

The abc music standard 2.1 (Dec 2011)

Contents

- 1. Introduction
 - 1.1 How to read this document
 - * 1.1.1 Terminology / definitions
 - 1.2 How to avoid reading this document
 - 1.3 Abc tutorials
 - 1.4 Abc extensions
 - 1.5 Further information and changes
 - 1.6 Document locations
- 2. Abc files, tunes and fragments
 - 2.1 Abc file identification
 - 2.2 Abc file structure
 - * 2.2.1 Abc tune
 - * 2.2.2 File header
 - * 2.2.3 Free text and typeset text
 - * 2.2.4 Empty lines and line-breaking
 - * 2.2.5 Comments and remarks
 - * 2.2.6 Continuation of input lines
 - 2.3 Embedded abc and abc fragments
 - * 2.3.1 Embedded abc fragment
 - * 2.3.2 Embedded abc tune
 - * 2.3.3 Embedded file header
 - * 2.3.4 Embedded abc file
- 3. Information fields
 - 3.1 Description of information fields
 - * 3.1.1 X: - reference number
 - * 3.1.2 T: - tune title
 - * 3.1.3 C: - composer

- * 3.1.4 O: - origin
- * 3.1.5 A: - area
- * 3.1.6 M: - meter
- * 3.1.7 L: - unit note length
- * 3.1.8 Q: - tempo
- * 3.1.9 P: - parts
- * 3.1.10 Z: - transcription
- * 3.1.11 N: - notes
- * 3.1.12 G: - group
- * 3.1.13 H: - history
- * 3.1.14 K: - key
- * 3.1.15 R: - rhythm
- * 3.1.16 B:, D:, F:, S: - background information
- * 3.1.17 I: - instruction
- * 3.1.18 Other fields
- 3.2 Use of fields within the tune body
- 3.3 Field continuation
- 4. The tune body
 - 4.1 Pitch
 - 4.2 Accidentals
 - 4.3 Note lengths
 - 4.4 Broken rhythm
 - 4.5 Rests
 - 4.6 Clefs and transposition
 - 4.7 Beams
 - 4.8 Repeat/bar symbols
 - 4.9 First and second repeats
 - 4.10 Variant endings
 - 4.11 Ties and slurs
 - 4.12 Grace notes
 - 4.13 Duplets, triplets, quadruplets, etc.

- 4.14 Decorations
 - 4.15 Symbol lines
 - 4.16 Redefinable symbols
 - 4.17 Chords and unisons
 - 4.18 Chord symbols
 - 4.19 Annotations
 - 4.20 Order of abc constructs
- 5. Lyrics
 - 5.1 Alignment
 - 5.2 Verses
 - 5.3 Numbering
- 6. Typesetting and playback
 - 6.1 Typesetting
 - * 6.1.1 Typesetting line-breaks
 - * 6.1.2 Typesetting extra space
 - * 6.1.3 Typesetting information fields
 - 6.2 Playback
- 7. Multiple voices
 - 7.1 Voice properties
 - 7.2 Breaking lines
 - 7.3 Inline fields
 - 7.4 Voice overlay
- 8. abc data format
 - 8.1 Tune body
 - 8.2 Text strings
- 9. Macros
 - 9.1 Static macros
 - 9.2 Transposing macros
- 10. Outdated syntax
 - 10.1 Outdated information field syntax
 - 10.2 Outdated dialects

- * 10.2.1 Outdated line-breaking
 - * 10.2.2 Outdated decorations
 - * 10.2.3 Outdated chords
 - 10.3 Outdated continuations
 - 10.4 Outdated directives
 - 10.5 Outdated file structure
 - * 10.5.1 Outdated tune header syntax
 - * 10.5.1 Outdated defaults
 - 10.6 Outdated lyrics alignment
 - 10.7 Other outdated syntax
 - * 10.7.1 Disallowed voice overlay
- 11. Stylesheet directives and pseudo-comments
 - 11.0 Introduction to directives
 - * 11.0.1 Disclaimer
 - * 11.0.2 Stylesheet directives
 - 11.1 Voice grouping
 - 11.2 Instrumentation directives
 - 11.3 Accidental directives
 - 11.4 Formatting directives
 - * 11.4.1 Page format directives
 - * 11.4.2 Font directives
 - * 11.4.3 Space directives
 - * 11.4.4 Measure directives
 - * 11.4.5 Text directives
 - * 11.4.6 Information directives
 - * 11.4.7 Separation directives
 - * 11.4.8 Miscellaneous directives
 - 11.5 Application specific directives
 - 11.6 Further information about directives
- 12. Dialects, strict / loose interpretation and backwards compatibility
 - 12.1 Dialect differences

- * 12.1.1 Line-breaking dialects
 - * 12.1.2 Decoration dialects
 - * 12.1.3 Chord dialects
 - 12.2 Loose interpretation
 - 12.3 Strict interpretation
 - 13. Sample abc tunes
 - 13.1 English.abc
 - 13.2 Strspys.abc
 - 13.3 Reels.abc
 - 13.4 Canzonetta.abc
 - 14. Appendix
 - 14.1 Supported accents & ligatures
 - 14.2 Errata
-

1. Introduction

Abc is a text-based music notation system designed to be comprehensible by both people and computers. Music notated in abc is written using characters - letter, digits and punctuation marks - on paper or in computer files.

This description of abc has been created for those who wish to understand the notation, and for implementers of abc software applications. Some example tunes are included in sample abc tunes.

1.1 How to read this document

Start at the beginning and work through to the end. Alternatively, for selected highlights, take a look at how to avoid reading this document.

1.1.1 Terminology / definitions

Note that the following terms have specific meanings in the context of the abc standard. For convenience, each time one of these terms is used in the standard it is linked to the section in which it is defined:

- abc file
- abc fragment

- abc tune
- abc tunebook
- code line-break
- comment
- embedded
- empty line
- file header
- free text
- information field
- inline field
- music code
- score line-break
- stylesheet directive
- text string
- tune body
- tune header
- typeset text

Please see also <http://www.ietf.org/rfc/rfc2119.txt> for formal definitions of the key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL.

Finally, the word *VOLATILE* is used to indicate sections which are under active discussion and/or likely to change in some future version of the standard.

1.2 How to avoid reading this document

The abc standard contains a lot of information, much of which will not be immediately useful to the beginner. Apart from reading this section, 1. Introduction, newcomers are recommended to familiarise themselves with all of 2.2 Abc file structure, 3.0 Information fields, a few subsections in 3.1 Description of information fields (in particular 3.1.1, 3.1.2, 3.1.6, 3.1.7 and 3.1.14), 3.2 Use of fields within the tune body, and as much of section 4. The tune body as is desired (but in particular 4.1, 4.3, 4.7, 4.8).

Newcomers are also advised to take a look at section 13. Sample abc tunes and one of the abc tutorials that is available.

After that, it may depend on what you want to use abc for, but further reading suggestions would be:

- 5. Lyrics for transcribing songs
- 6.1 Typesetting for printing abc transcriptions in staff notation
- 7. Multiple voices for working with multi-voice music

1.3 Abc tutorials

This document is also best read in conjunction with an introduction to abc notation. Several are available - see, for example:

- <http://abcnotation.com/learn> - a number of tutorials are linked from here
- <http://abcplus.sourceforge.net/#ABCGuide>
- http://www.lesession.co.uk/abc/abc_notation.htm
- <http://trillian.mit.edu/~jc/music/abc/doc/ABCTutorial.html>

1.4 Abc extensions

Since the abc notation system was originally written, a large number of abc software packages (programs which: produce printed sheet music; play or create audio files, usually MIDI; search or organise tune databases; or that analyse or manipulate tunes in some way) have been developed. However, not all of them follow this standard absolutely. This document aims at solving, or at least reducing, the problem of incompatibility between applications.

Nevertheless, when using abc it is good to be aware of the existence of such extensions. Extensions implemented by some major abc packages are described at the following links:

- <http://moinejf.free.fr/abcm2ps-features.txt> - extensions implemented by abcm2ps
- <http://abc.sourceforge.net/standard/abc2midi.txt> - extensions implemented by abc2midi
- <http://www.barfly.dial.pipex.com/bfextensions.html> - extensions implemented by BarFly
- <http://www.lautengesellschaft.de/cdmm/userguide/userguide.html> - extensions implemented by abctab2ps

1.5 Further information and changes

Questions about this standard, or abc in general, can be addressed to the abusers e-mail list, or the abcnotation forums:

- <http://groups.yahoo.com/group/abusers/> (abusers - subscriptions and archive of posts)
- <http://www.mail-archive.com/abusers@argyll.wisemagic.com/> (abusers - archive of old posts)
- <http://abcnotation.com/forums/>

To propose changes to the standard, please read

- <http://abcnotation.com/wiki/abc:standard:route-map> - a route map of proposed changes to the standard plus instructions for proposing changes

1.6 Document locations

This document can be found at:

- <http://abcnotation.com/wiki/abc:standard:v2.1>

The latest version of the standard, plus links to older versions and other developmental work, can always be found via:

- <http://abcnotation.com/wiki/abc:standard>
-

2. Abc files, tunes and fragments

Tunes written in abc are normally stored in abc files, either on a computer's hard-drive or linked from a web-page. However, an increasing number are found on web-pages or in databases.

This section describes the basic structure of abc files and abc tunes, as well as a definition for including fragments of abc tunes elsewhere (e.g. web-pages).

2.1 Abc file identification

All abc files should have the extension “abc” (all lower-case) on all platforms.

Comment: Some web-servers only allow a limited selection of file types; in this case a “txt” extension is the best alternative.

Every abc file should begin with the string %abc. An optional version number may follow on the same line, e.g.

%abc-2.1

Version numbers of 2.1 or higher indicate that the abc file is to be interpreted strictly according to the corresponding abc standard; if the version number is missing, the file will be treated under loose interpretation. The version field may also be used to indicate abc versions for individual tunes.

Note for developers: Software should ignore the byte order mark (BOM) if encountered as the first character of the file.

When an abc file is included in a multi-part e-mail, its MIME type must be “text/vnd.abc” (see IANA text/vnd.abc).

2.2 Abc file structure

An **abc file** consists of one or more abc tune transcriptions, optionally interspersed with free text and typeset text annotations. It may optionally start with a file header to set up default values for processing the file.

The file header, abc tunes and text annotations are separated from each other by empty lines (also known as blank lines).

An abc file with more than one tune in it is called an **abc tunebook**.

2.2.1 Abc tune

An **abc tune** itself consists of a tune header and a tune body, terminated by an empty line or the end of the file. It may also contain comment lines or stylesheet directives.

The **tune header** is composed of several information field lines, which are further discussed in information fields. The tune header should start with an **X:**(reference number) field followed by a **T:**(title) field and finish with a **K:**(key) field.

The **tune body**, which contains the music code, follows immediately after. Certain fields may also be used inside the tune body - see use of fields within the tune body.

It is legal to write an abc tune without a tune body. This feature can be used to document tunes without transcribing them.

Abc **music code** lines are those lines in the tune body which give notes, bar lines and other musical symbols - see the tune body for details. In effect, music code is the contents of any line which is not an information field, stylesheet directive or comment line.

2.2.2 File header

The file may optionally start with a **file header** (immediately after the version field), consisting of a block of consecutive information fields, stylesheet directives, or both, terminated with an empty line. The file header is used to set default values for the tunes in the file.

The file header may only appear at the beginning of a file, not between tunes.

Settings in a tune may override the file header settings, but when the end of a tune is reached the defaults set by the file header are reinstated.

Applications which extract separate tunes from a file must insert the fields of the original file header into the header of the extracted tune. However, since users may manually extract tunes without regard to the file header, it is not recommended to use a file header in an abc tunebook that is to be distributed.

2.2.3 Free text and typeset text

The terms free text and typeset text refer to any text not directly included within the information fields in a tune header. Typically such text is used for annotating abc tunebooks; free text is for annotating the abc file but is not included in the typeset score, whereas typeset text is intended for printing out.

Free text is just that. It can be included anywhere in an abc file, after the file header, but must be separated from abc tunes, typeset text and the file header by empty lines. Typically it is used for annotating the abc file but in principle can be any text not containing information fields.

Comment: Since raw html markup and email headers are treated as free text (provided they don't inadvertently contain information fields) this means that abc software can process a wide variety of text-based input files just by ignoring non-abc code.

By default free text is not included in the printed score, although typesetting software may offer the option to print it out (e.g. via a command line switch or GUI checkbox). In this case, the software should treat the free text as a text string, but may format it in any way it chooses.

Typeset text is any text specified using text directives. It may be inserted anywhere in an abc file after the file header, either separated from tunes by empty lines, or included in the tune header or tune body.

Typeset text should be printed by typesetting programs although its exact position in the printed score is program-dependent.

Typeset text that is included in an abc tune (i.e. within the tune header or tune body), must be retained by any programs, such as databasing software, that splits an abc file into separate abc tunes.

2.2.4 Empty lines and line-breaking

Empty lines (also known as blank lines) are used to separate abc tunes, free text and the file header. They also aid the readability of abc files.

Lines that consist entirely of white-space (space and tab characters) are also regarded as empty lines.

Line-breaks (also known as new lines, line feeds, carriage returns, end-of-lines, etc.) can be used within an abc file to aid readability and, if required, break up long input lines - see continuation of input lines.

More specifically, line-breaks in the music code can be used to structure the abc transcription and, by default, generate line-breaks in the printed music. For more details see typesetting line-breaks.

2.2.5 Comments and remarks

A percent symbol (%) will cause the remainder of any input line to be ignored. It can be used to add a **comment** to the end of an abc line or as a **comment line** in its own right. (To get a percent symbol, type \% - see text strings.)

Alternatively, you can use the syntax [r:remark] to write a **remark** in the middle of a line of music.

Example:

```
|:DEF FED| % this is an end of line comment
% this is a comment line
DEF [r:and this is a remark] FED:|
```

Abc code which contains comments and remarks should be processed in exactly the same way as it would be if all the comments and remarks were removed (although, if the code is preprocessed, and comments are actually removed, the stylesheet directives should be left in place).

Important clarification: lines which just contain a comment are processed as if the entire line were removed, even if the comment is preceded by white-space (i.e. the % symbol is the not first character). In other words, removing the comment effectively removes the entire line and so no empty line is introduced.

2.2.6 Continuation of input lines

It is sometimes necessary to tell abc software that an input line is continued on the next physical line(s) in the abc file, so that the two (or more) lines are treated as one. In abc 2.0 there was a universal continuation character (see outdated continuations) for this purpose, but it was decided that this was both unnecessary and confusing.

In abc 2.1, there are ways of continuing each of the 4 different input line types: music code, information fields, comments and stylesheet directives.

In abc music code, by default, line-breaks in the code generate line-breaks in the typeset score and these can be suppressed by using a backslash (or by telling abc typesetting software to ignore line-breaks using `I:linebreak $` or `I:linebreak <none>`) - see typesetting line-breaks for full details.

Comment for programmers: The backslash effectively acts as a continuation character for music code lines, although, for those used to encountering it in other computer language contexts, its use is very abc-specific. In particular it can continue music code lines through information fields, comments and stylesheet directives.

The 3 other input line types can be continued as follows:

- information fields can be continued using `+:` at the start of the following line - see field continuation;
- comments can easily be continued by adding a `%` symbol at the start of the following line - since they are ignored by abc software it doesn't matter how many lines they are split into;
- most stylesheet directives are too short to require a continuation syntax, but if one is required then use the `I:<directive>` form (see `I:instruction`), in place of `%%<directive>` and continue the line as a field - see field continuation.

Comment for developers: Unlike other languages, and because of the way in which both information fields and music code can be continued through comments, stylesheet directives and (in the case of music code) information fields, it is generally not possible to parse abc files by pre-processing continuations into single lines.

Note that, with the exception of abc music code, continuations are unlikely to be needed often. Indeed in most cases it should be possible, although not necessarily desirable, to write very long input lines, since most abc editing software will display them as wrapped within the text editor window.

Recommendation: Despite there being no limit on line length in abc files, it is recommended that users avoid writing abc code with very long lines. In particular, judiciously applied line-breaks can aid the (human) readability of abc code. More importantly, users who send abc tunes with long lines should be aware that email software sometimes introduces additional line-breaks into lines with more than 72 characters and these may even cause errors when the resulting tune is processed.

2.3 Embedded abc and abc fragments

Traditionally abc has been used in dedicated abc files. More recently, however, the possibility has arisen to include abc tunes, and even fragments, within other document types. An abc element included within another document type is referred to as **embedded** in that document.

Often, although not always, some form of markup is used to indicate where the embedded abc code starts and finishes.

Example: Within an html document a tune could be included as follows:

```
<pre class="abc-tune">
X:1
T:Title
K:C
DEF FED:|
</pre>
```

Important note: The abc standard makes no stipulation about *how* the abc code is included in the document. For example, in html it could be via a `<pre>`, `<div>`, `<object>`, `<script>` or some other tag.

Embedded abc elements can be one of four types:

- an abc fragment
- an abc tune
- a file header
- an entire abc file

In all cases, the type must be indicated to the abc parsing code which is going to process it (for example, via a `class` parameter). An exception is the embedded abc tune where the parser may instead use the `X:` field to identify it.

The following rules are applied to embedded elements:

2.3.1 Embedded abc fragment

An **abc fragment** is a partial abc tune. It may contain a partial tune header with no body or a tune body with optional tune header information fields.

Example 1: A fragment with no tune header:

```
<div class="abc-fragment">
CDEF GABc|
</div>
```

Example 2: A fragment with a partial tune header:

```

<div class="abc-fragment">
T:Major scale in D
K:D
DEFG ABcd|
</div>

```

Unless T:, M: and K: fields are present, a fragment is assumed to describe a stave in the treble clef with no title, no meter indication and no key signature, respectively.

An abc fragment does not require an empty line to mark the end of the tune body if it is terminated by the document markup.

Note for developers: For processing as an abc tune, the parsing code is notionally assumed to add empty X:, T: and K: fields, if these are missing. However, since the processing generally takes place internally within a software package, these need not be added in actuality.

2.3.2 Embedded abc tune

An embedded abc tune has the same structure as an ordinary abc tune except that it does not require an empty line to mark the end of the tune body.

An embedded abc tune could also be identified as an abc fragment (albeit complete), if preferred.

2.3.3 Embedded file header

As with the file header, an embedded file header can be used to set default values for all embedded abc tunes and abc fragments within the document.

Example: For setting the title font in every abc tune in the document:

```

<div class="abc-file-header">
%%titlefont Arial 10
</div>

```

Like its counterpart, there must only be one embedded file header per document and it should precede all other embedded abc tunes and abc fragments.

2.3.4 Embedded abc file

A document may include an entire embedded abc file with the usual structure - see abc file structure.

An embedded abc file should be treated independently from other embedded elements so that settings in one embedded abc file do not affect other embedded elements.

Recommendation: As a consequence, using other embedded elements in a document that contains an embedded abc file is not recommended.

3. Information fields

Any line beginning with a letter in the range A-Z or a-z and immediately followed by a colon (:) is an **information field**. Information fields are used to notate things such as composer, meter, etc. In fact anything that isn't music.

An information field may also be inlined in a tune body when enclosed by [and] - see use of fields within the tune body.

Many of these information field identifiers are currently unused so, in order to extend the number of information fields in the future, programs that comply with this standard must ignore the occurrence of information fields not defined here (although they should give a non-fatal error message to warn the user, in case the field identifier is an error or is unsupported).

Some information fields are permitted only in the file or tune header and some only in the tune body, while others are allowed in both locations. information field identifiers A-G, X-Z and a-g, x-z are not permitted in the body to avoid confusion with note symbols, rests and spacers.

Users who wish to use abc notation solely for transcribing (rather than documenting) tunes can ignore most of the information fields. For this purpose all that is really needed are the X:(reference number), T:(title), M:(meter), L:(unit note length) and K:(key) information fields, plus if applicable C:(composer) and w: or W: (words/lyrics, respectively within or after the tune).

Recommendation for newcomers: A good way to find out how to use the fields is to look at the example files, sample abc tunes (in particular English.abc), and try out some examples.

The information fields are summarised in the following table and discussed in description of information fields and elsewhere.

The table illustrates how the information fields may be used in the tune header and whether they may also be used in the tune body (see use of fields within the tune body for details) or in the file header (see abc file structure).

Abc Fields and their usage:

Field name	file header	tune header	tune body	inline	type	Examples and notes
A:area	yes	yes			string	A:Donegal, A:Bampton (
B:book	yes	yes			string	B:O'Neill's
C:composer	yes	yes			string	C:Robert Jones, C:Trad.
D:discography	yes	yes			string	D:Chieftains IV

Field name	file header	tune header	tune body	inline	type	Examples and notes
F:file url	yes	yes			string	F:http://a.b.c/file.abc
G:group	yes	yes			string	G:flute
H:history	yes	yes			string	H:The story behind this
I:instruction	yes	yes	yes	yes	instruction	I:papersize A4, I:newpage
K:key		last	yes	yes	instruction	K:G, K:Dm, K:AMix
L:unit note length	yes	yes	yes	yes	instruction	L:1/4, L:1/8
M:meter	yes	yes	yes	yes	instruction	M:3/4, M:4/4
m:macro	yes	yes	yes	yes	instruction	m: ~G2 = {A}G{F}G
N:notes	yes	yes	yes	yes	string	N:see also O’Neills - 234
O:origin	yes	yes			string	O:UK; Yorkshire; Bradfo
P:parts		yes	yes	yes	instruction	P:A, P:ABAC, P:(A2B)3
Q:tempo		yes	yes	yes	instruction	Q:“allegro” 1/4=120
R:rhythm	yes	yes	yes	yes	string	R:R, R:reel
r:remark	yes	yes	yes	yes	-	r:I love abc
S:source	yes	yes			string	S:collected in Brittany
s:symbol line			yes		instruction	s: !pp! ** !f!
T:tune title		second	yes		string	T:Paddy O’Rafferty
U:user defined	yes	yes	yes	yes	instruction	U: T = !trill!
V:voice		yes	yes	yes	instruction	V:4 clef=bass
W:words		yes	yes		string	W:lyrics printed after the
w:words			yes		string	w:lyrics printed aligned w
X:reference number		first			instruction	X:1, X:2
Z:transcription	yes	yes			string	Z:John Smith, <j.s@mail

Fields of type ‘string’ accept text strings as argument. Fields of type ‘instruction’ expect a special instruction syntax which will be detailed below. The contents of the remark field will be totally ignored.

Repeated information fields

All information fields, with the exception of **X:**, may appear more than once in an abc tune.

In the case of all string-type information fields, repeated use in the tune header can be regarded as additional information - for example, a tune may be known by many titles and an abc tune transcription may appear at more than one URL (using the **F:** field). Typesetting software which prints this information out may concatenate all string-type information fields of the same kind, separated by semi-colons (;), although the initial **T:**(title) field should be treated differently, as should **W:**(words) fields - see typesetting information fields.

Certain instruction-type information fields, in particular **I:**, **m:**, **U:** and **V:**, may also be used multiple times in the tune header to set up different instructions,

macros, user definitions and voices. However, if two such fields set up the same value, then the second overrides the first.

Example: The second `I:linebreak` instruction overrides the first.

```
I:linebreak <EOL>
I:linebreak <none>
```

Comment: The above example should not generate an error message. The user may legitimately wish to test the effect of two such instructions; having them both makes switching from one to another easy just by changing their order.

Other instruction-type information fields in the tune header also override the previous occurrence of that field.

Within the tune body each line of code is processed in sequence. Therefore, with the exception of `s:` (symbol line), `w:` (words) and `W:` (words) which have their own syntax, the same information field may occur a number of times, for example to change key, meter, tempo or voice, and each occurrence has the effect of overriding the previous one, either for the remainder of the tune, or until the next occurrence. See use of fields within the tune body for more details.

Order of information fields

Recommendation for users: Although information fields in the tune header may be written in any order (subject to `X:`, `T:` and `K:` coming first, second and last, respectively), it does make sense for users to stick to a common ordering, if for no other reason than it makes public domain abc code more readable. Typical ordering of the tune header puts fundamental tune identification details first (`X`, `T`, `C`, `O`, `R`), with information fields relating to how the tune is played last (`P`, `V`, `M`, `L`, `Q`, `K`). Background information (`B`, `D`, `F`, `G`, `H`, `N`, `S`, `Z`) and information on how the abc code should be interpreted (`I`, `m`, `U`) then tends to appear in the middle of the tune header. Words (`W`) may be included in the tune header but are usually placed at the end of the tune body.

3.1 Description of information fields

3.1.1 `X:` - reference number

The `X:` (reference number) field is used to assign to each tune within a tunebook a unique reference number (a positive integer), for example: `X:23`.

The `X:` field is also used to indicate the start of the tune (and hence the tune header), so all tunes must start with an `X:` field and only one `X:` field is allowed per tune.

The `X:` field may be empty, although this is not recommended.

Recommendation for developers: Software which writes abc files is recommended to offer users the possibility to manage **X:** field numbering automatically. GUI applications may even hide the **X:** field from users although they should always allow the user access to the raw abc file.

3.1.2 T: - tune title

A **T:** (title) field must follow immediately after the **X:** field; it is the human identifier for the tune (although it may be empty).

Some tunes have more than one title and so this field can be used more than once per tune to indicate alternative titles.

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.

See typesetting information fields for details of how the title and alternatives are included in the printed score.

3.1.3 C: - composer

The **C:** field is used to indicate the composer(s).

See typesetting information fields for details of how the composer is included in the printed score.

3.1.4 O: - origin

The **O:** field indicates the geographical origin(s) of a tune.

If possible, enter the data in a hierarchical way, like:

O:Canada; Nova Scotia; Halifax.

O:England; Yorkshire; Bradford and Bingley.

Recommendation: It is recommended to always use a ";" (semi-colon) as the separator, so that software may parse the field. However, abc 2.0 recommended the use of a comma, so legacy files may not be parse-able under abc 2.1.

This field may be especially useful for traditional tunes with no known composer.

See typesetting information fields for details of how the origin information is included in the printed score.

3.1.5 A: - area

Historically, the **A:** field has been used to contain area information (more specific details of the tune origin). However this field is now deprecated and it is recommended that such information be included in the **O:** field.

3.1.6 M: - meter

The **M:** field indicates the meter. Apart from standard meters, e.g. **M:6/8** or **M:4/4**, the symbols **M:C** and **M:C|** give common time (4/4) and cut time (2/2) respectively. The symbol **M:none** omits the meter entirely (free meter).

It is also possible to specify a complex meter, e.g. **M:(2+3+2)/8**, to make explicit which beats should be accented. The parentheses around the numerator are optional.

The example given will be typeset as:

$$\frac{2 + 3 + 2}{8}$$

When there is no **M:** field defined, free meter is assumed (in free meter, bar lines can be placed anywhere you want).

3.1.7 L: - unit note length

The **L:** field specifies the unit note length - the length of a note as represented by a single letter in abc - see note lengths for more details.

Commonly used values for unit note length are **L:1/4** - quarter note (crotchet), **L:1/8** - eighth note (quaver) and **L:1/16** - sixteenth note (semi-quaver). **L:1** (whole note) - or equivalently **L:1/1**, **L:1/2** (minim), **L:1/32** (demi-semi-quaver), **L:1/64**, **L:1/128**, **L:1/256** and **L:1/512** are also available, although **L:1/64** and shorter values are optional and may not be provided by all software packages.

If there is no **L:** field defined, a unit note length is set by default, based on the meter field **M:**. This default is calculated by computing the meter as a decimal: if it is less than 0.75 the default unit note length is a sixteenth note; if it is 0.75 or greater, it is an eighth note. For example, $2/4 = 0.5$, so, the default unit note length is a sixteenth note, while for $4/4 = 1.0$, or $6/8 = 0.75$, or $3/4 = 0.75$, it is an eighth note. For **M:C** (4/4), **M:C|** (2/2) and **M:none** (free meter), the default unit note length is 1/8.

A meter change within the body of the tune will not change the unit note length.

3.1.8 Q: - tempo

The **Q:** field defines the tempo in terms of a number of beats per minute, e.g. **Q:1/2=120** means 120 half-note beats per minute.

There may be up to 4 beats in the definition, e.g:

Q:1/4 3/8 1/4 3/8=40

This means: play the tune as if **Q:5/4=40** was written, but print the tempo indication using separate notes as specified by the user.

The tempo definition may be preceded or followed by an optional text string, enclosed by quotes, e.g.

Q: "Allegro" 1/4=120

Q: 3/8=50 "Slowly"

It is OK to give a string without an explicit tempo indication, e.g. Q:"Andante".

Finally note that some previous Q: field syntax is now deprecated (see outdated information field syntax).

3.1.9 P: - parts

VOLATILE: For music with more than one voice, interaction between the P: and V: fields will be clarified when multi-voice music is addressed in abc 2.2. The use of P: for single voice music will be revisited at the same time.

The P: field can be used in the tune header to state the order in which the tune parts are played, i.e. P:ABABCD, and then inside the tune body to mark each part, i.e. P:A or P:B. (In this context part refers to a section of the tune, rather than a voice in multi-voice music.)

Within the tune header, you can give instruction to repeat a part by following it with a number: e.g. P:A3 is equivalent to P:AAA. You can make a sequence repeat by using parentheses: e.g. P:(AB)3 is equivalent to P:ABABAB. Nested parentheses are permitted; dots may be placed anywhere within the header P: field to increase legibility: e.g. P:((AB)3.(CD)3)2. These dots are ignored by computer programs.

See variant endings and lyrics for possible uses of P: notation.

Player programs should use the P: field if possible to render a complete playback of the tune; typesetting programs should include the P: field values in the printed score.

See typesetting information fields for details of how the part information may be included in the printed score.

3.1.10 Z: - transcription

Typically the Z: field contains the name(s) of the person(s) who transcribed the tune into abc, and possibly some contact information, e.g. an (e-)mail address or homepage URL.

Example: Simple transcription notes.

Z:John Smith, <j.s@mail.com>

However, it has also taken over the role of the %%abc-copyright and %%abc-edited-by since they have been deprecated (see outdated directives).

Example: Detailed transcription notes.

```
Z:abc-transcription John Smith, <j.s@mail.com>, 1st Jan 2010
Z:abc-edited-by Fred Bloggs, <f.b@mail.com>, 31st Dec 2010
Z:abc-copyright &copy; John Smith
```

This new usage means that an update history can be recorded in collections which are collaboratively edited by a number of users.

Note that there is no formal syntax for the contents of this field, although users are strongly encouraged to be consistent, but, by convention, **Z:abc-copyright** refers to the copyright of the abc transcription rather than the tune.

See typesetting information fields for details of how the transcription information may be included in the printed score.

Comment: If required, software may even choose to interpret specific **Z:** strings, for example to print out the string which follows after **Z:abc-copyright**.

3.1.11 N: - notes

Contains general annotations, such as references to other tunes which are similar, details on how the original notation of the tune was converted to abc, etc.

See typesetting information fields for details of how notes may be included in the printed score.

3.1.12 G: - group

Database software may use this field to group together tunes (for example by instruments) for indexing purposes. It can also be used for creating medleys - however, this usage is not standardised.

3.1.13 H: - history

Designed for multi-line notes, stories and anecdotes.

Although the **H:** fields are typically not typeset, the correct usage for multi-line input is to use field continuation syntax (**+:**), rather than **H:** at the start of each subsequent line of a multi-line note. This allows, for example, database applications to distinguish between two different anecdotes.

Examples:

```
H:this is considered
+:as a single entry

H:this usage is considered as two entries
H:rather than one
```

The original usage of **H:** (where subsequent lines need no field indicator) is now deprecated (see outdated information field syntax).

See typesetting information fields for details of how the history may be included in the printed score.

3.1.14 **K:** - key

The key signature should be specified with a capital letter (A–G) which may be followed by a **#** or **b** for sharp or flat respectively. In addition the mode should be specified (when no mode is indicated, **major** is assumed).

For example, **K:C major**, **K:A minor**, **K:C ionian**, **K:A aeolian**, **K:G mixolydian**, **K:D dorian**, **K:E phrygian**, **K:F lydian** 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**. As a special case, **minor** may be abbreviated to **m**.

This table sums up how the same key signatures can be written in different ways:

Mode

Ionian

Aeolian

Mixolydian

Dorian

Phrygian

Lydian

Locrian

Key Signature

Major

Minor

7 sharps

C#

A#m

G#Mix

D#Dor

E#Phr

F#Lyd
B#Loc
6 sharps
F#
D#m
C#Mix
G#Dor
A#Phr
BLyd
E#Loc
5 sharps
B
G#m
F#Mix
C#Dor
D#Phr
ELyd
A#Loc
4 sharps
E
C#m
BMix
F#Dor
G#Phr
ALyd
D#Loc
3 sharps
A
F#m
EMix
BDor

C#Phr
DLyd
G#Loc
2 sharps
D
Bm
AMix
EDor
F#Phr
GLyd
C#Loc
1 sharp
G
Em
DMix
ADor
BPhr
CLyd
F#Loc
0 sharps/flats
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

By specifying an empty `K:` field, or `K:none`, it is possible to use no key signature at all.

The key signatures may be *modified* by adding accidentals, according to the format `K:<tonic> <mode> <accidentals>`. For example, `K:D Phr ^f` would give a key signature with two flats and one sharp, which designates a very common mode in Klezmer (Ahavoh Rabbah) and in Arabic music (Maqam Hedjaz). Likewise, `"K:D maj =c"` or `"K:D =c"` will give a key signature with F sharp and c natural (the D mixolydian mode). Note that there can be several modifying accidentals, separated by spaces, each beginning with an accidental sign (`_`, `=`, `^` or `^^`), followed by a note letter. The case of the letter is used to determine on which line the accidental is placed.

It is possible to use the format `K:<tonic> exp <accidentals>` to explicitly define all the accidentals of a key signature. Thus `K:D Phr ^f` could also be

notated as `K:D exp _b _e ^f`, where ‘exp’ is an abbreviation of ‘explicit’. Again, the case of the letter is used to determine on which line the accidental is placed.

Software that does not support explicit key signatures should mark the individual notes in the tune with the accidentals that apply to them.

Scottish highland pipes typically have the scale `G A B ^c d e ^f g a` and highland pipe music primarily uses the modes D major and A mixolyian (plus B minor and E dorian). Therefore 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 sharp, C sharp and G natural. Both force all the beams and stems of normal notes to go downwards, and of grace notes to go upwards.

By default, the abc tune will be typeset with a treble clef. You can add special clef specifiers to the `K:` field, with or without a key signature, to change the clef and various other staff properties, such as transposition. `K: clef=bass`, for example, would indicate the bass clef. See clefs and transposition for full details.

Note that the first occurrence of the `K:` field, which must appear in every tune, finishes the tune header. All following lines are considered to be part of the tune body.

3.1.15 R: - rhythm

Contains an indication of the type of tune (e.g. hornpipe, double jig, single jig, 48-bar polka, etc). This gives the musician some indication of how a tune should be interpreted as well as being useful for database applications (see background information). It has also been used experimentally by playback software (in particular, abcmus) to provide more realistic playback by altering the stress on particular notes within a bar.

See typesetting information fields for details of how the rhythm may be included in the printed score.

3.1.16 B:, D:, F:, S: - background information

The information fields `B:book` (i.e. printed tune book), `D:discography` (i.e. a CD or LP where the tune can be heard), `F:file url` (i.e. where the either the abc tune or the abc file can be found on the web) and `S:source` (i.e. the circumstances under which a tune was collected or learned), as well as the fields `H:history`, `N:notes`, `O:origin` and `R:rhythm` mentioned above, are used for providing structured background information about a tune. These are particularly aimed at large tune collections (common in abc since its inception) and, if used in a systematic way, mean that abc database software can sort, search and filter on specific fields (for example, to sort by rhythm or filter out all the tunes on a particular CD).

The abc standard does not prescribe how these fields should be used, but it is typical to employ several fields of the same type each containing one piece of information, rather than one field containing several pieces of information (see English.abc for some examples).

See typesetting information fields for details of how background information may be included in the printed score.

3.1.17 I: - instruction

The I:(instruction) field is used for an extended set of instruction directives concerned with how the abc code is to be interpreted.

The I: field can be used interchangeably with stylesheet directives so that any I:directive may instead be written %%directive, and vice-versa. However, to use the inline version, the I: version must be used.

Despite this interchangeability, certain directives have been adopted as part of the standard (indicated by I: in this document) and must be implemented by software conforming to this version of the standard; conversely, the stylesheet directives (indicated by %% in this document) are optional.

Comment: Since stylesheet directives are optional, and not necessarily portable from one program to another, this means that I: fields containing stylesheet directives should be treated liberally by abc software and, in particular, that I: fields which are not recognised should be ignored.

The following table contains a list of the I: field directives adopted as part of the abc standard, with links to further information:

directive	section
I:abc-charset	charset field
I:abc-version	version field
I:abc-include	include field
I:abc-creator	creator field
I:linebreak	typesetting line breaks
I:decoration	decoration dialects

Typically, instruction fields are for use in the file header, to set defaults for the file, or (in most cases) in the tune header, but not in the tune body. The occurrence of an instruction field in a tune header overrides that in the file header.

Comment: Remember that abc software which extracts separate tunes from a file must insert the fields of the original file header into the header of the extracted tune: this is also true for the fields defined in this section.

Charset field

The `I:abc-charset <value>` field indicates the character set in which text strings are coded. Since this affects how the file is read, it should appear as early as possible in the file header. It may not be changed further on in the file.

Example:

```
I:abc-charset utf-8
```

Legal values for the charset field are `iso-8859-1` through to `iso-8859-10`, `us-ascii` and `utf-8` (the default).

Software that exports abc tunes conforming to this standard should include a charset field if an encoding other than `utf-8` is used. All conforming abc software must be able to handle text strings coded in `utf-8` and `us-ascii`. Support for the other charsets is optional.

Extensive information about UTF-8 and ISO-8859 can be found on wikipedia.

Version field

Every abc file conforming to this standard should start with the line

```
%abc-2.1
```

(see abc file identification).

However to indicate tunes conforming to a different standard it is possible to use the `I:abc-version <value>` field, either in the tune header (for individual tunes) or in the file header.

Example:

```
I:abc-version 2.0
```

Include field

The `I:abc-include <filename.abh>` imports the definitions found in a separate abc header file (`.abh`), and inserts them into the file header or tune header.

Example:

```
I:abc-include mydefs.abh
```

The included file may contain information fields, stylesheet directives and comments, but no other abc constructs.

If the header file cannot be found, the `I:abc-include` instruction should be ignored with a non-fatal error message.

Comment: If you use this construct and distribute your abc files, make sure that you distribute the `.abh` files with them.

Creator field

The `I:abc-creator <value>` field contains the name and version number of the program that created the abc file.

Example:

```
I:abc-creator xml2abc-2.7
```

Software that exports abc tunes conforming to this standard must include a creator field.

3.1.18 Other fields

- For `m:` see macros.
- For `r:` see comments and remarks.
- For `s:` see symbol lines.
- For `U:` see redefinable symbols.
- For `V:` see multiple voices.
- For `W:` and `w:` see lyrics.

3.2 Use of fields within the tune body

It is often desired to change the key (`K`), meter (`M`), or unit note length (`L`) mid-tune. These, and most other information fields which can be legally used within the tune body, can be specified as an **inline field** by placing them within square brackets in a line of music

Example: The following two excerpts are considered equivalent - either variant is equally acceptable.

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

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

The first bracket, field identifier and colon must be written without intervening spaces. Only one field may be placed within a pair of brackets; however, multiple bracketed fields may be placed next to each other. Where appropriate, inline fields (especially clef changes) can be used in the middle of a beam without breaking it.

See information fields for a table showing the fields that may appear within the body and those that may be used inline.

3.3 Field continuation

A field that is too long for one line may be continued by prefixing `+` at the start of the following line. For string-type information fields (see the information fields table for a list of string-type fields), the continuation is considered to add a space between the two half lines.

Example: The following two excerpts are considered equivalent.

```
w:Sa-ys my au-l' wan to your aul' wan,  
+:will-ye come to the Wa-x-ies dar-gle?
```

```
w:Sa-ys my au-l' wan to your aul' wan, will-ye come to the Wa-x-ies dar-gle?
```

Comment: This is most useful for continuing long `w:(aligned lyrics)` and `H:(history)` fields. However, it can also be useful for preventing automatic wrapping by email software (see continuation of input lines).

Recommendation for GUI developers: Sometimes users may wish to paste paragraphs of text into an abc file, particularly in the `H:(history)` field. GUI developers are recommended to provide tools for reformatting such paragraphs, for example by splitting them into several lines each prefixed by `+`.

There is no limit to the number of times a field may be continued and comments and stylesheet directives may be interspersed between the continuations.

Example: The following is a legal continuation of the `w:` field, although the usage not recommended (the change of font could also be achieved by font specifiers - see font directives).

```
%%vocalfont Times-Roman 14  
w:nor-mal  
% legal, but not recommended  
%%vocalfont Times-Italic *  
+:i-ta-lic  
%%vocalfont Times-Roman *  
+:nor-mal
```

Comment: abc standard 2.3 is scheduled to address markup and will be seeking a more elegant way to achieve the above.

4. The tune body

4.1 Pitch

The following letters are used to represent notes using the treble clef:



and by extension other lower and higher notes are available.

Lower octaves are reached by using commas and higher octaves are written using apostrophes; each extra comma/apostrophe lowers/raises the note by an octave.

Programs should be able to parse any combinations of , and ' signs appearing after the note. For example C,', (C comma apostrophe comma) has the same meaning as C, (C comma) and (uppercase) C' (C apostrophe) should have the same meaning as (lowercase) c.

Alternatively, it is possible to raise or lower a section of music code using the `octave` parameter of the `K:` or `V:` fields.

Comment: The English note names C-B, which are used in the abc system, correspond to the note names do-si, which are used in many other languages: do=C, re=D, mi=E, fa=F, sol=G, la=A, si=B.

4.2 Accidentals

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

4.3 Note lengths

Throughout this document note lengths are referred as sixteenth, eighth, etc. The equivalents common in the U.K. are sixteenth note = semi-quaver, eighth = quaver, quarter = crotchet and half = minim.

The unit note length for the transcription is set in the `L:` field or, if the `L:` field does not exist, inferred from the `M:` field. For example, `L:1/8` sets an eighth note as the unit note length.

A single letter in the range A-G, a-g then represents a note of this length. For example, if the unit note length is an eighth note, DEF represents 3 eighth notes.

Notes of differing lengths can be obtained by simply putting a multiplier after the letter. Thus if the unit note length is 1/16, 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. If the unit note length is 1/8, A is an eighth note, A2 a quarter note, A3 a dotted quarter note, A4 a half note, and so on.

To get shorter notes, either divide them - e.g. if A is an eighth note, A/2 is a sixteenth note, A3/2 is a dotted eighth note, A/4 is a thirty-second note - or change the unit 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.

Note that A/ is shorthand for A/2 and similarly A// = A/4, etc.

Comment: Note lengths that can't be translated to conventional staff notation are legal, but their representation by abc typesetting software is undefined and they should be avoided.

Note for developers: All compliant software should be able to handle note lengths down to a 128th note; shorter lengths are optional.

4.4 Broken rhythm

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'.

Example: 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 <<<.

Note that the use of broken rhythm markers between notes of unequal lengths will produce undefined results, and should be avoided.

4.5 Rests

Rests can be transcribed with a `z` or an `x` and can be modified in length in exactly the same way as normal notes. `z` rests are printed in the resulting sheet music, while `x` rests are invisible, that is, not shown in the printed music.

Multi-measure rests are notated using `Z` (upper case) followed by the number of measures.

Example: The following excerpts, shown with the typeset results, are musically equivalent (although they are typeset differently).

`Z4|CD EF|GA Bc`



`z4|z4|z4|z4|CD EF|GA Bc`



When the number of measures is not given, `Z` is equivalent to a pause of one measure.

By extension multi-measure invisible rests are notated using `X` (upper case) followed by the number of measures and when the number of measures is not given, `X` is equivalent to a pause of one measure.

Comment: Although not particularly valuable, a multi-measure invisible rest could be useful when a voice is silent for several measures.

4.6 Clefs and transposition

VOLATILE: This section is subject to some clarifications with regard to transposition, rules for the `middle` parameter and interactions between different parameters.

Clef and transposition information may be provided in the `K:` key and `V:` voice fields. The general syntax is:

`[clef=]<clef name>[<line number>][+8 | -8] [middle=<pitch>] [transpose=<semitones>] [octave=<n>]`

(where `<...>` denotes a value, `[...]` denotes an optional parameter, and `|` separates alternative values).

- `<clef name>` - may be `treble`, `alto`, `tenor`, `bass`, `perc` or `none`. `perc` selects the drum clef. `clef=` may be omitted.
- `[<line number>]` - indicates on which staff line the base clef is written. Defaults are: `treble: 2`; `alto: 3`; `tenor: 4`; `bass: 4`.
- `[+8 | -8]` - draws '8' above or below the staff. The player will transpose the notes one octave higher or lower.
- `[middle=<pitch>]` - is an alternate way to define the line number of the clef. The pitch indicates what note is displayed on the 3rd line of the staff. Defaults are: `treble: B`; `alto: C`; `tenor: A,`; `bass: D,`; `none: B`. This setting does not affect the playback.
- `[transpose=<semitones>]` - for playback, transpose the current voice by the indicated amount of semitones; positive numbers transpose up, negative down. This setting does not affect the printed score. The default is 0.
- `[octave=<number>]` to raise (positive number) or lower (negative number) the music code in the current voice by one or more octaves. This usage can help to avoid the need to write lots of apostrophes or commas to raise or lower notes.
- `[stafflines=<lines>]` - the number of lines in the staff. The default is 5.

Note that the `clef`, `middle`, `transpose`, `octave` and `stafflines` specifiers may be used independent of each other.

Examples:

```
K:   clef=alto
K:   perc stafflines=1
K:Am transpose=-2
V:B  middle=d bass
```

Note that although this standard supports the drum clef, there is currently no support for special percussion notes.

The middle specifier can be handy when working in the bass clef. Setting `K:bass middle=d transpose=-24` will save you from adding comma specifiers to the notes (the `transpose` setting is required to get the playback sounding at the correct pitch). The specifier may be abbreviated to `m=`.

The transpose specifier is useful, for example, for a Bb clarinet, for which the music is written in the key of C although the instrument plays it in the key of Bb:

```
V:Clarinet
K:C transpose=-2
```

The transpose specifier may be abbreviated to `t=`.

To notate the various standard clefs, one can use the following specifiers:

The seven clefs

Name	specifier
Treble	K:treble
Bass	K:bass
Baritone	K:bass3
Tenor	K:tenor
Alto	K:alto
Mezzosoprano	K:alto2
Soprano	K:alto1

More clef names may be allowed in the future, therefore unknown names should be ignored. If the clef is unknown or not specified, the default is treble.

Applications may introduce their own clef line specifiers. These specifiers should start with the name of the application, followed a colon, followed by the name of the specifier.

Example:

```
V:p1 perc stafflines=3 m=C  mozart:noteC=snare-drum
```

4.7 Beams

To group notes together under one beam they must 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 stems are typeset automatically.

Notes that cannot be beamed may be placed next to each other. For example, if L:1/8 then `ABC2DE` is equivalent to `AB C2 DE`.

Back quotes ``` may be used freely between notes to be beamed, to increase legibility. They are ignored by computer programs. For example, `A2``B``C` is equivalent to `A2BC`.

4.8 Repeat/bar symbols

Bar line symbols are notated as follows:

Symbol	Meaning
	bar line
]	thin-thick double bar line
	thin-thin double bar line
[thick-thin double bar line
:	start of repeated section
:	end of repeated section
::	start & end of two repeated sections

Recommendation for developers: If an ‘end of repeated section’ is found without a previous ‘start of repeated section’, playback programs should restart the music from the beginning of the tune, or from the latest double bar line or end of repeated section.

Note that the notation `::` is short for `:|` followed by `|:`. The variants `::`, `:|:` and `:||:` are all equivalent.

By extension, `|::` and `::|` mean the start and end of a section that is to be repeated three times, and so on.

A dotted bar line can be notated by preceding it with a dot, e.g. `.|` - this may be useful for notating editorial bar lines in music with very long measures.

An invisible bar line may be notated by putting the bar line in brackets, e.g. `[|]` - this may be useful for notating voice overlay in meter-free music.

Abc parsers should be quite liberal in recognizing bar lines. In the wild, bar lines may have any shape, using a sequence of `|` (thin bar line), `[` or `]` (thick bar line), and `:` (dots), e.g. `|[|` or `[|:::`.

4.9 First and second repeats

First and second repeats can be notated 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, while

```
| 1
```

is not.

Thus, a tune with different ending for the first and second repeats has the general form:

|: <common body of tune> |1 <first ending> :|2 <second ending> |]

Note that in many abc files the |: may not be present.

4.10 Variant endings

In combination with P: part notation, it is possible to notate more than two variant endings for a section that is to be repeated a number of times.

For example, if the header of the tune contains P:A4.B4 then parts A and B will each be played 4 times. To play a different ending each time, you could write in the tune:

P:A
<notes> | [1 <notes> :| [2 <notes> :| [3 <notes> :| [4 <notes> |]

The Nth ending starts with [N and ends with one of ||, :|] or [|. You can also mark a section as being used for more than one ending e.g.

[1,3 <notes> :|

plays on the 1st and 3rd endings and

[1-3 <notes> :|

plays on endings 1, 2 and 3. In general, '[' can be followed by any list of numbers and ranges as long as it contains no spaces e.g.

[1,3,5-7 <notes> :| [2,4,8 <notes> :|

4.11 Ties and slurs

You can tie two notes of the same pitch together, within or between bars, with a - symbol, e.g. abc|cba or c4-c4. The tie symbol must always be adjacent to the first note of the pair, but does not need to be adjacent to the second, e.g. c4 -c4 and abc| -cba are not legal - see order of abc constructs.

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) .

Slurs may be nested:

(c (d e f) g a)



and they may also start and end on the same note:

(c d (e) f g a)



A dotted slur may be notated by preceding the opening brace with a dot, e.g. `.(cde)`; it is optional to place a dot immediately before the closing brace. Likewise, a dotted tie can be transcribed by preceding it with a dot, e.g. `C.-C`. This is especially useful in parts with multiple verses: some verses may require a slur, some may not.

It should be noted that although the tie - and slur () produce similar symbols in staff notation they have completely different meanings to player programs and should not be interchanged. Ties connect two successive notes *of the same pitch*, causing them to be played as a single note, while slurs connect the first and last note of any series of notes, and may be used to indicate phrasing, or that the group should be played legato. Both ties and slurs may be used into, out of and between chords, and in this case the distinction between them is particularly important.

4.12 Grace notes

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. Although nominally grace notes have no melodic time value, expressions such as `{a3/2b/}` or `{a>b}` can be useful and are legal although some software may ignore them. The unit duration to use for gracenotes is not specified by the abc file, but by the software, and might be a specific amount of time (for playback purposes) or a note length (e.g. `1/32` for Highland pipe music, which would allow `{ge4d}` to code a piobaireachd ‘cadence’).

To distinguish between appoggiaturas and acciaccaturas, the latter are notated with a forward slash immediately following the open brace, e.g. `{/g}C` or `{/gagab}C`:

4.14 Decorations

A number of shorthand decoration symbols are available:

.	staccato mark
~	Irish roll
H	fermata
L	accent or emphasis
M	lowermordent
O	coda
P	uppermordent
S	segno
T	trill
u	up-bow
v	down-bow

Decorations should be placed before the note which they decorate - see order of abc constructs

Examples:

```
(3.a.b.c    % staccato triplet
vAuBvA     % bowing marks (for fiddlers)
```

Most of the characters above (~HLMOPSTuv) are just short-cuts for commonly used decorations and can even be redefined (see redefinable symbols).

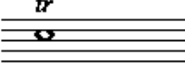
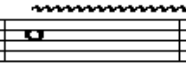
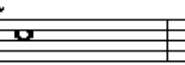
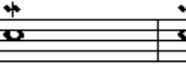
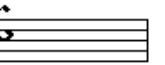
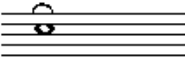
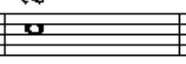
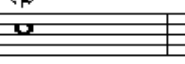
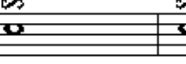
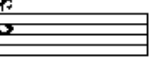
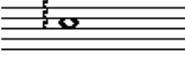
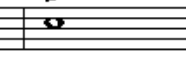
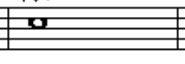
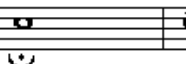
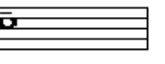
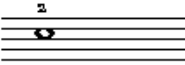
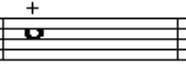
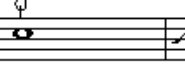
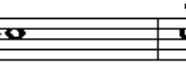
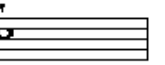
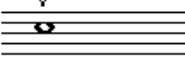
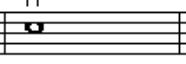
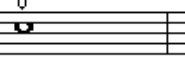
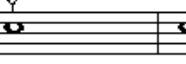
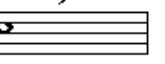
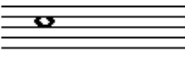
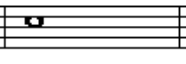
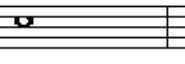
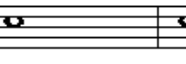
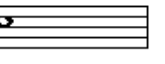
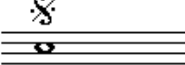
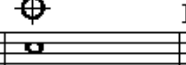
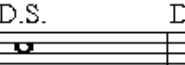
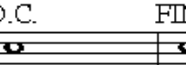
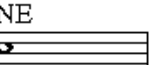
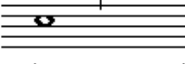
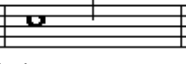

More generally, symbols can be entered using the syntax **!symbol!**, e.g. **!trill!A4** for a trill symbol. (Note that the abc standard version 2.0 used instead the syntax **+symbol+** - this dialect of abc is still available, but is now deprecated - see decoration dialects.)

The currently defined symbols are:

!trill!	"tr" (trill mark)
!trill(!	start of an extended trill
!trill)!	end of an extended trill
!lowermordent!	short / / / squiggle with a vertical line through it
!uppermordent!	short / / / squiggle
!mordent!	same as !lowermordent!
!pralltriller!	same as !uppermordent!
!roll!	a roll mark (arc) as used in Irish music
!turn!	a turn mark (also known as gruppetto)
!turnx!	a turn mark with a line through it
!invertedturn!	an inverted turn mark
!invertedturnx!	an inverted turn mark with a line through it
!arpeggio!	vertical squiggle
!>!	> mark
!accent!	same as !>!

!emphasis!	same as !>!
!fermata!	fermata or hold (arc above dot)
!invertedfermata!	upside down fermata
!tenuto!	horizontal line to indicate holding note for full duration
!0! - !5!	fingerings
!+!	left-hand pizzicato, or rasp for French horns
!plus!	same as !+!
!snap!	snap-pizzicato mark, visually similar to !thumb!
!slide!	slide up to a note, visually similar to a half slur
!wedge!	small filled-in wedge mark
!upbow!	V mark
!downbow!	squared n mark
!open!	small circle above note indicating open string or harmonic
!thumb!	cello thumb symbol
!breath!	a breath mark (apostrophe-like) after note
!pppp! !ppp! !pp! !p!	dynamics marks
!mp! !mf! !f! !ff!	more dynamics marks
!fff! !ffff! !sfz!	more dynamics marks
!crescendo(!	start of a < crescendo mark
!<(!	same as !crescendo(!
!crescendo)!	end of a < crescendo mark, placed after the last note
!<)!	same as !crescendo)!
!diminuendo(!	start of a > diminuendo mark
!>(!	same as !diminuendo(!
!diminuendo)!	end of a > diminuendo mark, placed after the last note
!>)!	same as !diminuendo)!
!segno!	2 ornate s-like symbols separated by a diagonal line
!coda!	a ring with a cross in it
!D.S.!	the letters D.S. (=Da Segno)
!D.C.!	the letters D.C. (=either Da Coda or Da Capo)
!dacoda!	the word "Da" followed by a Coda sign
!dacapo!	the words "Da Capo"
!fine!	the word "fine"
!shortphrase!	vertical line on the upper part of the staff
!mediumphrase!	same, but extending down to the centre line
!longphrase!	same, but extending 3/4 of the way down

Here is a picture of most decorations:

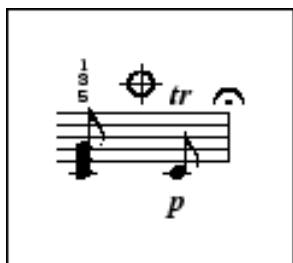
				
<code>!trill!</code>	<code>!trill(!</code>	<code>!trill)!</code>	<code>!lowermordent!</code>	<code>!uppermordent!</code>
				
<code>!roll!</code>	<code>!turn!</code>	<code>!turnx!</code>	<code>!invertedturn!</code>	<code>!invertedturnx</code>
				
<code>!arpeggio!</code>	<code>!>!</code>	<code>!fermata!</code>	<code>!invertedfermata!</code>	<code>!tenuto!</code>
				
<code>!2!</code>	<code>!+!</code>	<code>!snap!</code>	<code>!slide!</code>	<code>!wedge!</code>
				
<code>!upbow!</code>	<code>!downbow!</code>	<code>!open!</code>	<code>!thumb!</code>	<code>!breath!</code>
				
<code>!pppp!</code>	<code>!<(!</code>	<code>!<)!</code>	<code>!>(!</code>	<code>!>)!</code>
				
<code>!segno!</code>	<code>!coda!</code>	<code>!D.S.!</code>	<code>!D.C.!</code>	<code>!fine!</code>
				
<code>!shortphrase!</code>	<code>!mediumphrase!</code>	<code>!longphrase!</code>		

Note that the decorations may be applied to notes, rests, note groups, and bar lines. If a decoration is to be typeset between notes, it may be attached to the y spacer - see typesetting extra space.

Spaces may be used freely between each of the symbols and the object to which it should be attached. Also an object may be preceded by multiple symbols, which should be printed one over another, each on a different line.

Example:

```
[!1!C!3!E!5!G] !coda! y !p! !trill! C !fermata!|
```



Player programs may choose to ignore most of the symbols mentioned above, though they may be expected to implement the dynamics marks, the accent mark and the staccato dot. Default volume is equivalent to `!mf!`. On a scale from 0-127, the relative volumes can be roughly defined as: `!pppp!` = `!ppp!` = 30, `!pp!` = 45, `!p!` = 60, `!mp!` = 75, `!mf!` = 90, `!f!` = 105, `!ff!` = 120, `!fff!` = `!ffff!` = 127.

Abc software may also allow users to define new symbols in a package dependent way.

Note that symbol names may not contain any spaces, `[`, `]`, `|` or `:` signs, e.g. while `!dacapo!` is legal, `!da capo!` is not.

If an unimplemented or unknown symbol is found, it should be ignored.

Recommendation: A good source of general information about decorations can be found at <http://www.dolmetsch.com/musicalsymbols.htm>.

4.15 Symbol lines

Adding many symbols to a line of music can make a tune difficult to read. In such cases, a symbol line (a line that contains only `!...!` decorations, `"..."` chord symbols or annotations) can be used, analogous to a line of lyrics.

A symbol line starts with `s:`, followed by a line of symbols. Matching of notes and symbols follow the alignment rules defined for lyrics (meaning that symbols in an `s:` line cannot be aligned on grace notes, rests or spacers).

Example:

```
CDEF      | G````AB`c
s: "~slow" | !f! ** !fff!
```

It is also possible to stack `s:` lines to produced multiple symbols on a note.

Example: The following two excerpts are equivalent and would place a decorations plus a chord on the E.

```
C2  C2 Ez  A2|
s: "C" *  "Am" * |
s: *    *  !>! * |
```

```
"C" C2 C2 "Am" !>! Ez A2|
```

4.16 Redefinable symbols

As a short cut to writing symbols which avoids the `!symbol!` syntax (see decorations), the letters H-W and h-w and the symbol ~ can be assigned with the `U:` field. For example, to assign the letter T to represent the trill, you can write:

```
U: T = !trill!
```

You can also use `^text`, etc (see annotations below) in definitions

Example: To print a plus sign over notes, define `p` as follows and use it before the required notes:

```
U: p = "^+"
```

Symbol definitions can be written in the file header, in which case they apply to all the tunes in that file, or in a tune header, when they apply only to that tune, and override any previous definitions. Programs may also make use of a set of global default definitions, which apply everywhere unless overridden by local definitions. You can assign the same symbol to two or more letters e.g.

```
U: T = !trill!
```

```
U: U = !trill!
```

in which case the same visible symbol will be produced by both letters (but they may be played differently), and you can de-assign a symbol by writing:

```
U: T = !nil!
```

or

```
U: T = !none!
```

The standard set of definitions (if you do not redefine them) is:

```
U: ~ = !roll!
```

```
U: H = !fermata!
```

```
U: L = !accent!
```

```
U: M = !lowermordent!
```

```
U: O = !coda!
```

```
U: P = !uppermordent!
```

```
U: S = !segno!
```

```
U: T = !trill!
```

```
U: u = !upbow!
```

```
U: v = !downbow!
```

Please see macros for an advanced macro mechanism.

4.17 Chords and unisons

Chords (i.e. more than one note head on a single stem) can be coded with [] symbols around the notes, e.g.

[CEGc]

indicates the chord of C major. They can be grouped in beams, e.g.

[d2f2] [ce] [df]

but there should be no spaces within the notation for a chord. See the tune 'Kitchen Girl' in the sample file Reels.abc for a simple example.

All the notes within a chord should normally have the same length, but if not, the chord duration is that of the first note.

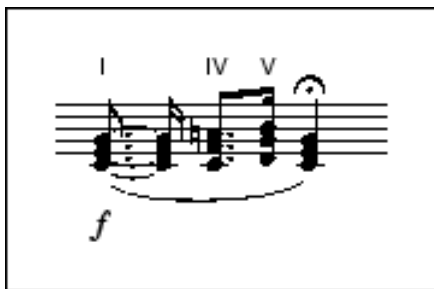
Recommendation: Although playback programs should not have any difficulty with notes of different lengths, typesetting programs may not always be able to render the resulting chord to staff notation (for example, an eighth and a quarter note cannot be represented on the same stem) and the result is undefined. Consequently, this is not recommended.

More complicated chords can be transcribed with the & operator (see voice overlay).

The chord forms a syntactic grouping, to which the same prefixes and postfixes can be attached as to an ordinary note (except for accidentals which should be attached to individual notes within the chord and decorations which may be attached to individual notes within the chord or may be attached to the chord as a whole).

Example:

```
( "^I" !f! [CEG]- > [CEG] "^IV" [F=AC] 3/2 "^V" [GBD]/ H[CEG]2 )
```



When both inside and outside the chord length modifiers are used, they should be multiplied. *Example:* [C2E2G2]3 has the same meaning as [CEG]6.

If the chord contains two notes of the same pitch, then it is a unison (e.g. a note played on two strings of a violin simultaneously) and is shown with one stem and two note-heads.

Example:

[DD]



4.18 Chord symbols

VOLATILE: The list of chords and how they are handled will be extended at some point. Until then programs should treat chord symbols quite liberally.

Chord symbols (e.g. chords/bass notes) can be put in under the melody line (or above, depending on the package) using double-quotation marks placed to the left of the note it is sounded with, e.g. "Am7"A2D2.

The chord has the format `<note><accidental><type></bass>`, where `<note>` can be A-G, the optional `<accidental>` can be b, #, the optional `<type>` is one or more of

m or min	minor
maj	major
dim	diminished
aug or +	augmented
sus	suspended
7, 9 ...	7th, 9th, etc.

and `</bass>` is an optional bass note.

A slash after the chord type is used only if the optional bass note is also used, e.g., "C/E". If the bass note is a regular part of the chord, it indicates the inversion, i.e., which note of the chord is lowest in pitch. If the bass note is not a regular part of the chord, it indicates an additional note that should be sounded with the chord, below it in pitch. The bass note can be any letter (A-G or a-g), with or without a trailing accidental sign (b or #). The case of the letter used for the bass note does not affect the pitch.

Alternate chords can be indicated for printing purposes (but not for playback) by enclosing them in parentheses inside the double-quotation marks after the regular chord, e.g., "G(Em)".

Note to developers: Software should also be able to recognise and handle appropriately the unicode versions of flat, natural and sharp symbols (¯, ¸, ¸) - see special symbols.

4.19 Annotations

General text annotations can be added above, below or on the staff in a similar way to chord symbols. In this case, the string within double quotes is preceded by one of five symbols ^, _, <, > or @ which controls where the annotation is to be placed; above, below, to the left or right respectively of the following note, rest or bar line. Using the @ symbol leaves the exact placing of the string to the discretion of the interpreting program. These placement specifiers distinguish annotations from chord symbols, and should prevent programs from attempting to play or transpose them. All text that follows the placement specifier is treated as a text string.

Where two or more annotations with the same placement specifier are placed consecutively, e.g. for fingerings, the notation program should draw them on separate lines, with the first listed at the top.

Example: The following annotations place the note between parentheses.

```
"<(" ">)" C
```

4.20 Order of abc constructs

The order of abc constructs for a note is: *<grace notes>*, *<chord symbols>*, *<annotations>/<decorations>* (e.g. Irish roll, staccato marker or up/downbow), *<accidentals>*, *<note>*, *<octave>*, *<note length>*, i.e. `~^c'3` or even `"Gm7"v.=G,2.`

Each tie symbol, -, should come immediately after a note group but may be followed by a space, i.e. `=G,2-`. Open and close chord delimiters, [and], should enclose entire note sequences (except for chord symbols), e.g.

```
"C"[CEGc]|
|"Gm7" [. =G, ^c ']
```

and open and close slur symbols, (), should do likewise, i.e.

```
"Gm7"(v.=G,2~^c'2)
```

5. Lyrics

The **W:** information field (uppercase W) can be used for lyrics to be printed separately below the tune.

The **w:** information field (lowercase w) in the tune body, supplies lyrics to be aligned syllable by syllable with previous notes of the current voice.

5.1 Alignment

When adjacent, **w:** fields indicate different verses (see below), but for non-adjacent **w:** fields, the alignment of the lyrics:

- starts at the first note of the voice if there is no previous **w:** field; or
- starts at the first note after the notes aligned to the previous **w:** field; and
- associates syllables to notes up to the end of the **w:** line.

Example: The following two examples are equivalent.

```
C D E F|
w: doh re mi fa
G A B c|
w: sol la ti doh

C D E F|
G A B c|
w: doh re mi fa sol la ti doh
```

Comment: The second example, made possible by an extension (introduced in abc 2.1) of the alignment rules, means that lyrics no longer have to follow immediately after the line of notes to which they are attached. Indeed, the placement of the lyrics can be postponed to the end of the tune body. However, the extension of the alignment rules is not fully backwards compatible with abc 2.0 - see outdated lyrics alignment for an explanation.

If there are fewer syllables than available notes, the remaining notes have no lyric (blank syllables); thus the appearance of a **w:** field associates all the notes that have appeared previously with a syllable (either real or blank).

Example: In the following example the empty **w:** field means that the 4 **G** notes have no lyric associated with them.

```
C D E F|
w: doh re mi fa
G G G G|
w:
F E F C|
w: fa mi re doh
```

If there are more syllables than available notes, any excess syllables will be ignored.

Recommendation for developers: If a **w:** line does not contain the correct number of syllables for the corresponding notes, the program should warn the user. However, having insufficient syllables is legitimate usage (as above) and so the program may allow these warnings to be switched off.

Note that syllables are not aligned on grace notes, rests or spacers and that tied, slurred or beamed notes are treated as separate notes in this context.

The lyrics lines are treated as text strings. Within the lyrics, the words should be separated by one or more spaces and to correctly align them the following symbols may be used:

Symbol	Meaning
-	(hyphen) break between syllables within a word
_	(underscore) previous syllable is to be held for an extra note
*	one note is skipped (i.e. * is equivalent to a blank syllable)
~	appears as a space; aligns multiple words under one note
\-	appears as hyphen; aligns multiple syllables under one note
	advances to the next bar

Note that if - is preceded by a space or another hyphen, the - is regarded as a separate syllable.

When an underscore is used next to a hyphen, the hyphen must always come first.

If there are not as many syllables as notes in a measure, typing a | automatically advances to the next bar; if there are enough syllables the | is just ignored.

Examples:

```
w: syll-a-ble      is aligned with three notes
w: syll-a--ble     is aligned with four notes
w: syll-a -ble     (equivalent to the previous line)
w: time__         is aligned with three notes
w: of~the~day     is treated as one syllable (i.e. aligned with one note)
                  but appears as three separate words
```

```
gf|e2dc B2A2|B2G2 E2D2|.G2.G2 GABc|d4 B2
w: Sa-ys my au-l' wan to your aul' wan,
+: Will~ye come to the Wa-x-ies dar-gle?
```

See field continuation for the meaning of the +: field continuation.

5.2 Verses

It is possible for a music line to be followed by several adjacent **w:** fields, i.e. immediately after each other. This can be used, together with part notation, to represent different verses. The first **w:** field is used the first time that part is played, then the second and so on.

Examples: The following two examples are equivalent and contain two verses:

```

CDEF FEDC|
w: these are the lyr-ics for verse one
w: these are the lyr-ics for verse two

CDEF FEDC|
w: these are the lyr-ics
+: for verse one
w: these are the lyr-ics
+: for verse two

```

5.3 Numbering

VOLATILE: The following syntax may be extended to include non-numeric “numbering”.

If the first word of a **w:** line starts with a digit, this is interpreted as numbering of a stanza. Typesetting programs should align the corresponding note with the first letter that occurs. This can be used in conjunction with the `~` symbol mentioned in the table above to create a space between the digit and the first letter.

Example: In the following, the `1.~Three` is treated as a single word with a space created by the `~`, but the fact that the **w:** line starts with a number means that the first note of the corresponding music line is aligned to **Three**.

```
w: 1.~Three blind mice
```

6. Typesetting and playback

6.1 Typesetting

6.1.1 Typesetting line-breaks

Terminology: **Line-breaks** in a document (also known in computing as new lines, line-feeds, carriage-returns, end-of-lines, etc.) determine how the document is set out on the page. Throughout this section, and elsewhere in the standard, a distinction should be noted between

- a **code line-break**, meaning a line-break in the abc tune body, and, in particular, at the end of a line of music code;
- a **score line-break**, meaning a line-break in the printed score.

The fundamental mechanism for typesetting score line-breaks is by using code line-breaks - one line of music code in the tune body normally corresponds to one line of printed music.

Of course the printed representation of a line of music code may be too long for the staff, so if necessary, typesetting programs should introduce additional score line-breaks. As a consequence, if you would prefer score line-breaks to be handled completely automatically (as is common in non-abc scoring software), then just type the tune body on a single line of music code.

Even though most abc GUI software should wrap over-long lines, typing the tune body on a single line may not always be convenient, particularly for users who wish to include code line-breaks to aid readability or if the abc code is to be emailed (see continuation of input lines).

Furthermore, in the past some typesetting programs used `!` characters in the abc code to force score line-breaks.

As a result, abc 2.1 introduces a new line-breaking instruction.

I:linebreak

To allow for all line-breaking preferences, the **I:linebreak** instruction may be used, together with four possible values, to control score line-breaking.

- **"I:linebreak \$"** indicates that the `$` symbol is used in the tune body to typeset a score line-break. Any code line-breaks are ignored for typesetting purposes.

Example: The following abc code should be typeset on two lines.

```
I:linebreak $
K:G
|:abc def|$fed cba:|
```

- **"I:linebreak !"** indicates that the `!` symbol is used to typeset a score line-break. Any code line-breaks are ignored for typesetting purposes.

Comment: The **"I:linebreak !"** instruction works in the same way as **I:linebreak \$** and is primarily provided for backwards compatibility - see line-breaking dialects, so that **"I:linebreak \$"** is the preferred usage. **"I:linebreak !"** also automatically invokes the **"I:decoration +"** instruction - see decoration dialects. Finally, **"I:linebreak !"** is equivalent to the deprecated directive `%%continueall true` - see outdated directives.

- **"I:linebreak <EOL>"** indicates that the End Of Line character (CR, LF or CRLF) is used to typeset a score line-break. In other words, code line-breaks are used for typesetting score line-breaks.
- **"I:linebreak <none>"** indicates that all line-breaking is to be carried out automatically and any code line-breaks are ignored for typesetting purposes.

The values `<EOL>`, `$` and `!` may also be combined so that more than one symbol can indicate a score line-break.

The default line-break setting is:

```
I:linebreak <EOL> $
```

meaning that both code line-breaks, and \$ symbols, generate a score line-break.

Comment: Although "I:linebreak \$!" is legal it is not recommended as it uses two different symbols to mean the same thing.

An I:linebreak instruction can be used either in the file header (in which case it is applied to every tune in the abc file), or in a tune header (in which case it is applied to that tune only and overrides a line-breaking instruction in the file header). Similarly, if two I:linebreak instructions appear in a file header or a tune header, the second cancels the first.

Comment: It can be sometimes be useful to include two instructions together - for example, "I:linebreak <EOL> \$" and "I:linebreak <none>" can be used to toggle between default and automatic line-breaking simply by swapping the position of the two lines.

I:linebreak instructions are not allowed in the tune body (principally because it conflicts with the human readability of the music code).

Suppressing score line-breaks

When the <EOL> character is being used in the tune body to indicate score line-breaks, it sometimes useful to be able to tell typesetting software to ignore a particular code line-breaks. This is achieved using a backslash (\) at the end of a line of music code. The backslash may be followed by trailing whitespace and/or comments, since they are removed before the line is processed.

Example: The following two excerpts are considered equivalent and should be typeset as a single staff in the printed score.

```
abc cba|\ % end of line comment
abc cba|
```

```
abc cba|abc cba|
```

The backslash effectively joins two lines together for processing so if space is required between the two half lines (for example, to prevent the notes from being beamed together), it can be placed before the backslash, or at the beginning of the next half line.

Example: The following three excerpts are considered equivalent.

```
abc \
cba|
```

```
abc\
cba|
```

```
abc cba|
```

There is no limit to the number of lines that may be joined together in this way. However, a backslash must not be used before an empty line.

Example: The following is legal.

```
cdef|\  
\  
cedf:|
```

Example: The following is not legal.

```
cdef|\  
  
cedf:|
```

In the examples above, where a line of music code follows immediately after a line ending in backslash, the backslash acts as a continuation for two lines of music code and can therefore be used to split up long music code lines.

More importantly, however, any information fields and stylesheet directives are processed (and comments are removed) at the point where the physical line-break occurs. Hence the backslash is commonly used to include meter or key changes halfway through a line of music.

Example: The following should be typeset as a single staff in the printed score.

```
abc cab|\  
%%setbarnb 10  
M:9/8  
%comment  
abc cba abc|
```

Alternative usage example: The above could also be achieved using inline fields, the `I:<directive>` form instead of `%%<directive>` and a `r:remark` in place of the comment, i.e.

```
abc cab|[I:setbarnb 10][M:9/8][r:comment]abc cba abc|
```

Finally, note that if the `<EOL>` character is not being used to indicate score line-breaks, then the backslash is effectively redundant.

Recommendation to users: If you find that you are using backslash symbols on most lines of music code, then consider instead using `"I:linebreak <none>"` or `"I:linebreak $"` which will mean that all the code line-breaks will be ignored for the purposes of generating score line-breaks (and, in the latter case, you can encode a score line-breaks with the `$` character).

6.1.2 Typesetting extra space

y can be used to add extra space between the surrounding notes; moreover, chord symbols and decorations can be attached to it, to separate them from notes.

Example:

"Am" !pp! y

Note that the y symbol does *not* create rests in the music.

6.1.3 Typesetting information fields

By default typesetting programs should include the the title (T), composer (C), origin (O), parts (P), tempo (Q), aligned words (w) and other words (W) in the printed score, using the follow scheme:

- the **T:title** should be printed centred above the tune; alternative titles should be printed underneath the main title in smaller print
- the **C:composer** should be printed right-aligned, just below the title, each composer on a separate line
- the contents of the **O:origin** field should be appended to the **C:composer** field, surrounded by parentheses
- each **P:part** in the tune body should have the string identifying it printed immediately above the start of that part; if there is a **P:parts** field in the tune header (describing which order the parts are played in) it should be printed left-aligned above the start of the tune
- the **Q:tempo** should be printed above the tune at the start of the section to which it applies
- the aligned **w:words** (lyrics) should be printed under each line of music with other **W:words** printed beneath the tune - see lyrics

To suppress any of these, or alternatively to typeset additional information fields such as notes (N), history (H), rhythm (R), book (B), discography (D), file (F), source (S) or transcription (Z), use the **%%writefields** directive - see information directives.

To customise the typesetting (for example, by changing the font), see formatting directives.

6.2 Playback

Many of the information fields are ignored by playback programs - exceptions are **I:**, **K:**, **L:**, **M:**, **m:**, **P;**, **Q:**, **s:**, **U:** and **V:**.

In addition, playback programs that store their output in file types which have provisions for metadata (e.g. MIDI, ogg, mp3), may record the contents the **T:**, **C:**, **w:** and **W:** fields in that metadata.

Furthermore, playback programs may use the **R:** field to infer stress patterns in a tune (i.e. to make playback closer to real music, by for example, placing more stress on the first note in each bar); however, such usage is not standardised.

Most playback customisation is handled by instrumentation directives.

7. Multiple voices

VOLATILE: Multi-voice music is under active review, with discussion about control voices and interaction between **P:**, **V:** and **T:** fields. It is intended that the syntax will be finalised in abc 2.2.

The **V:** field allows the writing of multi-voice music. In multi-voice abc tunes, the tune body is divided into several voices, each beginning with a **V:** field. All the notes following such a **V:** field, up to the next **V:** field or the end of the tune body, belong to the voice.

The basic syntax of the field is:

V:ID

where ID can be either a number or a string, that uniquely identifies the voice in question. When using a string, only the first 20 characters of it will be distinguished. The ID will not be printed on the staff; it's only function is to indicate, throughout the abc tune, which music line belongs to which voice.

Example:

```
X:1
T:Zocharti Loch
C:Louis Lewandowski (1821-1894)
M:C
Q:1/4=76
%%score (T1 T2) (B1 B2)
V:T1          clef=treble-8  name="Tenore I"   snm="T.I"
V:T2          clef=treble-8  name="Tenore II"  snm="T.II"
V:B1 middle=d clef=bass     name="Basso I"   snm="B.I"  transpose=-24
V:B2 middle=d clef=bass     name="Basso II"  snm="B.II" transpose=-24
K:Gm
%              End of header, start of tune body:
% 1
[V:T1] (B2c2 d2g2) | f6e2      | (d2c2 d2)e2 | d4 c2z2 |
[V:T2] (G2A2 B2e2) | d6c2      | (B2A2 B2)c2 | B4 A2z2 |
```



```

[V:B1]      z8      | z2f2 g2a2 | b2z2 z2 e2 | f4 f2z2 |
[V:B2]      x8      |      x8      |      x8      |      x8      |
% 5
[V:T1]  (B2c2 d2g2) | f8      | d3c (d2fe) | H d6      ||
[V:T2]      z8      |      z8      | B3A (B2c2) | H A6      ||
[V:B1]  (d2f2 b2e'2) | d'8     | g3g  g4    | H~f6      ||
[V:B2]      x8      | z2B2 c2d2 | e3e (d2c2) | H d6      ||

```

This layout closely resembles printed music, and permits the corresponding notes on different voices to be vertically aligned so that the chords can be read directly from the abc. The addition of single remark lines “%” between the grouped staves, indicating the bar numbers, also makes the source more legible.

Here follows the visible output:

V: can appear both in the body and the header. In the latter case, V: is used exclusively to set voice properties. For example, the **name** property in the example above, specifies which label should be printed on the first staff of the voice in question. Note that these properties may be also set or changed in the tune body. The V: properties are fully explained below.

Please note that the exact grouping of voices on the staff or staves is not specified by V: itself. This may be specified with the **%%score** stylesheet directive. See voice grouping for details.

For playback, see instrumentation directives for details of how to assign a General MIDI instrument to a voice using a **%%MIDI** stylesheet directive.

Although it is not recommended, the tune body of fragment X:1, could also be notated this way:

```

X:2
T:Zocharti Loch
%...skipping rest of the header...
K:Gm
%
%           Start of tune body:
V:T1
  (B2c2 d2g2) | f6e2 | (d2c2 d2)e2 | d4 c2z2 |
  (B2c2 d2g2) | f8 | d3c (d2fe) | H d6 ||
V:T2
  (G2A2 B2e2) | d6c2 | (B2A2 B2)c2 | B4 A2z2 |
  z8 | z8 | B3A (B2c2) | H A6 ||
V:B1
  z8 | z2f2 g2a2 | b2z2 z2 e2 | f4 f2z2 |
  (d2f2 b2e'2) | d'8 | g3g g4 | H^f6 ||
V:B2
  x8 | x8 | x8 | x8 |
  x8 | z2B2 c2d2 | e3e (d2c2) | H d6 ||

```

In the example above, each **V:** label occurs only once, and the complete part for that voice follows. The output of tune **X:2** will be exactly the same as the output of tune **X:1**; the source code of **X:1**, however, is much easier to read.

7.1 Voice properties

VOLATILE: See above.

V: fields can contain voice specifiers such as name, clef, and so on. For example,

```
V:T name="Tenor" clef=treble-8
```

indicates that voice **T** will be drawn on a staff labelled **Tenor**, using the treble clef with a small **8** underneath. Player programs will transpose the notes by one octave. Possible voice definitions include:

- **name=“voice name”** - the voice name is printed on the left of the first staff only. The characters `\n` produce a newline in the output.
- **subname=“voice subname”** - the voice subname is printed on the left of all staves but the first one.
- **stem=up/down** - forces the note stem direction.
- **clef=** - specifies a clef; see clefs and transposition for details.

The name specifier may be abbreviated to **nm=**. The subname specifier may be abbreviated to **snm=**.

Applications may implement their own specifiers, but must gracefully ignore specifiers they don't understand or implement. This is required for portability of abc files between applications.

7.2 Breaking lines

VOLATILE: See above. In particular the following may be relaxed with the introduction of a control voice.

The rules for typesetting line-breaks in multi-voice abc tunes are the same as for single voice music although additionally a line-break in one voice must be matched in the other voices. See the example tune Canzonetta.abc.

7.3 Inline fields

VOLATILE: See above.

To avoid ambiguity, inline fields that specify music properties should be repeated in every voice to which they apply.

Example:

```
[V:1] C4| [M:3/4] CEG|Gce|
[V:2] E4| [M:3/4] G3 |E3 |
```

7.4 Voice overlay

VOLATILE: See above.

The `&` operator may be used to temporarily overlay several voices within one measure. Each `&` operator sets the time point of the music back by one bar line, and the notes which follow it form a temporary voice in parallel with the preceding one. This may only be used to add one complete bar's worth of music for each `&`.

Example:

```
A2 | c d e f g a &\
    A A A A A A &\
    F E D C B, A, |]
```



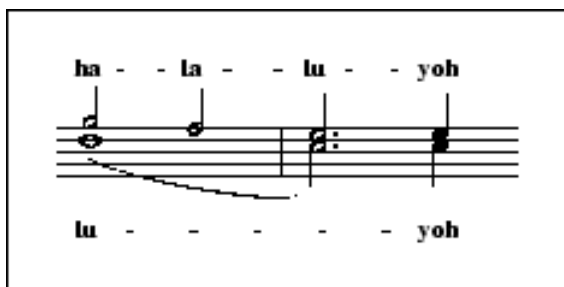
Words in **w:** lines (and symbols in **s:** lines) are matched to the corresponding notes as per the normal rules for lyric alignment (see lyrics), disregarding any overlay in the accompanying music code.

Example:

```

      g4 f4 | e6 e2 |
&& (d8      | c6) c2|
w: ha-la- | lu-yoh
+: lu-    | -yoh

```



This revokes the abc 2.0 usage of **&** in **w:** and **s:** lines, which is now deprecated (see disallowed).

8. Abc data format

Each line in the file may end with white-space which will be ignored. For the purpose of this standard, ASCII tab and space characters are equivalent and are both included in the term ‘white-space’. Applications must be able to interpret end-of-line markers in Unix (<LF>), Windows/DOS (<CR><LF>), and Macintosh style (<CR>) correctly.

8.1 Tune body

Within the tune body, all the printable ASCII characters may be used for the music code. These are:

```

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ `
a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

```

Of these, the following characters are currently reserved:

```

# * ; ? @

```

In future standards they may be used to extend the abc syntax.

To ensure forward compatibility, current software should ignore these characters when they appear inside or between note groups, possibly giving a warning. However, these characters may not be ignored when they appear inside text strings or information fields.

Example:

```
@a !pp! #bc2/3* [K:C#] de?f "@this $2was difficult to parse?" y |**  
should be treated as:  
a !pp! bc2/3 [K:C#] def "@this $2was difficult to parse?" y |
```

8.2 Text strings

Text written within an abc file, either as part of an information field, an annotation or as free text / typeset text, is known as a **text string**, or more fully, an **abc text string**. (Note that the abc standard version 2.0 referred to a text string as an *abc string*.)

Typically when there are several lines of text, each line forms a separate text string, although the distinction is not essential.

The contents of a text string may be written using any legal character set. The default character set is **utf-8**, giving access to every Unicode character.

However, not all text editors support **utf-8** and so to avoid portability problems when writing accented characters in text strings, it is also possible to use three other encoding options:

- **mnemonics** - for example, é can be represented by `\'e`. These mnemonics are based on TeX encodings and are always in the format *backslash-mnemonic-letter*. They have been available since the earliest days of abc and are widely used in legacy abc files. They are generally easy to remember and easy to read, but are not comprehensive in terms of the possible accents they can represent.
- **named html entities** - for example, é can be represented by `´`. These encodings are not common in legacy abc files but are convenient for websites which use abc and generally easy to remember. However they are not particularly easy to read and are not fully comprehensive in terms of the possible accents they can represent.
- **fixed width unicode** - for example, é can be represented by `\u00e9` using the 16-bit unicode representation `00e9` (or `\U000000e9` using 32-bit). These encodings are not common in legacy abc files and are not easy to read but give comprehensive access to all unicode characters.

All conforming abc typesetting software should support (understand and be able to convert) the subset of accents and ligatures given in the appendix, supported accents & ligatures, together with the special characters and symbols listed below.

A summary, with examples, is as follows:

Accent	Examples	Encodings
grave	À à È è	\`A \`a \`E \`e
acute	Á á É é	\'A \'a \'E \'e
circumflex	Â â Ê ê	\^A \^a \^E \^e
tilde	Ã ã Ñ ñ	\~A \~a \~E \~e
umlaut	Ä ä Ö ö	\"A \"a \"E \"e
cedilla	Ç ç	\cC \cc
ring	Å å	\AA \aa
slash	Ø ø	\O \o
breve	Ă ă Ę ę	\uA \ua \uE \ue
caron	Š š Ž ž	\vS \vs \vZ \vz
double acute	Ő ő Ű ú	\HO \Ho \HU \Hu
ligatures	ß Æ æ œ	\ss \AE \ae \oe

Programs that have difficulty typesetting accented letters may reduce them to the base letter or, in the case of ligatures, the two base letters ignoring the backslash.

Examples: When reduced to the base letter, \oA becomes A, \"o becomes o, \ss becomes ss, \AE becomes AE, etc.

For fixed width unicode, \u or \U must be followed by 4 or 8 hexadecimal characters respectively. Thus if any of the 4 characters after \u is not hexadecimal, then it is interpreted as a breve.

Special characters

Characters that are meaningful in the context of a text string can be escaped using a backslash as follows:

- type \\ to get a backslash;
- type \% to get a percent symbol that is not interpreted as the start of a comment;
- type \& to get an ampersand that is not interpreted as the start of a named html entity (although an ampersand followed by white-space is interpreted as is - for example, gin & tonic is OK, but G\&T requires the backslash);
- type " or \u0022 to get double quote marks in an annotation

Special symbols

The following symbols are also useful:

- type `©` or `\u00a9` for the copyright symbol ©
- type `\u266d` for a flat symbol
- type `\u266e` for a natural symbol
- type `\u266f` for a sharp symbol

VOLATILE: Finally note that currently the specifiers `$1`, `$2`, `$3` and `$4` can be used to change the font within a text string. However, this feature is likely to change in future versions of the standard - see font directives for more details.

9. Macros

This standard defines an **optional** system of macros which is principally used to define the way in which ornament symbols such as the tilde ~ are played (although it could be used for many other purposes).

Software implementing these macros, should first expand the macros defined in this section, and only afterwards apply any relevant U: replacement (see Redefinable symbols).

When these macros are stored in an abc header file (see include field), they may form a powerful library.

There are two kinds of macro, called Static and Transposing.

9.1 Static macros

You define a static macro by writing into the tune header something like this:

```
m: ~G3 = G{A}G{F}G
```

When you play the tune, the program searches the tune header for macro definitions, then does a search and replace on its internal copy of the text before passing that to the parser which plays the tune. Every occurrence of `~G3` in the tune is replaced by `G{A}G{F}G`, and that is what gets played. Only `~G3` notes are affected, `~G2`, `~g3`, `~F3` etc. are ignored.

You can put in as many macros as you want, and indeed, if you only use static macros you will need to write a separate macro for each combination of pitch and note-length. Here is an example:

```

X:50
T:Apples in Winter
S:Trad, arr. Paddy O'Brien
R:jig
m: ~g2 = {a}g{f}g
m: ~D2 = {E}D{C}D
M:6/8
K:D
G/2A/2|BEE dEE|BAG FGE|~D2D FDF|ABc ded|
BEE BAB|def ~g2 e|fdB AGF|GEE E2:|
d|efe edB|ege fdB|dec dAF|DFA def|
[1efe edB|def ~g2a|bgB afa|gee e2:|
[2edB def|gba ~g2e|fdB AGF|GEE E2||

```

Here I have put in two static macros, since there are two different notes in the tune marked with a tilde.

A static macro definition consists of four parts:

- the field identifier `m:`
- the target string - e.g. `~G3`
- the equals sign
- the replacement string - e.g. `G{A}G{F}G`

The target string can consist of any string up to 31 characters in length, except that it may not include the letter 'n', for reasons which will become obvious later. You don't have to use the tilde, but of course if you don't use a legal combination of abc, other programs will not be able to play your tune.

The replacement string consists of any legal abc text up to 200 characters in length. It's up to you to ensure that the target and replacement strings occupy the same time interval (the program does not check this). Both the target and replacement strings may include spaces if necessary, but leading and trailing spaces are stripped off so

```
m:~g2={a}g{f}g
```

is perfectly OK, although less readable.

9.2 Transposing macros

If your tune has ornaments on lots of different notes, and you want them to all play with the same ornament pattern, you can use transposing macros to achieve this. Transposing macros are written in exactly the same way as static macros, except that the note symbol in the target string is represented by 'n' (meaning any note) and the note symbols in the replacement string by other

letters (h to z) which are interpreted according to their position in the alphabet relative to n.

So, for example I could re-write the static macro `m: ~G3 = G{A}G{F}G` as a transposing macro `m: ~n3 = n{o}n{m}n`. When the transposing macro is expanded, any note of the form `~n3` will be replaced by the appropriate pattern of notes. Notes of the form `~n2` (or other lengths) will be ignored, so you will have to write separate transposing macros for each note length.

Here's an example:

```
X:35
T:Down the Broom
S:Trad, arr. Paddy O'Brien
R:reel
M:C|
m: ~n2 = (3o/n/m/ n          % One macro does for all four rolls
K:ADor
EAAG~A2 Bd|eg~g2 egdc|BGGF GAGE|~D2B,D GABG|
EAAG ~A2 Bd|eg~g2 egdg|eg~g2 dgba|gedB BAA2:|
~a2ea agea|agbg agef|~g2dg Bgdg|gfga gedel
~a2 ea agea|agbg ageg|dg~g2 dgba|gedB BA A2:|
```

A transposing macro definition consists of four parts:

- the field identifier `m:`
- the target string - e.g `~n3`
- the equals sign
- the replacement string - e.g. `n{o}n{m}n`

The target string can consist of any string up to 31 characters in length, except that it must conclude with the letter 'n', followed by a number which specifies the note length.

The replacement string consists of any legal abc text up to 200 characters in length, where note pitches are defined by the letters h - z, the pitches being interpreted relative to that of the letter n. Once again you should ensure that the time intervals match. You should not use accidentals in transposing macros

Comment: It is almost impossible to think of a way to transpose `~a3` or `~G2` which will work correctly under all circumstances, so a static macro should be used for cases like these.

10. Outdated syntax

The abc standard contains a variety of outdated syntax that is no longer recommended or, in some cases even supported, according to the following definitions:

- **Deprecated** syntax is rules or constructs that have been outdated by newer syntax. Deprecated syntax must be supported by conforming abc software under strict interpretation but is not recommended for new transcriptions. Deprecated syntax may become obsolete in future versions of abc. Conforming abc software that encounters deprecated syntax should issue a warning when using strict interpretation (although it may offer the user the option to switch warnings off).
- **Obsolete** syntax is rules or constructs for which there is no guarantee of support by conforming abc software. Obsolete syntax may be supported under loose interpretation but must not be used for new transcriptions. Conforming abc software that encounters obsolete syntax should issue a (preferably non-fatal) error message when using strict interpretation, or a warning when using loose interpretation (although it may offer the user the option to switch warnings off).
- **Disallowed** syntax has the same definition as obsolete syntax, but has not gone through a formal process of deprecation.
- **Outdated** syntax is the collective term for deprecated, obsolete and disallowed syntax.

Please see <http://www.ietf.org/rfc/rfc2119.txt> for formal definitions of the key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this context.

Outdated abc syntax is listed below so that users who come across it are able to interpret (and preferably update) it according to the latest standard.

10.1 Outdated information field syntax

The **A:** field was originally used to contain area information. In version 2.0 this was changed to contain the name of the lyrics author. In version 2.1, to maintain backwards compatibility, this has been changed back to area, but for clarity, the **A:** field is deprecated - area information can be stored in the **O:** field and a new field (*to be decided*) will be used for author information.

Comment: Of the 160,000 tunes currently available in the abcnotation.com tune search 16,300 contain an **A:** field with 680 distinct values. Of these, only around 10 contain author information rather than area (in some cases it is difficult to tell).

An **E:** field was once used by `abc2mtex` to explicitly control note spacing; this is no longer necessary with current formatting algorithms and the **E:** field is now

deprecated.

The original usage of the **H:** history field, where the contents of the history field is considered to continue over several lines until the next field occurs, is now deprecated.

The **Q:** tempo field is still very much in use, but earlier versions of the standard permitted two syntax variants, now deprecated, which specified how many unit note lengths to play per minute.

Examples: Both examples mean “play 120 unit note-lengths per minute”.

Q:C=120

Q:120

This is not very musical, and the usage is deprecated. However, there are many abc files which employ this syntax and programs should accept it.

10.2 Outdated dialects

10.2.1 Outdated line-breaking

The popular abc software abc2win introduced an exclamation mark (!) as a way of forcing a score line-break and this was adopted by abc 2.0, conflicting with the previous usage of !...! to delimit decorations.

The ! is now deprecated for score line-breaks although it is still available (even under strict interpretation) for legacy abc transcriptions by use of the **"I:linebreak !"** directive - see line-breaking dialects.

10.2.2 Outdated decorations

Abc standard 2.0 adopted **+...+** syntax to indicate decorations in place of !...!. It never gained much favour, however, and the latter is in much more common (see decoration dialects).

Therefore, and since a non-conflicting mechanism has now been found to allow ! for line-breaking (see line-breaking dialects), the **+...+** decoration syntax is now deprecated in favour of !...!.

Nonetheless, the **+...+** decoration syntax is still available using the **"I:decoration +"** instruction (see decoration dialects).

10.2.3 Outdated chords

Early versions of the abc standard used the **+** symbol to delimit chords (in place of **[]** symbols). This usage is now deprecated - see chord dialects for more details.

10.3 Outdated continuations

From the earliest days of abc (in abc standard 1.0 through to abc 1.7.6), the backslash (\) has been used to suppress score line-breaks by placing it at the end of a line of music code. Thus, effectively, it has acted as a continuation character, although with its own special rules, in particular that it could act through information fields and comments.

Abc 2.0 extended this usage to make \ a general continuation character, which also allows continuation of information fields and in particular the **w:** lyrics field (usually the only field which actually requires a continuation in typical transcriptions). Unfortunately, however, the rules of precedence were never well established (should the \ be treated by a pre-processor joining together continued lines and, if so, should comments be removed before or after that happened?) and the usage was never widely adopted, nor even well understood.

Comment: Of the 160,000 tunes currently available in the abcnotation.com tune search, only 22 (0.01%) use continuations for the **w:** field and only around 50 (0.03%) use it for any other field; of these latter usages, almost all are actually in error.

Furthermore, discussions during the development of abc 2.1 led to the suggestion that a new character should be introduced to suppress score line-breaks - in other words the \ (as described in abc 2.0) had evolved far enough away from its initial definition so that another character was required to replace what it had originally been designed to do.

Consequently, in abc 2.1 the \ has been reinstated to its original purpose of suppressing score line-breaks (see typesetting line-breaks) and its use as a general continuation character is now disallowed (see continuation of input lines for the alternatives).

10.4 Outdated directives

The `%%continueall true` directive is replaced by `"I:linebreak !"` in abc 2.1 (see typesetting line-breaks) and deprecated.

The `%%abc-copyright` and `%%abc-edited-by` extended information fields from section 3.3. of abc 2.0 have been deprecated in favour of the **Z:** - transcription field.

Comment: Of the 131,000 files currently available in the abcnotation.com tune search only 32 use `%%abc-edited-by` and only 1 uses `%%abc-copyright`.

10.5 Outdated file structure

10.5.1 Outdated tune header syntax

Abc standard 2.0 included the rule that 'if the abc file contains only one tune the **X:** field may be dropped'. However, it was pointed out that as a consequence, a user who pasted an additional tune into such a file would get an error message from a tune which previously contained no errors.

Despite considerable discussion on the abcusers mail list (see for example the threads <http://tech.groups.yahoo.com/group/abcusers/message/3950> and <http://tech.groups.yahoo.com/group/abcusers/message/4113>) and, a number of good suggestions, no consensus was reached. As a result the above rule is deprecated in abc 2.1; a tune must start with a **X:** field followed by a **T:** field.

However, this decision may be revisited in the future and the specification relaxed.

10.5.2 Outdated defaults

In early versions of the abc standard, defaults could be set throughout an abc file, using information fields, which applied to all subsequent tune. In other words, the file header could effectively appear anywhere inside a file instead of just at the top.

This usage significantly complicates random access of the tunes in the file, since the all the preceding contents of the file must be scanned for default settings before a tune can be processed. As result this was deprecated in abc 2.0 and is deprecated in abc 2.1.

10.6 Outdated lyrics alignment

Abc 2.1 introduced an extension to lyrics alignment meaning that lyric lines (i.e. those using the **w:** field) no longer need to follow immediately after the line of music code to which they are attached, meaning that they can even be postponed to the end of the tune body.

Examples: The following two excerpts are equivalent in abc 2.1; under abc 2.0 and previous versions of the standard, only the first version would be legal. Note that there are 4 (numbered) verses and hence 4 **w:** fields for each line of music code.

In the first excerpt the lyrics follow immediately after the line of music code to which they are attached.

In the second excerpt, the lyrics are postponed to the end of the tune, arguably aiding readability substantially and meaning that each verse is contiguous. The comment lines in the second excerpt (those lines beginning with **%**) are added for readability and are entirely optional.

```
D2DE G2GG|A2EE ED-D2|c2cc B2AG|
w:1\--Si les ma-tins de gri-sail-le se tein-tent,*s'ils ont cou-leur en la
```

```

w:2\--Si mo-ri-bonds sont les rois en ri-pail-le,*si leurs pri-sons sont des
w:3\--Si mill' so-leils de mé-tal pren-nent voi-le,*dix mill' so-leils de cris-
w:4\--Si mill' bri-gands à l'en-can font par-ta-ge,*dix mille en-fants des tor-
A2B~c d4|e2ee d2BA|G2EF GABc|
w:nuit qui s'é-teint, vien-dront d'o-pal's len-de-mains, re-vien-dront les siè-cles
w:ca-ges sans fond, vien-ne l'heur' des é-va-sions,*****
w:\-tal font mer-veille vienn'nt des lu-eurs de ver-meil,*****
w:\-rents font ar-gent, vien-nent des fleurs de sa-fran,*****

%
% music
%
D2DE G2GG|A2EE ED-D2|c2cc B2AG|
A2B~c d4|e2ee d2BA|G2EF GABc|
%
% lyrics
%
w:1\--Si les ma-tins de gri-sail-le se tein-tent,*s'ils ont cou-leur en la
+:nuit qui s'é-teint, vien-dront d'o-pal's len-de-mains, re-vien-dront les siè-cles
%
w:2\--Si mo-ri-bonds sont les rois en ri-pail-le,*si leurs pri-sons sont des
+:ca-ges sans fond, vien-ne l'heur' des é-va-sions,*****
%
w:3\--Si mill' so-leils de mé-tal pren-nent voi-le,*dix mill' so-leils de cris-
+:\-tal font mer-veille vienn'nt des lu-eurs de ver-meil,*****
%
w:4\--Si mill' bri-gands à l'en-can font par-ta-ge,*dix mille en-fants des tor-
+:\-rents font ar-gent, vien-nent des fleurs de sa-fran,*****

```

Unfortunately, however, this extension is not fully backwards compatible with abc 2.0.

The difficulty arises when there is a line (or lines) of music code without lyrics attached, followed by a line with lyrics attached.

Example: In the following excerpt, using abc 2.0 the lyrics would be aligned with the adjacent music code, i.e. with `cdef`; using abc 2.1 they would be aligned at the start of the tune (or voice), i.e. with `CDEF`.

```

CDEF|
FEDC|
cdef|]
w:these are lyr-ics

```

The work around for users who have files with such usage is either to avoid writing `%abc-2.1` as the file identifier or to add an empty `w:` field after the final line of music code that should be without lyrics.

Example: The following excerpt should be treated the same way (with regard

to lyrics alignment) under abc 2.0 and abc 2.1. Under abc 2.1 the empty **w:** field means that the lyrics are aligned with **cdef**.

```
CDEF|  
FEDC|  
w:  
cdef|]  
w:these are lyr-ics
```

10.7 Other outdated syntax

10.7.1 Disallowed voice overlay

Although the use of ampersand (&) to overlay voices (as introduced in abc 2.0) is still perfectly acceptable, this usage has been deprecated within **w:** lyric and **s:** symbol information fields.

The reason is that, as far as is known, this usage has never been implemented in software and, furthermore, & symbols are widely used within **w:** fields in legacy abc files to indicate ampersands.

Instead lyrics are matched to notes without regard to the voice overlay - see voice overlay.

11. Stylesheet directives and pseudo-comments

11.0 Introduction to directives

11.0.1 Disclaimer

In the early days of abc, pseudo-comments (lines starting with **%%**) were introduced as a means of adding software-specific information and formatting instructions into abc files; because they started with a **%** symbol software that didn't recognise them would ignore them as a comment.

In a valiant effort, abc 2.0 made an attempt to standardise these pseudo-comments with the introduction stylesheet directives and the abc stylesheet specification. This was described as “not part of the ABC specification itself” but as “an additional standard” containing directives to control how the content and structural information described by the abc code “is to be actually rendered, for example by a typesetting or player program”.

Unfortunately, however, there are a very large number of pseudo-comment directives and not all of them are well-defined. Furthermore, some directives, in particular the text directives and accidental directives, actually contain content and / or structural information (as opposed to rendering instructions).

Abc 2.1 has stepped away from this approach somewhat.

The pseudo-comments are still very much accepted as a way for developers to introduce experimental features and software-specific formatting instructions. However, when a directive gains acceptance, either by being implemented in more than one piece of software, or by its use in a substantial body of tunes, the aim is that the usage will be standardised and adopted in the standard and the `I:` instruction form recommended in place of the `%%` pseudo-comment form.

In particular, it is intended that abc 2.3 will address markup and embedding and at that point a number of the text-based directives, together with other widely accepted forms, will be formally incorporated.

For the moment, section 11 is retained mostly unchanged from abc 2.0 (save for typo corrections) but, as a result of the foregoing, the whole of section 11 and all stylesheet directives should be regarded as *VOLATILE*.

11.0.2 Stylesheet directives

A **stylesheet directive** is a line that starts with `%%`, followed by a directive that gives instructions to typesetting or player programs.

Examples:

```
%%papersize A4
%%newpage
%%setbarnb 10
```

Alternatively, any stylesheet directive may be written as an `I:instruction` field although this is not recommended for usages which have not been standardised (i.e. it is not recommended for any directives described in section 11).

Examples: Not recommended.

```
I:papersize A4
I:newpage
I:setbarnb 10
```

Inline field notation may be used to place a stylesheet directive in the middle of a line of music:

Example:

```
CDEFG|[I:setbarnb 10]ABc
```

If a program doesn't recognise a stylesheet directive, it should just ignore it.

It should be stressed that the stylesheet directives are not formally part of the abc standard itself. Furthermore, the list of possible directives is long and not standardised. They are provided by a variety of programs for specifying layout, text annotations, fonts, spacings, voice instruments, transposition and other details.

Strictly speaking, abc applications don't have to conform to the same set of stylesheet directives. However, it is desirable that they do in order to make abc files portable between different computer systems.

11.1 Voice grouping

VOLATILE: This section is under review as part of the general discussion about multiple voices for abc 2.2. See also the section 11 disclaimer.

Basic syntax:

```
%%score <voice-id1> <voice-id2> ... <voice-idn>
```

The score directive specifies which voices should be printed in the score and how they should be grouped on the staves.

Voices that are enclosed by parentheses () will go on one staff. Together they form a voice group. A voice that is not enclosed by parentheses forms a voice group on its own that will be printed on a separate staff.

If voice groups are enclosed by curly braces {}, the corresponding staves will be connected by a big curly brace printed in front of the staves. Together they form a voice block. This format is used especially for typesetting keyboard music.

If voice groups or braced voice blocks are enclosed by brackets [], the corresponding staves will be connected by a big bracket printed in front of the staves. Together they form a voice block.

If voice blocks or voice groups are separated from each other by a | character, continued bar lines will be drawn between the associated staves.

Example:

```
%%score Solo [(S A) (T B)] {RH | (LH1 LH2)}
```

If a single voice surrounded by two voice groups is preceded by a star (*), the voice is marked to be floating. This means that the voice won't be printed on its own staff; rather the software should automatically determine, for each note of the voice, whether it should be printed on the preceding staff or on the following staff.

Software that does not support floating voices may simply print the voice on the preceding staff, as if it were part of the preceding voice group.

Examples:

```
%%score {RH *M| LH}  
%%score {(RH1 RH2) *M| (LH1 LH2)}
```

String parts in an orchestral work are usually bracketed together and the top two (1st/2nd violins) then braced outside the bracket:

`%%score [{Vln1 | Vln2} | Vla | Vc | DB]`

Any voices appearing in the tune body will only be printed if it is mentioned in the score directive.

When the score directive occurs within the tune body, it resets the music generator, so that voices may appear and disappear for some period of time.

If no score directive is used, all voices that appear in the tune body are printed on separate staves.

See Canzonetta.abc for an extensive example.

An alternative directive to `%%score` is `%%staves`.

Both `%%score` and `%%staves` directives accept the same parameters, but measure bar indications work the opposite way. Therefore, `%%staves [S|A|T|B]` is equivalent to `%%score [S A T B]` and means that continued bar lines are not drawn between the associated staves, while `%%staves [S A T B]` is equivalent to `%%score [S|A|T|B]` and means that they are drawn.

11.2 Instrumentation directives

VOLATILE: See the section 11 disclaimer.

`%%MIDI voice [<ID>] [instrument=<integer> [bank=<integer>]] [mute]`

Assigns a MIDI instrument to the indicated abc voice. The MIDI instruments are organized in banks of 128 instruments each. Both the instruments and the banks are numbered starting from one.

The General MIDI (GM) standard defines a portable, numbered set of 128 instruments (numbered from 1-128) - see <http://www.midi.org/techspecs/gm1sound.php>. The GM instruments can be used by selecting bank one. Since the contents of the other MIDI banks is platform dependent, it is highly recommended to only use the first MIDI bank in tunes that are to be distributed.

The default bank number is 1 (one).

Example: The following assigns GM instrument 59 (tuba) to voice ‘Tb’.

`%%MIDI voice Tb instrument=59`

If the voice ID is omitted, the instrument is assigned to the current voice.

Example:

```
M:C
L:1/8
Q:1/4=66
K:C
V:Rueckpos
```

```

%%MIDI voice instrument=53 bank=2
A3B    c2c2    |d2e2    de/f/P^c3/d/|d8      |z8          |
V:Organo
%%MIDI voice instrument=73 bank=2
z2E2-  E2AG    |F2E2    F2E2          |F6  F2|E2CD    E3F/G/|

```

You can use the keyword `mute` to mute the specified voice.

Some abc players can automatically generate an accompaniment based on the chord symbols specified in the melody line. To suggest a GM instrument for playing this accompaniment, use the following directive:

```
%%MIDI chordprog 20 % Church organ
```

11.3 Accidental directives

VOLATILE: This section is under active discussion. See also the section 11 disclaimer.

```
%%propagate-accidentals not | octave | pitch
```

When set to `not`, accidentals apply only to the note they're attached to. When set to `octave`, accidentals also apply to all the notes of the same pitch in the same octave up to the end of the bar. When set to `pitch`, accidentals also apply to all the notes of the same pitch in all octaves up to the end of the bar.

The default value is `pitch`.

```
%%writeout-accidentals none | added | all
```

When set to `none`, modifying or explicit accidentals that appear in the key signature field (K:) are printed in the key signature. When set to `added`, only the accidentals belonging to the mode indicated in the K: field, are printed in the key signature. Modifying or explicit accidentals are printed in front of the notes to which they apply. When set to `all`, both the accidentals belonging to the mode and possible modifying or explicit accidentals are printed in front of the notes to which they apply; no key signature will be printed.

The default value is `none`.

11.4 Formatting directives

VOLATILE: See the section 11 disclaimer.

Typesetting programs should accept the set of directives in the next sections. The parameter of a directive can be a text string, a logical value `true` or `false`, an integer number, a number with decimals (just 'number' in the following), or a unit of length. Units can be expressed in cm, in, and pt (points, 1/72 inch).

The following directives should be self-explanatory.

11.4.1 Page format directives

VOLATILE: See the section 11 disclaimer.

%%pageheight	<length>
%%pagewidth	<length>
%%topmargin	<length>
%%botmargin	<length>
%%leftmargin	<length>
%%rightmargin	<length>
%%indent	<length>
%%landscape	<logical>

11.4.2 Font directives

VOLATILE: Font directives are due to be considered in abc 2.3 - see the section 11 disclaimer.

PostScript and PDF are the standard file formats for distributing printable material. For portability reasons, typesetters will use the PostScript font names. The size parameter should be an integer, but is optional.

%%titlefont		<size>	
%%subtitlefont		<size>	
%%composerfont		<size>	
%%partsfont		<size>	
%%tempofont		<size>	
%%gchordfont		<size>	% for chords symbols
%%annotationfont		<size>	% for "^..." annotations
%%infofont		<size>	
%%textfont		<size>	
%%vocalfont		<size>	% for w:
%%wordsfont		<size>	% for W:

The specifiers \$1, \$2, \$3 and \$4 can be used to change the font within a text string. The font to be used can be specified with the %%setfont-n directives. \$0 resets the font to its default value. \$\$ gives an actual dollar sign.

%%setfont-1		<size>
%%setfont-2		<size>
%%setfont-3		<size>
%%setfont-4		<size>

11.4.3 Space directives

VOLATILE: See the section 11 disclaimer.

%%topspace	<length>
%%titlespace	<length>

%%subtitlespace	<length>
%%composerspace	<length>
%%musicospace	<length> % between composer and 1st staff
%%partsspace	<length>
%%vocalspace	<length>
%%wordsspace	<length>
%%textspace	<length>
%%infospace	<length>
%%staffsep	<length> % between systems
%%sysstaffsep	<length> % between staves in the same system
%%barsperstaff	<integer>
%%parskipfac	<number> % space between parts
%%lineskipfac	<number> % space between lines of text
%%stretchstaff	<logical>
%%stretchlast	<logical>
%%maxshrink	<number> % shrinking notes
%%scale	<number>

11.4.4 Measure directives

VOLATILE: See the section 11 disclaimer.

%%measurefirst	<integer> % number of first measure
%%barnumbers	<integer> % bar numbers every 'n' measures
%%measurenb	<integer> % same as %%barnumbers
%%measurebox	<logical>
%%setbarnb	<integer> % set measure number

11.4.5 Text directives

VOLATILE: Text directives are due to be considered in abc 2.3 - see the section 11 disclaimer.

The following directives can be used for inserting typeset text within an abc file.

%%text	<text string>
%%center	<text string>
%%begintext	
%%...	<text string>
%%endtext	

Notes:

- %%text prints the following text, treated as a text string.
- %%center prints the following text, treated as a text string and centred.
- %%begintext and %%endtext mark a section of lines, each of which start with %, followed by some text. It is an alternative to several %%text lines.

[*Important note:* some extensions offered by abc software programs relax the rule that each line between `%%begintext` and `%%endtext` must start with `%%`. Whilst this should not cause problems for typeset text between tunes, typeset text within a tune header or tune body should respect this rule and, in particular, must not introduce blank lines.]

See further information about directives for more details and to find out about additional parameters for these directives.

Recommendation for users: If you are using text directives for tune-specific information, consider instead using one of the background information fields together with a `%%writefields` directive (see information directives) so that the information can be correctly identified by databasing software.

11.4.6 Information directives

VOLATILE: The `%%writefields` directive and its formatting options are likely to be enhanced when markup is considered in abc 2.3. See also the section 11 disclaimer.

`%%writefields <list of field identifiers> [<logical>]`

The `%%writefields` directive allows users to choose which string-type information fields appear in the printed score (see the information fields table for a list of string-type fields). It is followed by a list of field identifiers and, optionally, the logical value `true` or `false`. If the logical value is missing it is taken as `true`.

The `%%writefields` directive also applies to certain instruction fields - namely `X:reference number`, `P:parts` and `Q:tempo`.

The default is `%%writefields TCOPQwW` meaning that the title (T), composer (C), origin (O), parts (P), tempo (Q), aligned words (w) and other words (W) are printed out by default (see typesetting information fields for how these should be typeset). Each subsequent `%%writefields` directive combines with this list, rather than overriding it.

Examples:

```
%%writefields 0 false      % the 0 field is not printed out - other defaults remain
%%writefields X           % the X: field is printed out
%%writefields BCDFGHNORSTWwXZ % all string-type fields are printed out
```

Typesetting software conforming to abc 2.1 may format the information strings in any way it chooses.

Comment: The `%%writefields` directive can be used in place of a number of directives introduced in abc 2.0:

- `%%writefields X` can be used as an alternative to `%%withxrefs`

- `%%writefields Ww false` can be used as an alternative to `%%musiconly`
- `%%writefields` is a partial alternative to `%%writehistory` and `%%infoname`

See further information about directives for more details of the 2.0 alternatives.

11.4.7 Separation directives

VOLATILE: See the section 11 disclaimer.

```
%%sep      % draw a horizontal separator, i.e. a line
%%vskip    % insert some vertical space
%%newpage  % start a new page
```

See further information about directives for more details and to find out about additional parameters for these directives.

11.4.8 Miscellaneous directives

VOLATILE: See the section 11 disclaimer.

```
%%exprabove    <logical>
%%exprbelow    <logical>
%%graceslurs   <logical> % grace notes slur to main note
%%infoline     <logical> % rhythm and origin on the same line
%%oneperpage   <logical>
%%vocalabove   <logical>
%%freegchord   <logical> % print '#', 'b' and '=' as they are
%%printtempo   <logical>
```

The default value for these directives is false.

11.5 Application specific directives

Applications may introduce their own directives. These directives should start with the name of the application, followed a colon, followed by the name of the directive.

Example:

```
%%noteedit:fontcolor blue
```

11.6 Further information about directives

Since stylesheet directives are not formally part of the abc standard, only a subset is included here. For additional directives and further information about

those listed here, see the user manuals for programs that implement them, in particular:

- the `format.txt` file included with `abcm2ps`
 - the `abcguides.txt` file included with `abcMIDI`
 - the `abctab2ps` User's guide
-

12. Dialects, strict / loose interpretation and backwards compatibility

Unfortunately a number of dialects of `abc` have arisen over the years, partly due to differences in implementation, together with unfinished drafts of the `abc` standard and ambiguities within it.

Version 2.1 of the standard aims to address this fragmentation of `abc` notation with a robust, but tolerant approach that should accommodate as many users as possible for several years to come and, as far as possible, restore backwards compatibility.

There are three main approaches:

- the introduction of new `I:` directives to allow for preferences in dialects;
- the concepts of strict and loose interpretation of the standard (together with recommendations to software developers for dealing with loose interpretations);
- statistically-based decisions about default settings.

The aim is that, even under strict interpretation, most current dialects are still available via the new `I:` directives.

Comment: Dialects not available under strict interpretation are those where one symbol is used for two different purposes - for example, a `!` symbol used to denote both line-breaks and decorations; fortunately, of the 160,000 tunes currently available in the `abcnotation.com` tune search only around 60 (0.04%) employ this usage.

12.1 Dialect differences

The main differences that have arisen are line-breaks, decoration delimiters and chord delimiters.

12.1.1 Line-breaking dialects

By default, a (forced) score line-break is typeset by using a code line-break - see typesetting line-breaks.

In the past the `!` symbol has instead been used to indicate score line-breaks - this symbol is now used to denote decorations.

Comment: The `!` symbol was introduced by `abc2win`, a very popular program in its time, although now moribund. Of the 160,000 tunes currently available in the `abcnotation.com` tune search, only around 1,600 (10%) use the `!` symbol to denote line-breaks.

Although the use of the `!` symbol for line-breaking is now deprecated (see outdated line-breaking), users who wish to continue using the `!` symbol for line-breaking merely need to include the `"I:linebreak !"` directive, either in the file header or individually tune by tune - see typesetting line-breaks.

Example: The following abc code would result in two lines of music.

```
I:linebreak !
K:G
ABC DEF|!FED ABC|]
```

Finally a new line-breaking symbol, `$`, has been introduced as an alternative to using code line-breaks.

Comment: The `$` symbol is effectively a replacement for `!`. It is aimed at those users who want `!` as the decoration delimiter but who prefer to use code line-breaks without generating corresponding score line-breaks.

12.1.2 Decoration dialects

Decorations are delimited using the `!` symbol - see decorations.

In the past the `+` symbol has instead been used to denote decorations - this symbol is now deprecated for decorations.

Comment: Decorations were first introduced in draft standard 1.7.6 (which was never formally adopted) with the `!` symbol. In `abc 2.0` (adopted briefly whilst discussions about `abc 2.1` were taking place) this was changed to the `+` symbol. Neither are in widespread use, but the `!` symbol is much more common - of the 160,000 tunes currently available in the `abcnotation.com` tune search, only around 100 (0.07%) use the `+` symbol to delimit decorations, whereas around 1,350 (0.85%) use the `!` symbol.

Users who wish to continue using the `+` symbol for decorations merely need to include the `"I:decoration +"` directive, either in the file header or individually tune by tune - see decorations. All `+...+` decorations will then be treated as if they were the corresponding `!...!` decoration and any `!...!` decorations will generate an error message.

Note that the `"I:decoration +"` directive is automatically invoked by the `"I:linebreak !"` directive. Also note that the `!+!` decoration has no + equivalent - `+plus+` should be used instead.

Recommendation for users: Given the very small uptake of the + symbol for decorations, `"I:decoration +"` directive is not recommended. However, it is retained for users who wish to use the ! symbol for line-breaking in legacy abc files.

For completeness the `"I:decoration !"`, the default setting, is also available to allow individual tunes to use `!...!` decorations in a file where `"I:decoration +"` is set in the file header.

12.1.3 Chord dialects

Chords are delimited using `[]` symbols - see chords and unisons.

In the past the + symbol has instead been used to delimit chords - this symbol is no longer in use for chords.

Comment: In early versions of the abc standard (1.2 to 1.5), chords were delimited with + symbols. However, this made it hard to see where one chord ended and another began and the chord delimiters were changed to `[]` in 1.6 (November 1996). Of the 160,000 tunes currently available in the abcnotation.com tune search, only around 420 (0.25%) use the + symbol to delimit chords. Given the small uptake and the successful introduction of the `[]` symbols, there is no `I:` directive available which allows the use of + symbols and this usage is now obsolete.

12.2 Loose interpretation

Comment: There are around 160,000 tunes currently available in the abcnotation.com tune search - loose interpretation of the abc standard maintains backwards compatibility without any changes required for this huge and valuable resource.

Any abc file without a version number, or with a version number of 2.0 or less (see abc file identification and version field), should be interpreted loosely. Developers should do their best to provide programs that understand legacy abc files, but users should be aware that loose interpretations may differ from one abc program to another.

Recommendation for users: Try to avoid loose interpretation if possible; loose interpretation means that if you pass abc notated tunes on to friends, or post them on the web, they may not appear as you hoped.

Recommendation 1 for developers: Do your best! The most difficult tunes to deal with are those which use the same symbol for two different purposes - in

particular the ! symbol for both decorations and line-breaking. Here is an algorithm for helping to deal with !**decoration**! syntax and ! line-breaks in the same tune:

When encountering a !, scan forward. If you find another ! before encountering any of |[:], a space, or the end of a line, then you have a decoration, otherwise it is a line-break.

Recommendation 2 for developers: Although moving towards strict interpretations should make life easier for everybody (developers and users alike), you should allow users to switch easily between strict and loose interpretation, perhaps via a command line switch or a GUI check-box. For example, a user who imports an old abc file may wish to see how it would be interpreted strictly, perhaps to establish how many strict errors need fixing.

12.3 Strict interpretation

Any abc file with an abc version number greater than or equal to 2.1 (see abc file identification and version field) should be interpreted strictly, with errors indicated to the user as such.

13. Sample abc tunes

13.1 English.abc

```
%abc-2.1
H:This file contains some example English tunes
% note that the comments (like this one) are to highlight usages
% and would not normally be included in such detail
O:England          % the origin of all tunes is England

X:1                % tune no 1
T:Dusty Miller, The % title
T:Binny's Jig      % an alternative title
C:Trad.            % traditional
R:DH               % double hornpipe
M:3/4              % meter
K:G                % key
B>cd BAG|FA Ac BA|B>cd BAG|DG GB AG:|
Bdd gfg|aA Ac BA|Bdd gfa|gG GB AG:|
BG G/2G/2G BG|FA Ac BA|BG G/2G/2G BG|DG GB AG:|
W:Hey, the dusty miller, and his dusty coat;
W:He will win a shilling, or he spend a groat.
```

W:Dusty was the coat, dusty was the colour;
W:Dusty was the kiss, that I got frae the miller.

X:2
T:Old Sir Simon the King
C:Trad.
S:Offord MSS % from Offord manuscript
N:see also Playford % reference note
M:9/8
R:SJ % slip jig
N:originally in C % transcription note
K:G
D|GFG GAG G2D|GFG GAG F2D|EFE EFE EFG|A2G F2E D2:|
D|GAG GAB d2D|GAG GAB c2D|[1 EFE EFE EFG|A2G F2E D2:|\ % no line-break in score
M:12/8 % change of meter
[2 E2E EFE E2E EFG|\ % no line-break in score
M:9/8 % change of meter
A2G F2E D2|]

X:3
T:William and Nancy
T:New Mown Hay
T:Legacy, The
C:Trad.
O:England; Gloucs; Bledington % place of origin
B:Sussex Tune Book % can be found in these books
B:Mally's Cotswold Morris vol.1 2
D:Morris On % can be heard on this record
P:(AB)2(AC)2A % play the parts in this order
M:6/8
K:G
[P:A] D|"G"G2G GBd|"C"e2e "G"dBG|"D7"A2d "G"BAG|"C"E2"D7"F "G"G2:|
[P:B] d|"G"e2d B2d|"C"gfe "G"d2d| "G"e2d B2d|"C"gfe "D7"d2c|
"G"B2B Bcd|"C"e2e "G"dBG|"D7"A2d "G"BAG|"C"E2"D7"F "G"G2:|
% changes of meter, using inline fields
[T:Slows] [M:4/4] [L:1/4] [P:C] "G"d2|"C"e2 "G"d2|B2 d2|"Em"gf "A7"e2|"D7"d2 "G"d2|\
"C"e2 "G"d2|[M:3/8] [L:1/8] "G"B2 d |[M:6/8] "C"gfe "D7"d2c|
"G"B2B Bcd|"C"e2e "G"dBG|"D7"A2d "G"BAG|"C"E2"D7"F "G"G2:|

13.2 Strspys.abc

%abc-2.1
M:4/4
O:Scottish
R:Strathspey

```

X:1
T:A. A. Cameron's
K:D
e<A A2 B>G d>B|e<A A2 d>g (3fed|e<A A2 B>G d>B|B<G G>B d>g (3fed:|
B<e e>f g>e a>f|B<e e>f g>e (3fed|B<e e>f g>e a>f|d<B G>B d>g (3fed:|

X:2
T:Atholl Brose
% in this example, which reproduces Highland Bagpipe gracing,
% the large number of grace notes mean that it is more convenient to be specific about
% score line-breaks (using the $ symbol), rather than using code line breaks to indicate them
I:linebreak $
K:D
{gcd}c<{e}A {gAGAG}A2 {gef}e>A {gAGAG}Ad|
{gcd}c<{e}A {gAGAG}A>e {ag}a>f {gef}e>d|
{gcd}c<{e}A {gAGAG}A2 {gef}e>A {gAGAG}Ad|
{g}c/d/e {g}G>{d}B {gf}gG {dc}d>B:|$
{g}c<e {gf}g>e {ag}a>e {gf}g>e|
{g}c<e {gf}g>e {ag}a2 {GdG}a>d|
{g}c<e {gf}g>e {ag}a>e {gf}g>f|
{gef}e>d {gf}g>d {gBd}B<{e}G {dc}d>B|
{g}c<e {gf}g>e {ag}a>e {gf}g>e|
{g}c<e {gf}g>e {ag}a2 {GdG}ad|
{g}c<{GdG}e {gf}ga {f}g>e {g}f>d|
{g}e/f/g {Gdc}d>c {gBd}B<{e}G {dc}d2|]

```

13.3 Reels.abc

```

%abc-2.1
M:4/4
O:Irish
R:Reel

X:1
T:Untitled Reel
C:Trad.
K:D
eg|a2ab ageg|agbg agef|g2g2 fgag|f2d2 d2:|\
ed|cecA B2ed|cAcA E2ed|cecA B2ed|c2A2 A2:|
K:G
AB|cdec BcdB|ABAF GFE2|cdec BcdB|c2A2 A2:|

X:2
T:Kitchen Girl

```

```

C:Trad.
K:D
[c4a4] [B4g4]|efed c2cd|e2f2 gaba|g2e2 e2fg|
a4 g4|efed cdef|g2d2 efed|c2A2 A4:|
K:G
ABcA BAGB|ABAG EDEG|A2AB c2d2|e3f edcB|ABcA BAGB|
ABAG EGAB|cBAc BAG2|A4 A4:|

```

13.4 Canzonetta.abc

```

%abc-2.1
%%pagewidth      21cm
%%pageheight     29.7cm
%%topspace       0.5cm
%%topmargin      1cm
%%botmargin      0cm
%%leftmargin     1cm
%%rightmargin    1cm
%%titlespace     0cm
%%titlefont      Times-Bold 32
%%subtitlefont   Times-Bold 24
%%composerfont   Times 16
%%vocalfont      Times-Roman 14
%%staffsep       60pt
%%sysstaffsep    20pt
%%musicospace    1cm
%%vocalspace     5pt
%%measurenb      0
%%barsperstaff   5
%%scale          0.7
X: 1
T: Canzonetta a tre voci
C: Claudio Monteverdi (1567-1643)
M: C
L: 1/4
Q: "Andante mosso" 1/4 = 110
%%score [1 2 3]
V: 1 clef=treble name="Soprano"sname="A"
V: 2 clef=treble name="Alto" sname="T"
V: 3 clef=bass middle=d name="Tenor" sname="B"
%%MIDI program 1 75 % recorder
%%MIDI program 2 75
%%MIDI program 3 75
K: Eb
% 1 - 4

```

```

[V: 1] |:z4 |z4 |f2ec |ddcc |
w: Son que-sti-i cre-spi cri-ni~e
w: Que-sti son gli~oc-chi che mi-
[V: 2] |: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
w: Que-sti son~gli oc-chi che mi-ran - - - do fi-so mi-
[V: 3] |:z4 |f2ec|ddcf |(B/c/_d/e/)ff|
w: Son que-sti-i cre-spi cri-ni~e que - - - sto~il
w: Que-sti son~gli oc-chi che mi-ran - - - do
% 5 - 9
[V: 1] cAB2 |cAAA |c3B|G2!fermata!Gz ::e4|
w: que-sto~il vi-so ond' io ri-man-go-uc-ci-so. Deh,
w: ran-do fi-so, tut-to re-stai con-qui-so.
[V: 2] AAG2 |AFFF |A3F|=E2!fermata!Ez::c4|
w: que-sto~il vi-so ond' io ri-man-go-uc-ci-so. Deh,
w: ran-do fi-so tut-to re-stai con-qui-so.
[V: 3] (ag/f/e2)|A_ddd|A3B|c2!fermata!cz ::A4|
w: vi - - - so ond' io ti-man-go-uc-ci-so. Deh,
w: fi - - - so tut-to re-stai con-qui-so.
% 10 - 15
[V: 1] f_dec |B2c2|zAGF |\
w: dim-me-lo ben mi-o, che que-sto\
=EFG2 |1F2z2:|2F8|] % more notes
w: sol de-si-o_. % more lyrics
[V: 2] ABGA |G2AA|GF=EF |(GF3/2=E//D//E)|1F2z2:|2F8|]
w: dim-me-lo ben mi-o, che que-sto sol de-si - - - o_.
[V: 3] _dBc>d|e2AF|=EFc_d|c4 |1F2z2:|2F8|]
w: dim-me-lo ben mi-o, che que-sto sol de-si-o_.

```

14. Appendix

14.1 Supported accents & ligatures

Conforming abc software must support the following encodings for accents and ligatures. It may offer support for other named entities and hex unicode representations (which may be adopted by the standard at a later date).

For more details see text strings and for further information see, for example:

- <http://www.w3.org/TR/html4/sgml/entities.html>
- http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references
- http://en.wikipedia.org/wiki/List_of_Unicode_characters

- <http://www.fileformat.info/info/unicode/char/search.htm> - unicode character search

Accents:

Character	Mnemonic	Named html entity	16-bit hex unicode
À	\`A	À	\u00c0
à	\`a	à	\u00e0
È	\`E	È	\u00c8
è	\`e	è	\u00e8
Ì	\`I	Ì	\u00cc
ì	\`i	ì	\u00ec
Ò	\`O	Ò	\u00d2
ò	\`o	ò	\u00f2
Û	\`U	Ù	\u00d9
ù	\`u	ù	\u00f9
Á	\`'A	Á	\u00c1
á	\`'a	á	\u00e1
É	\`'E	É	\u00c9
é	\`'e	é	\u00e9
Í	\`'I	Í	\u00cd
í	\`'i	í	\u00ed
Ó	\`'O	Ó	\u00d3
ó	\`'o	ó	\u00f3
Ú	\`'U	Ú	\u00da
ú	\`'u	ú	\u00fa
Ý	\`'Y	Ý	\u00dd
ý	\`'y	ý	\u00fd
Â	\`^A	Â	\u00c2
â	\`^a	â	\u00e2
Ê	\`^E	Ê	\u00ca
ê	\`^e	ê	\u00ea
Î	\`^I	Î	\u00ce
î	\`^i	î	\u00ee
Ô	\`^O	Ô	\u00d4
ô	\`^o	ô	\u00f4
Û	\`^U	Û	\u00db
û	\`^u	û	\u00fb
Û	\`^Y	Ŷ	\u0176
Û	\`^y	ŷ	\u0177
Ã	\`~A	Ã	\u00c3
ã	\`~a	ã	\u00e3
Ñ	\`~N	Ñ	\u00d1
ñ	\`~n	ñ	\u00f1
Õ	\`~O	Õ	\u00d5

Character	Mnemonic	Named html entity	16-bit hex unicode
õ	\~o	õ	\u00f5
Ä	\"A	Ä	\u00c4
ä	\"a	ä	\u00e4
Ë	\"E	Ë	\u00cb
ë	\"e	ë	\u00eb
Ï	\"I	Ï	\u00cf
ï	\"i	ï	\u00ef
Ö	\"O	Ö	\u00d6
ö	\"o	ö	\u00f6
Ü	\"U	Ü	\u00dc
ü	\"u	ü	\u00fc
ÿ	\"Y	Ÿ	\u0178
ÿ	\"y	ÿ	\u00ff
Ç	\cC	Ç	\u00c7
ç	\cc	ç	\u00e7
Å	\AA	Å	\u00c5
å	\aa	&aaring;	\u00e5
Ø	\ /O	Ø	\u00d8
ø	\ /o	ø	\u00f8
Ă	\uA	Ă	\u0102
ă	\ua	ă	\u0103
Ě	\uE	not available	\u0114
ě	\ue	not available	\u0115
Š	\vS	Š	\u0160
š	\vs	š	\u0161
Ž	\vZ	Ž	\u017d
ž	\vz	ž	\u017e
Ő	\HO	not available	\u0150
ó	\Ho	not available	\u0151
Ű	\HU	not available	\u0170
ű	\Hu	not available	\u0171

Ligatures, etc:

Character	Mnemonic	Named html entity	16-bit hex unicode
Æ	\AE	Æ	\u00c6
æ	\ae	æ	\u00e6
Œ	\OE	Œ	\u0152
œ	\oe	œ	\u0153
ß	\ss	ß	\u00df
Ð	\DH	Ð	\u00d0
ð	\dh	ð	\u00f0

Character	Mnemonic	Named html entity	16-bit hex unicode
Þ	\TH	Þ	\u00de
þ	\th	þ	\u00fe

14.2 Errata

The following corrections have been made since the standard was published:

- Section 1.1.1 Terminology / definitions: The definition of *VOLATILE* has been clarified; it is used to indicate “sections which are under active discussion and/or are likely to change in some future version of the standard” rather than “sections which are under active discussion or likely to change at some point in the future” (8th Jan 2012).
- Section 6.1.1 Typesetting line-breaks: Typo: `setbarno` corrected to `setbarnb` in two places (8th Jan 2012).
- Section 8.2 Text strings: Typos for accent mnemonics (cedilla and ring): “\,C \,c” and “\oA \oa” corrected to “\cC \cc” and “\AA \aa”, respectively, as per Section 14.1 Supported accents & ligatures (8th Jan 2012). *TODO*: \, and \o are non-standard accent mnemonics introduced in abc 2.0; however, it is probably sensible to support them in addition to the standard, but less memorable, \c and \a.
- Section 11.4.6 Information directives: The statement “Note that the `%writefields` directive does not apply to instruction-type fields, such as parts (P) and tempo (Q)” has now been removed, as it conflicted with other information in the same section (8th Jan 2012).
- Section 4.18 Chord symbols: Typo: `sustained` corrected to `suspended` (26th May 2012).
- Section 4.6 Clefs and transposition: following discussion, this section has been corrected to clarify that the `middle` setting does not affect the playback (since there is no consistent way that it can do so). The Zocharti Loch example has been corrected and its accompanying midi file (which has persisted unchanged from abc draft 2.0, and which seemed to indicate `middle` might, in some circumstances, affect the playback) has been removed (20th February 2013).

— papersize: article documentclass: book fontsize: 10pt geometry: margin=.25in title: The abc music standard 2.1 (Dec 2011) —