example, to transpose everything down a fourth, use abc2mtex -t:_4 or abc2mtex -t:1:-3

3.4 Running MusicTeX

Once you have created `music.tex`, simply run `tex music.tex`. This will produce a file `music.dvi` which can be viewed (e.g. with `xdvi`) or converted into postscript (e.g. with `dvips`). The viewing/printing will depend on your system.

3.5 Customising the output

The file `header.tex` contains most of the parameters which decide the way the printed music looks. If you want to customise the music output this is the first place you should start. Brief instructions are contained in this file and an example of how it could be set up contained in `header1.tex`. To try this out copy `header.tex` to somewhere safe and then copy `header1.tex` to `header.tex`. Experiment!

3.6 Using make

If you are on a system that has the `make` command there are a number of shortcuts available. For example, entering `make English.ps` will run `abc2mtex`, TeX and `dvips` on `English.abc` and put the results in `English.ps` (though you may need to modify the Makefile to suit your system). To tidy up at the end of a session, simply type `make tidy`, although be warned that this will remove the files `music.tex`, `transpose.abc` and `index`.

4 Extensions

4.1 MusiXTeX

MusiXTeX is a recent enhancement to MusicTeX available at various ftp sites (see the `README` file). Currently it is in beta test phase and there are two versions available; see the file `Changes` for compatibility of versions. It is a three pass system – you run `tex music`, then `musicflx music` and then `tex music` again – and this extra processing allows some nice features to be built in such as proper slurs. It also runs considerably faster than MusicTeX. The `abc` notation isn't affected but the main changes in the output should be that:

- The music is automatically right-justified.
- Tuplets will have a slur marker by them and other slurs (see §2.2.11) are now possible.

To generate MusiXTeX output simply run `abc2mtex` with the command line option `-x`.

4.2 Multi-stave output

A simple amendment has been made to the code to allow the generation of multi-stave music. Specifically, the character `&` is carried straight through to the TeX output and the characters `&&` produce a `\enotes\notes` pair. Thus the input `DEFG & ABcd && A4 & e2 c2|` produces:

To explain this to those unfamiliar with MusicTeX, the `DEFG` are put on the lowest stave. The `&` then tells MusicTeXto move up a stave, where it puts the `ABcd`. The first notes of each group are aligned. The `&&` (or a bar line) moves the output back down to the lowest stave and resets the alignment, so that in this case, the `A4` is on the lower stave, and is aligned with the `e2` on the upper stave.

The number of staves and whether or not they are bass or treble clefs is specified by a simple amendment to the `K:` field. Thus `K:G 1&3` means 4 staves with the bottom one in the bass clef; `K:G 0&2` means 2 staves both in the treble clef. Notes on the bass clef are automatically shifted down one octave. The `i&j` part of the `K:` field should come after any mode/scale specification and before lists of global accidentals, e.g. `K:D aeolian 1&3 ^f`.

Note that this is regarded as an essentially unsupported feature put in to take some of the pain out of coding MusicTEX. Several things may not work properly, such as bar counting or slurs & ties (although it would not be too difficult to make them work if there was the demand) and it is likely that, for all but the simplest melodies the raw TEX output will need editing. Also, because `abc` notation in this form is both difficult to write and read, it is probable that if multi-stave music becomes properly a part of ABC2MTEX it will be in the form already available with `playabc` (using `I:part 1`, ..., `I:part final`).

5 Bugs and features

Error handling is reasonably good; the code won't accept characters it doesn't recognise and it also tries to give helpful error messages. It exits if it finds the unescaped special characters `# &` in an information field since TEX will choke on them. They can, however, be used by escaping with a backslash (`\`) – e.g. use `\#` rather than `#`.

The code will also attempt to check the length of each bar. It ignores leading notes and bars that finish with a repeat or double bar symbol, but for other bars which are too long or too short it gives a warning indicating the bar and line number. The bar numbering is a little quirky; the first **complete** bar of a line should be number 1 (so that leading notes are bar number 0). It can correctly handle duplets, triplets and quadruplets but quietly ignores bars with other tuplets.

6 The future

Sometime in the future it is hoped to provide:

- Alignment of music & lyrics – this may be **difficult** and will have to wait until MusiXTEX has settled down.
- A proper front end parser (probably using Don Ward's lex parser for `playabc`).