musixser—a MusiXTEX extension package for voice-wise music coding (version 1.1)

© Rainer Dunker 1999–1999
rainer.dunker@ebe-online.de

May 22, 1999

(The page layout of this document is tailored to A4 paper format, but the text width and height can be freely changed in order to meet other paper sizes; no part of the layout relies on a particlular size.)

Contents

1 Introduction

1.1 Basic concept

`musixser` ("MusiXTEX SERial voices") is a set of TEX macros to be used with Taupin MusiXTEX (version T.90 or later). It changes MusiXTEX's coding style from chord or "column" oriented to voice oriented. Each voice of a score (which nevertheless is allowed to contain chords) is entered as a separate "thread" of musical events, thus imposing a polyphonic flavour on music coding.

`musixser` is no MusiXTEX preprocessor like PMX or M-Tx. It neither makes any layout decisions nor saves any typing work; it merely rearranges the input and passes everything over to MusiXTEX. The main benefits of using it—apart from the more intuitive, polyphonic coding—are:

- Repeated music structures (like voice-wise ostinato patterns) can be entered in a code-sharing manner (i.e. coded once, called multiple times).
- Much easier editing: voice-wise copying and pasting, rearranging and excluding of single or multiple staves and voices.
- There's no built-in limit on the number of voices per staff; the polyphony of arrangements can be arbitrarily complicated.

1.2 Restrictions and limitations

- `musixser` allocates one of TEX's token list registers per voice. If you use other packages which heavily use those registers, the possible number of voices may get seriously limited; nevertheless, I do not consider this to be likely during practical use.
- `musixser` does not support multi-staff instruments, i.e. all staves of such an instrument must be handled as (in MusiXTEX terms) separate one-staff instruments.

2 Installation

Put the `musixser.tex` file into a directory that is included in your TEXINPUTS search path—that's all. (Refer to your local TEX documentation for details on this.)

3 Entering music

Before using `musixser` macros, state

```
\input musixser
```

near the beginning of your source file but **after** the `\input musixtex` (LATEX: `\usepackage{musixtex}`) command.

If you are combining `musixser` with the lyrics macro package `musixlyr`, the `\input musixser` statement must precede the `\input musixlyr` statement!

3.1 Voices as "threads" of music events

A *voice* in the sense of `musixser` has the following properties:

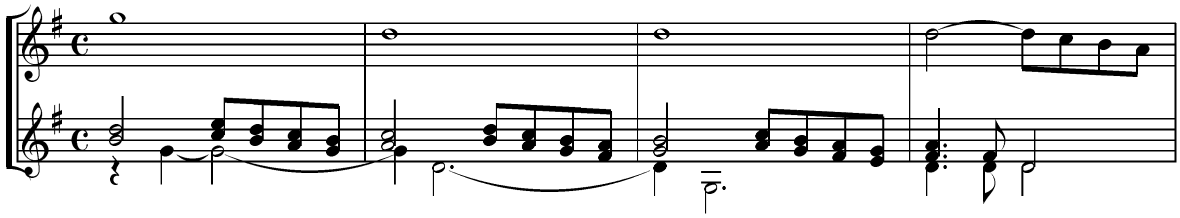
(1) It is restricted to be notated within only one staff, i.e. it must not jump between staves.

(2) It may contain chords.

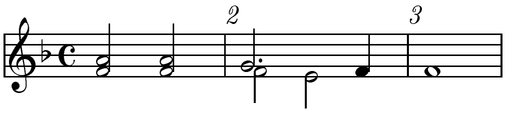
(3) It is inherently homophonic, i.e. it consists of one unique rhythmic sequence; no multiple notes with different starting points and/or durations are allowed to occur simultaneously.

(4) It does not need to be filled completely with notes or rests, but may be partially empty.

According to these rules, the following example (taken from a well-known french Christmas song) consists of three voices:



Property (3) stated above implies that, in bar 2 of the following example, two voices must be used:



However, according to property (4) mentioned above, the second voice does not need to be filled with anything like "invisible rests" in the other bars.

Note that the number of voices does not depend on the number of staves: Any number of voices (including none) can be set on each staff.

3.2 Setting up a voice

A voice is set up with the command:

`\setvoice`⟨staff no⟩⟨voice no⟩`{`⟨material⟩`}`

⟨staff no⟩ is the number of the staff (or, in MusiXTEX terminology, the instrument) the voice will be typeset in. ⟨voice no⟩ is the number of the voice within the staff; all voices of a staff must be enumerated beginning from 1. (The order of voice definitions, however, does not matter, but after having defined all voices, the sequence of voice numbers must be unbroken for each staff.) If ⟨staff no⟩ or ⟨voice no⟩ consists of more than one digit, you have to enclose it in braces (`{`⟨voice no⟩`}`).

⟨material⟩ consists of a sequence of "musical events", each of them of the following syntax:

`+`⟨duration⟩␣⟨contents⟩

The plus sign must be entered literally; it is simply a separator which marks the beginning of an event. ⟨duration⟩ is explained below; it must be followed by a literal space character or, alternatively, by a line break. ⟨contents⟩ is exactly what you would type within a `\notes`...`\enotes` construct of plain MusiXTEX—but with an important restiction: You may not specify multiple spacing notes or pauses or whatever; i.e. it is impossible to give a whole sequence of spacing notes (e.g. `\qu{cdefg}`) at once.

The ⟨duration⟩ element determines the time consumption of the event, i.e. the time elapsing until the voice's next event will happen. It may have one of several formats:

(1) ⟨numer⟩`/`⟨denom⟩

This format represents a fraction of a whole note, as usual; for example, `3/4` means the duration of a dotted half note.

(2) ⟨denom⟩

As an abbreviation of (1), omitting the numerator means setting it to one; i.e. `4` just means the duration of a quarter note.

(3) ⟨bars⟩`b`

This means that the following event shall occur after ⟨bars⟩ completed bars. In combination with the full bar rests feature (see below), this is a convenient way to express long "tacet"s of a voice. The ⟨bars⟩ amount, however, is not interpreted as a multiple of the bar duration (which would imply that if the current event occurs, say, on the 2nd beat of a bar, the following event would do also). Instead, it is the number of barlines that have to be "crossed"—independent from the current event's position within the current bar. The following event will then occur on the first beat after the specified number of bars.

(4) `b`

Corresponding to format (3), omitting ⟨bars⟩ means setting it to one, i.e. the following event will occur on the first beat of the immediately following bar.

(5) ⟨bars⟩`b`⟨numer⟩`/`⟨denom⟩

... which means: Wait for the completion of ⟨bars⟩ bars as in (3), then count the specified fraction as in (1). You can again omit ⟨bars⟩ and/or ⟨numer⟩`/`.

3.3 Processing voice material

The `\setvoice` command introduced in 3.2 only collects and buffers musical material; it does not perform the typesetting process. Instead, this is done by the command

```
\processvoices
```

which assembles all material given in previous `\setvoice` calls to a MusiXTEX input stream. However, it does not add any kind of preamble or postamble elements; everything it outputs is valid only within some `\startpiece...\endpiece` or similar command pair. Thus, a typical embedding of `\setvoice` and `\processvoices` in a MusiXTEX source file is:

⟨preamble commands, such as `\instrumentnumber`*n*⟩

`\setvoice11{`⟨material for instrument 1, voice 1⟩`}`
...
`\setvoice`*nm*`{`⟨material for instrument *n*, voice *m*⟩`}`

```
\startpiece \processvoices \Endpiece
\end
```

3.4 Full bar rests

Corresponding to MusiXTEX's `\atnextbar` feature, `musixser` provides a means to issue material which has to be processed at the next barline. Before this can be used, it has to be switched on with the

```
\barproctrue
```

command in the score preamble (just for performance reasons in case you don't need it). After that, there are several possibilities:

```
\onebarproc{...}
```

means that the issued material will be applied only once at the next barline (exactly as `\def\atnextbar{\znotes...\en}` would do).

```
\barproc{...}
```

establishes the processing of the specified material at all following barlines until explicitly cancelled by a new `\barproc` definition, e.g. `\barproc{}`.
Now there are abbreviations for the most common purposes:

`\onepause`	just one full bar rest
`\pauseon`	turn on full bar rests for all following bars
`\pauseoff`	switch off full bar rests again

As an example, here's how to issue a single raised full bar rest:

```
\onebarproc{\centerbar{\raise2\internote\pause}}
```

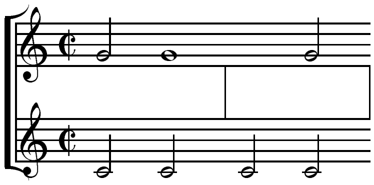
3.5 Specifying bars

Bars are set automatically. You only have to specify the bar duration as a fraction of a whole note (see above) with the `\barduration` command; for example

```
\barduration{3/4}
```

does exactly what you would expect.

Voice events (notes or rests) need not fit in a single bar with their whole duration; instead, they are allowed to extend across bars. This may seem to be a missed chance for a rhythm consistency check, but in enables typesetting notes tied across bars in a baroque flavour:



This example has been set up using:

```
\barduration{2/2}
\setvoice21{+2 \ha g+1 \wh g+2 \ha g}
\setvoice11{+2 \ha c+2 \ha c+2 \ha c+2 \ha c}
```

You can perform meter changes by re-issuing the command within the master track; see 3.7. For barless music, just omit any `\barduration` command. To switch back to barless music after having chosen some bar duration, set the bar duration to zero, e.g. with `\barduration{0/1}`.

You can influence what happens when a new bar is about to be set with the `\barlinecmd` parameter. This is set by default to equal `\bar`, but you can specify arbitrary TEX code instead. For example, if you want to force a system break after every third bar, you may specify:

```
\newcount\mybarcount
\barlinecmd={\advance\mybarcount by 1
             \ifnum \mybarcount = 3 \alaligne \mybarcount = 0
             \else \bar \fi}
```

For similar purposes, you may also evaluate MusiXTEX's `\barno` counter.

3.6 Specifying the duration-to-space mapping

`musixser` has to translate event durations into horizontal spacing. To accomplish this, it first converts the fractional representation of time values into kind of atomic "tics". Corresponding to the MIDI file format, a tic is the underlying duration unit, and all representable durations are multiples of a tic. In `musixser`, a whole note is defined to equal 192 tics. Thus, a half note equals 96 tics, a quarter note 48 tics, and so on.

As a second step, numbers of tics are mapped onto the MusiXTEX `\notes` commands, i.e. `\notes`, `\Notes`, `\NOtes`, etc. This mapping has to be established by means of the `\setticsskip` command. For example,

```
\setticsskip{48}\NOtes
```

defines that an event duration of 48 tics (i.e. a quarter note) shall correspond to a horizontal space expressed by the \NOtes command.

musixser itself predefines a small number of such relations, following the recommendation of the MusiXTeX manual concerning the use of the various \notes commands. However, if your piece contains unusual note durations, you'll have to add your own mappings. This is indicated by the TeX error message:

```
! No column skip for <number> tics defined; using default spacing.
```

If you encounter this during TeXing your source, you'll have to provide a \setticsskip call concerning the given <number> of tics within your piece's preamble.

You may also use \setticsskip to redefine the standard mapping of musixser.

3.7 The master track

All material you provide as an argument to \setvoice will be poured into some \notes...\enotes construct during processing. Everything that has to be given outside of a \notes group (e.g. \hardspace or \scale) has to be placed on a *master track* (in analogy to the standard MIDI file format) by means of the command:

```
\setmaster{⟨material⟩}
```

While the context of evaluating ⟨material⟩ differs, the *material* syntax is exactly the same as concerning the \setvoice command (see 3.2). For example: If the third bar should be typeset double as wide as the others, you may use:

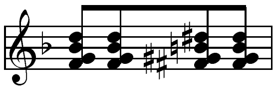
```
\setmaster{+2b +b \scale{2}+b \scale{1}}
```

Explanation: The first event is empty and has the effect of just skipping over the first two bars. The \scale{2} event has the "duration" of one bar which in this context means that the following event \scale{1} occurs at the beginning of the first following bar. The duration of the very last event of the master track never has any significance; it is safe to set it to b.

3.7.1 Synchronization of master track and voices

If a master track event and one or more voice events occur together on the same beat, the master track event becomes processed first. Layout changes issued on the master track thus are effective already for the notes occurring on the same beat.

An example for the tight relationship between a master track event and a specific note event is hard space inserted before a note in order to make room for accidentals:

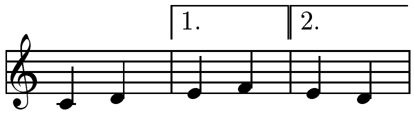


Here, the master track was set up as follows:

```
\setmaster{+2/8 \relax             % skip over first two chords
           +8 \hardspace{6.5mm}}  % add space before third chord
```

3.7.2 Synchronization of master track and bars

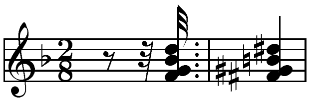
If a master track event is scheduled to occur on the first beat of a bar (especially if the preceding event has duration `b` or *n*`b`—see 3.2), the event material becomes processed *before* the barline is set. This is reasonable because MusiXTEX requires all modifications to the bar or barline (meter and signature changes, voltas, repeats, …) to be issued before the `\bar` (or similar) command. You can thus enter such bar modifications on the master track and synchronize them with the first beat of the bar to be affected. Example:



This is the master track setup:

```
\setmaster{+b \relax             % skip over first bar
           +b \setvoltabox{1}    % set volta atop second bar
           +b \setvoltabox{2}}   % set volta atop third bar
```

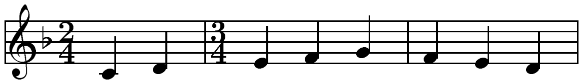
If, instead, you want a master track event (or parts of it)—occurring on a bar's first beat—to be effective *after* the bar has been set, enclose the event material in `\afterbar{...}`. Example:



Hard space has been inserted before and behind the barline between the two chords:

```
\setmaster{+b \relax                           % skip over first bar
           +8 \hardspace{1.5mm}%               % add space before barline
              \afterbar{\hardspace{5.5mm}}} % add space behind barline
```

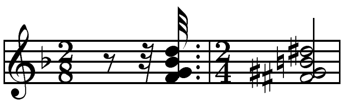
Finally, you may need to affect the barline setting process itself via a master track event; exampes include signature and meter changes where the `\bar` command must be replaced by `\changecontext` (or a related command). This can be accomplished by enclosing the material in `\thisbarcmd{...}`; material issued this way replaces the default bar command `\bar` or whichever is specified with `\barlinecmd` (see 3.5). Example:



The master track is:

```
\setmaster{+b \relax                                    % skip over first bar
           +b \thisbarcmd{\generalmeter{\meterfrac34}% set new meter
                          \barduration{3/4}%          % inform musixser about
                                                      % meter change
                          \changecontext}             % print out meter change
```

An example that combines material to be processed before, at, and behind the barline:



The master track is:

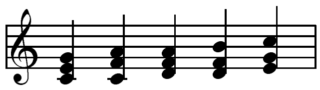
```
\setmaster{+b \relax                      % skip over first bar
           +8 \hardspace{1.5mm}%          % add space before barline
              \thisbarcmd{\barduration{2/4}% meter change at barline
                          \generalmeter{\meterfrac24}%
                          \changecontext}%
              \afterbar{\hardspace{5.5mm}}}% add space behind barline
```

3.8 Abbreviations for recurring structures

If the same structural element (e.g. an ostinato pattern) occurs frequently in one or several voices of your score, you can reflect this structure property by defining a `musixser` "pattern" as a template and then calling this template wherever it occurs in a voice. The interesting thing about this is that this level of structure abstraction affects individual voices independent from what other voices do simultaneously; this is impossible with plain MusiXTEX because elements of all voices are inseparably interwoven there.

3.8.1 Single-event structures

Structure elements based on a single event (e.g. chords) do not need special `musixser` features; they can be expressed as normal TEX macros which are then used as an event's ⟨contents⟩ (in the sense of section 3.2). Example:

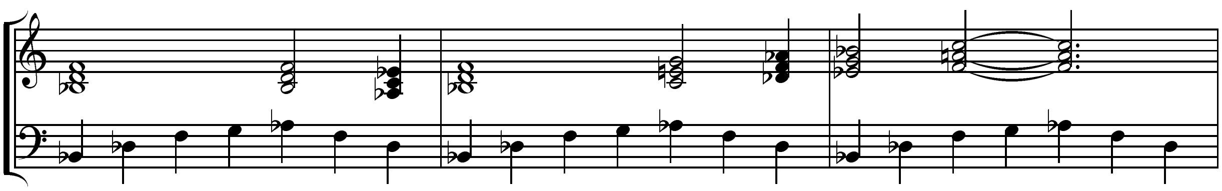


Code excerpt:

```
\def\mychord#1#2#3{\zq#1\zq#2\qu#3}
\setvoice11{+4 \mychord ceg%
            +4 \mychord cfh%
            +4 \mychord dfh%
            +4 \mychord dfi%
            +4 \mychord egj}
```

3.8.2 Multi-event structures

Consider the following example of an ostinato pattern as part of a polyphonic arrangement (taken from *Der schwarze Mond* by HARALD GENZMER):



Setting up a template for this seven-notes ostinato would require including not only the event contents but whole event specifications (including the event header `+`⟨duration⟩␣) of all notes in a macro—but this would not work. (The `\setvoice` command parses the structure of the voice material by scanning its argument for occurrences of exactly this event header, thus it relies on event headers being issued literally in the voice material and not buried inside any macros.) Therefore, `musixser` provides its own mechanism for defining such multi-event patterns. A pattern is defined as follows:

```
\defpattern\⟨pattern name⟩{⟨material⟩}
```

\⟨pattern name⟩ is a unique name for the pattern to be defined. ⟨material⟩ can be anything that is also valid as an argument to the `\setvoice` command; it can also be a material fragment, but should preferrably consist of a sequence of complete event specifications.

Within a `\setvoice` call, a previously defined pattern is referenced as follows:

```
\setvoicenm{...\pat\⟨pattern name⟩...}
```

The control sequence `\pat` must be entered literally and tells `\setvoice` to replace \⟨pattern name⟩ by the material given in the pattern's definition.

To continue the above example from *Der schwarze Mond*—here is how the ostinato pattern is defined and referenced:

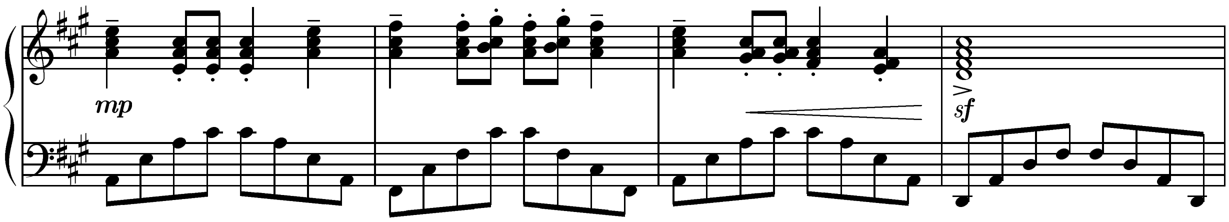
```
\defpattern\bassostinato
    {+4 \qa{_I}+4 \qa{_K}+4 \qa M+4 \qa N+4 \qa{_a}+4 \qa M+4 \qa K}
\setvoice11{\pat\bassostinato
            \pat\bassostinato
            \pat\bassostinato}
```

Patterns can be nested. It is well possible to define structures like:

```
\defpattern\a{...}
\defpattern\b{...\pat\a...}
\setvoicenm{...\pat\a...\pat\b...}
```

3.8.3 Parameterized multi-event structures

As discussed in section 3.8.2, patterns can express only musical fragments that recur always exactly equal. A much more interesting case, however, are recurring structures that follow a common scheme but do *not* exactly match each other. Consider the left-hand part of the following example (taken from *Suite bergamasque*, 4th movement "Passepied", by CLAUDE DEBUSSY):



The form element underlying all four bars can easily be recognized although the concrete note pitches differ from one bar to the other. `musixser`'s solution for this kind of structure abstraction are parameterized patterns: The common structure is defined once as a pattern, the differences between the structure "instances" are given as arguments to individual pattern calls.

The syntax of a parameterized pattern definition resembles the one of a TEX macro definition with arguments:

```
\defpattern{\⟨pattern name⟩⟨parameter text⟩}{⟨material⟩}
```

⟨parameter text⟩ is meant here exactly as in *The TEXbook*, i.e. a formal description of the arguments that the pattern expects to find when it is called; this is usually an enumeration of the number of parameters—i.e. `#1#2...`—but can as well be arbitrarily sophisticated (as described in *The TEXbook*). ⟨material⟩ is again the replacement text for \⟨pattern name⟩, but in this case it is allowed to contain references to the parameters mentioned in ⟨parameter text⟩, i.e. `#1`, `#2`, and so on.

The syntax for a parameterized pattern call is:

`\setvoice`*nm*`{...\pat{\`⟨pattern name⟩⟨arguments⟩`}...}`

Note that the \⟨pattern name⟩⟨arguments⟩ construct must be enclosed in braces. ⟨arguments⟩ must of course match the ⟨parameter text⟩ of the \⟨pattern name⟩ definition.

Back to the above example: The structure common to all four bars of the left-hand part is two groups of four beamed eighth notes each, the notes pitches mirroring each other, and fixed beam slopes of 1 or −1, respectively. The variable elements are the four different notes pitches, the starting pitches of the two beams, and the beam position (upper or lower). This leads to the following definition of a one-bar pattern:

```
\defpattern{\lefthand #1#2#3#4#5#6#7}
  % Arguments:
  % #1:   beam position ('u' or 'l')
  % #2:   1st beam's start pitch
  % #3:   2nd beam's start pitch
  % #4-7: note pitches
  {+8 \if #1u\ibu0#22\else\ibl0#22\fi \qb0{#4}%       1st beam group
   +8 \qb0{#5}%
   +8 \qb0{#6}%
   +8 \if #1u\tbu0\else\tbl0\fi \qb0{#7}%
   +8 \if #1u\ibu0#3{-2}\else\ibl0#3{-2}\fi \qb0{#7}% 2nd beam group
   +8 \qb0{#6}%
   +8 \qb0{#5}%
   +8 \if #1u\tbu0\else\tbl0\fi \qb0{#4}%
  }
```

After this, the setup of the left-hand voice merely consists of four pattern calls:

```
\setvoice11{\pat{\lefthand lJLHLac}%
            \pat{\lefthand lHJFJMc}%
            \pat{\lefthand lJLHLac}%
            \pat{\lefthand uJLDHKM}}
```

3.9 Automatic slur and beam number allocation

Patterns—as introduced in section 3.8—are not restricted to being used in one specific voice. Instead, their definitions are "global" and can be referenced from any voice and even simultaneously, e.g. in order to set up unisono phrases from a single melody source. Consider the following example (taken from *Carmina burana* by CARL ORFF):

In this context, the slurs cannot be specified in the "normal" way. If you would say something like

```
\defpattern\unisono{...\isluru0h...}   % This
\setvoice11{\pat\unisono}              % won't
\setvoice21{\pat\unisono}              % work!
```

`musixser` would include the initiation of slur number 0 in both voices simultaneously which would result in a slur number clash; the same applies to beam numbers.

`musixser` helps to cope with this situation by providing "per voice" slur and beam number ranges, i.e. each slur/beam number reference is relative to the voice that it has been issued in; references to the same number issued in different voices do not interfere with each other. Behind the scenes, `musixser` maps those relative slur/beam numbers onto MusiXTEX's absolute numbers by automatically allocating as many numbers as are needed at each point of time.

The syntax for slur/beam commands using automatic numbers is a bit complicated to explain. Let's start considering the general structure of a MusiXTEX slur command:

⟨slur command⟩⟨slur number⟩⟨further arguments⟩

An example:

```
\isluru0h
```

... where ⟨slur command⟩ = `\isluru`, ⟨slur number⟩ = 0, and ⟨further arguments⟩ = `h`.

Using automatic slur numbers, this has to be replaced by (general structure):

⟨slur header⟩⟨slur command⟩⟨further arguments⟩

Example:

```
\isl\isluru h
```

... where ⟨slur header⟩ = `\isl`, ⟨slur command⟩ = `\isluru`, and ⟨further arguments⟩ = `h`. The underlying idea is to replace the explicit absolute slur number by the ⟨slur header⟩ which instructs `musixser` to maintain a voice-related reference to a slur number.

There are basically three different slur headers:

`\isl` allocates a free slur number; to be used in conjunction with MusiXTEX slur initiation commands. Example: `\isl\isluru c`

`\rsl` references an allocated slur number. Example: `\rsl\invertslur`

`\tsl` frees an allocated slur number; to be used in conjunction with MusiXTEX slur termination commands. Example: `\tsl\tslur c`

These slur headers totally omit mentioning any slur number; they are convenient and appropriate if you have at most one slur per voice at the same time. If you have more than one slur per voice simultaneously, you must be able to distinguish between them when issuing slur commands; in this case, replace the ⟨slur header⟩ `\isl` by `\Isl`⟨number⟩ where ⟨number⟩ is a voice-related "handle" to a specific slur. Also replace `\rsl` by `\Rsl`⟨number⟩ and `\tsl` by `\Tsl`⟨number⟩. Of course each `\Isl`⟨number⟩ must be followed by a matching `\Tsl`⟨number⟩, and each `\Rsl`⟨number⟩ must be preceded by a matching `\Isl`⟨number⟩. (`\isl`, `\rsl`, and `\tsl` actually are abbreviations for `\Isl0`, `\Rsl0`, and `\Tsl0`, respectively.)

As a comprehensive example, here is a code excerpt from the above *Carmina burana* phrase:

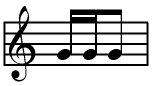
```
\defpattern\unisono
  {% bar 1
   +4 \Isl1\ibsluru h%         % initiate "long" upper slur with
                               % explicit relative number 1
      \isl\issluru h%          % initiate 1st "short" slur with
                               % implicit relative number 0
      \qu h%                   % set first note
   +4 \tsl\tslur i\ql i%       % terminate 1st "short" slur; set 2nd note
   +4 \ql j+2 \hl i+4 \qu h%
   +4 \isl\isluru i\ql i%
   +4 \tsl\tslur h\qu h%
   +4 \qu g+3/4 \hup h%
```

```
    % bar 2
    +4 \isl\issluru h\qu h%
    +4 \tsl\tslur i\ql i%
    +4 \ql j+4 \ql i+4 \qu h%
    +4 \isl\isluru i\Isl2\itieu i\ql i%
    +4 \Tsl2\ttie\ql i%
    +4 \tsl\tslur h\qu h%
    +4 \isl\islurd g\qu g%       % initiate last "short" slur
    +3/4 \Tsl1\tbsluru h%        % terminate last "short" slur
         \tsl\tslur h%           % terminate "long" upper slur
         \hup h}                 % set last note
\setvoice11{\pat\unisono}
\setvoice21{\pat\unisono}
```

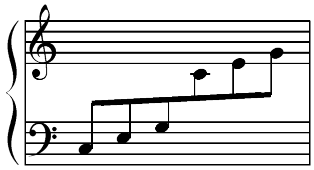
Everything said so far also applies to beam numbers. For these, `musixser` supplies the beam headers `\ibm`, `\rbm`, `\tbm`, `\Ibm`⟨number⟩, `\Rbm`⟨number⟩, and `\Tbm`⟨number⟩. The `\rbm` and `\Rbm`⟨number⟩ beam headers are mainly used to attach note heads to beams and for beam multiplicity changes. Example:

```
\setvoice11
 {+16 \ibm\ibbu g0\rbm\qb g%
  +16 \rbm\tbbu\rbm\qb g%
  +8 \tbm\tqh g}
```

Which yields:



`musixser`'s slur/beam number management encapsulates each slur/beam in the voice that it belongs to. However, beams spanning over multiple voices are sometimes needed. Example:



To accomplish this, a further set of beam headers is offered: `\genibm`⟨number⟩, `\genrbm`⟨number⟩, and `\gentbm`⟨number⟩ (meaning "generic ..."). A beam ⟨number⟩ issued this way can be referenced from within any voice, similar to MusiXTEX's absolute beam numbers.
Here is a code excerpt from the above example:

```
\setvoice11{+8 \genibm0\ibu L1%
               \genrbm0\qb J%
            +8 \genrbm0\qb L%
            +8 \genrbm0\qb N}
\setvoice21{+3/8
            +8 \genrbm0\qb c%
            +8 \genrbm0\qb e%
            +8 \gentbm0\tqb g}
```

(For the sake of completeness, `musixser` also provides equivalent slur headers `\genisl`⟨number⟩, `\genrsl`⟨number⟩, and `\gentsl`⟨number⟩—but you will not need to use them since cross-voice slurs are accomplished by MusiXTEX in a different way.)

Note that you cannot simply replace, say, `\genibm0\ibu L1` by `\ibu0L1`. The latter will allocate absolute beam number 0 whereas the former instructs `musixser` to allocate a free beam number and to map relative number 0 onto it. The latter will clash with `musixser`'s automatic beam number management, the former will cooperate.

Even if you do not use patterns, automatic slur/beam numbers are a convenient means to avoid keeping track of already used numbers when typesetting multi-voice scores.

3.10 Typesetting long scores

`musixser` has to keep in memory all voices' material simultaneously; therefore, it's impossible to set up an arbitrarily long score within a single `\setvoice` per voice without exceeding TEX's memory. The solution is to split the score into multiple "blocks" of material which are processed by TEX sequentially. The general pattern for this is:

```
\startpiece
  \setvoice11{⟨material for block 1, instrument 1, voice 1⟩}%
  ...
  \setvoicenm{⟨material for block 1, instrument n, voice m⟩}%
  \setmaster{...}%
  \processvoices
  ...
  ⟨arbitrarily many blocks⟩
  ...
  \setvoice11{⟨material for block x, instrument 1, voice 1⟩}%
  ...
  \setvoicenm{⟨material for block x, instrument n, voice m⟩}%
  \setmaster{...}%
  \processvoices
\endpiece
```

For choosing appropriate "breakpoints" for this division into blocks, it is reasonable to follow rather "natural" divisions like movements, verses, instrument number changes, etc. On the other hand, splitting material into blocks has no influence on the resulting music layout (e.g. line breaks).

`musixser` keeps track of the bar progress even across subsequent `\processvoices` calls, i.e. if one block ends with an incomplete bar, the following one will start with another incomplete one forming just the "complement" of the ending bar before. This makes it possible to split a piece into two blocks even in the middle of a bar (which may be reasonable when using pickup bars). However, if you want to force the second block to begin with a complete bar, issue the command

```
\resetbarticscount
```

somewhere before the next `\processvoices` call.

3.11 Typesetting plus signs

In order to utilize the plus sign as an event separator, `musixser` makes it—in terms of TEX—an active character. This inhibits typesetting a literal plus sign just by entering a plus sign in your TEX source. Instead, you must enclose the occurrence of the plus sign in the command pair:

```
\makeplusother...\makeplusactive
```

Due to TEX's macro expansion rules, however, this construct does not work if it is issued within a macro argument. If you want a plus sign as part of a macro argument (e.g. `\setvoltabox{1.+3}`), you have to bury it in another macro. Example:

```
\makeplusother \def\plussign{+} \makeplusactive
...
\setmaster{...\setvoltabox{1.\plussign 3}...}
```

4 Score arrangement and part extraction

`musixser` has no dedicated part extraction feature, but the explicit assignment of voice material to staff numbers makes it easy to rearrange the voices-to-staves distribution in a more general way. Simply permuting the ⟨staff no⟩ entries of the `\setvoice`⟨staff no⟩… constructs straightforwardly redestributes voices to staves.

Since `musixser` processes only those voices with a ⟨staff no⟩ that is valid with regard to the current `\instrumentnumber`, voices can easily be hidden by giving them a ⟨staff no⟩ greater than the current `\instrumentnumber`. (Do not set ⟨staff no⟩ to zero—this would lead to problems.) In this respect, extracting an instrument part from a score is accomplished by setting `\instrumentnumber` and the ⟨staff no⟩ of the voice(s) to be extracted to one and giving all other voices a greater ⟨staff no⟩. (Okay, this method admittedly does not convert multiple subsequent full bar rests into a single multi-bar rest, but appropriate use of the pattern expansion feature—see 3.8—should make it easy to switch between the two rest layout styles manually.)

You can enhance dealing with the voices-to-staves distribution by defining macros for staff numbers so that maintenance of staff numbers happens at one single place of your source (i.e. in the macro definition). This works because most MusiXTEX commands dealing with staff properties accept a macro instead of a literal instrument number.

Consider as an example a score of three instruments. Begin with the following definitions:

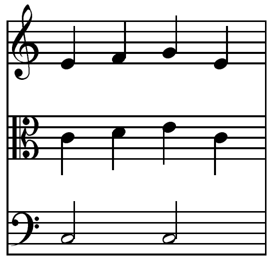
```
\def\violin{3} \def\viola{2} \def\basso{1}
```

Then you can continue setting up the score referring to these symbolic staff number names:

```
\instrumentnumber3
\setclef\basso\bass
\setclef\viola\alto

\setvoice\violin1{+4 \qa e+4 \qa f+4 \qa g+4 \qa e}
\setvoice\viola 1{+4 \qa c+4 \qa d+4 \qa e+4 \qa c}
\setvoice\basso 1{+2 \ha J+2 \ha J}
```

Which yields something like:

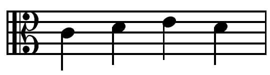


To extract the viola part, simply change the following two lines of the score header to:

```
\def\violin{4} \def\viola{1} \def\basso{4}

\instrumentnumber1
```

Which yields:



5 Error handling

5.1 Finding error sources

The whole processing of the material given within `\setvoice` commands takes place within the `\processvoices` macro which has to be called within a `\startpiece...\endpiece` pair. Therefore, if a syntax error occurs within some ⟨material⟩, the line number given by TEX along with the error message won't point to the true location of the error; instead, it will always refer to the line of the `\processvoices` call. In order to locate the exact error source in your source file, you can utilize two kinds of information:

(1) The bar number trace that MusiXTEX outputs on the console while running; the last bar number printed before the error message refers to the last completed bar before TEX has run into the error. In order to be able to utilize this information, it's good style to include bar numbers as comments in your ⟨material⟩.

(2) `musixser`'s voice trace. If you issue the command `\tracingvoicestrue` somewhere in your source file preamble, `musixser`—before processing any ⟨material⟩—prints out the staff number and number of the voice that this ⟨material⟩ belongs to. Thus the numbers of staff and voice printed immediately befor the error message point to the voice containing the defective code.

Some typical error messages and interpretations:

```
! Missing number, treated as zero.
```

Possibly an error within the ⟨duration⟩ syntax.

```
Runaway argument? ...
! Forbidden control sequence found while scanning use of +.
```

Most probably a forgotten space character between ⟨duration⟩ and ⟨contents⟩.

```
! No column skip for ... tics defined; using default spacing.
```

You have to provide a duration-to-space mapping as explained in 3.6.

5.2 Tracing options

If TEXing your source ends without error but you feel that something's gone wrong with the synchronization of the voices, you may try to issue the command

```
\tracingticstrue
```

somewhere in your piece's preamble. MusiXTEX will then print below each "column" (i.e. each beat where something musical happens) the number of tics it has determined as the time value of this column. This provides some help for tracing the synchronization decisions `musixser` has made when assembling the score from the distinct voices.
Other tracing options—mainly useful if you get a TEX error message—include:

`\tracingvoicestrue` logs for each ⟨material⟩ currently being processed to which voice it belongs; see 5.1.

`\tracingslurnumberstrue` logs slur number allocations and deallocations; helps finding `\isl` and `\tsl` mismatches (see 3.9).

`\tracingbeamnumberstrue` similar to `\tracingslurnumberstrue`.

`\tracingcolumnstrue` prints out details about `musixser`'s analysis of event durations and score assembly; rather useful for `musixser` debugging than for source error detection.

`\tracingpatternstrue` prints out details about pattern expansion (see 3.8); also mainly for `musixser` debugging.

All mentioned tracing functions can be activated anywhere in your source; if the activation shall take place somewhere within a `\processvoices` run, the respective command must be given within the master track. Tracing may be switched off by issuing `\tracing...false` instead of `\tracing...true`.

6 Software license

Author: Rainer Dunker
Lilienstr. 18
85591 Vaterstetten
Germany

E-mail: `rainer.dunker@ebe-online.de`