PMX – a Preprocessor for MusiXTEX

Version 2.84 – 20 December 2017

Don Simons
Dr. Don's PC and Harpsichord Emporium
Redondo Beach, California, USA.
dsimons@roadrunner.com

Preface

Compared to version 2.80, aside from bug fixes which are documented in the opening comments in the source code, **PMX** version 2.84 allows arbitrary numbers of rests in xtuplets, chordal notes in 2-note tremolos, and user-defined adjustments of beam height and slope in 2-note tremolos.

Contents

Dedication

The MusiXTEX community was stunned by the sudden death of Werner Icking on February 8, 2001. He had been a benevolent patriarch, touching many of us not only with his technical savvy and gentle guidance, but also his genuine kindness and generosity. His spirit runs deep through all of **PMX** . His encouragement fueled its development from its very beginning. Many enhancements have been his proposals, including one he made on what turned out to be his last day. Werner, my friend, I dedicate this work to you and your memory.

1 Introduction

PMX is a preprocessor for MusiXTEX. Before using it you should have installed MusiXTEX Version 1.21 or higher, and any available version of TEX that includes e-TEX. The goal of **PMX** is to facilitate the efficient typesetting of scores and parts that have an almost professional appearance. It can do *all* the work involved in setting up `\notes-\enotes` groupings, selecting groups of notes to be beamed, defining beam heights and slopes, spreading the entire piece evenly over specified numbers of systems and pages, and inserting extra spaces where needed to make room for accidentals, flags, dots, and new clefs. The input language for **PMX** is much simpler than MusiXTEX. You can enter note values and rests from 64ths to double whole notes (*breves*), ornaments, slurs, and limited text strings. Every voice in every bar must have exactly the correct number of beats in the current meter, but you may change the meter at the beginning of any measure, with or without printing the new time signature. Before making a TEX file, **PMX** checks these timings and other aspects of the input. **PMX** has special features for dealing with baroque chamber music, including the ability to notate figured bass below the bottom staff in each system. If **PMX** hasn't yet learned to do something you want to do, you can usually work around the problem by inserting literal TEX strings in the **PMX** input file. 2.71

You can automatically create parts from a score using **scor2prt**. This auxiliary program generates a set of `.pmx` input files, one for each part, from a single `.pmx` file for the score. You can control the appearance of the parts with special commands in the main file, thereby making it possible to include within a single input file all the information that defines the score and the individual parts.

The basic **PMX** distribution as of this version of the manual is `pmx276.zip`. It contains the FORTRAN sources, binaries that will run in a DOS window on a PC with WINDOWS95 or higher, manuals for useage and for installation in DOS/Windows, and example typesetting files. Alternatively, the software section of the Werner Icking Music Archive (WIMA) has instructions for acquiring and installing MusiXTEX and **PMX** on various platforms (Windows, Mac-OSX, Unix-like systems) including using automated procedures from several external TEX distribution sites. The packages from those other sites will all eventually incorporate all the upgrades in `pmx276.zip` but will take varying amounts of time to do so. **PMX** is often upgraded; the most current version will always be available directly from the "News" paragraph here. 2.76

1.1 Conventions for This Manual

Hey, this is boring stuff, but if you take a minute to understand the typographic conventions and a little jargon, it may avoid some confusion down the road.

The typewriter typeface always indicates verbatim text as it would be input to a computer. This includes file names, MusiXTEX tokens, and **PMX** commands, e.g., `barsant.pmx`, `\internote, c44`.

Bold is used for program names (e.g., **pmxab**), or when applied to a single letter, to relate a **PMX** command to its meaning (e.g., "`e` signifies a **l**eft shift").

When viewing the PDF version of this document on a computer screen, clickable internal hyperlinks are colored blue, and clickable external links are underlined and colored cyan.

Italics may mean several different things depending on the context: simple emphasis, or the first appearance of *jargon* (buzz-words that need to be explicitly defined), or finally to represent input variables for which some verbatim text would need to be substituted. In the latter case the variable will be surrounded by square brackets, e.g., [*basename*], but the brackets are not to be included with the substituted text.

Speaking of jargon, there are several special words that have very specific meanings here: A *staff* is one set of 5 lines (plural *staves*), a *system* is a group of staves, and *voice* refers to one of

the one or two simultaneous allowable sequences of notes in a staff. Note that this is a change from versions prior to 2.5, where *voice* was used interchangeably with staff.

A **PMX** *command* is a string of characters with no spaces between them. The first character determines the type of command. Any other characters are parameters that may be either required or optional. Sometimes we loosely use the word *command* to refer just to the initial character.

1.2 Setup

Here we briefly describe the setup for the Windows OS, assuming TEX and MusiXTEX have already been installed. After compiling the FORTRAN source code, users of other OS's may either adapt these instructions as needed or use one of the other setup methods referenced earlier.

After decompressing the distribution file `pmx276.zip`, you should have these files: `pmx276.for`, `scor2prt.for`, two Windows executables `pmxab.exe` and `scor2prt.exe`, several sample `.pmx` files, `pmx.tex`, `ref276.tex` (TEX source for a command summary), `pmx276.tex` (TEX source for this file), PDF images of the latter two files, `pmx25-276.html` showing changes from version 2.50 to 2.76, and `install_run_PMX271.pdf` which gives more details about installing and running on different platforms. 2.76

Once you have assembled a full set of files, put the executables somewhere in the path or in your working directory, `pmx.tex` into the texinput directory, the sample `*.pmx` files in your working directory (the one from which you will run **PMX**), and the source code and document files wherevever you wish.

1.3 Basic Operation, by Example

Edit the 15th line of `barsant.pmx` to contain the path to the directory where you want **PMX** to write the `.tex` file. For example, if you want this to be the same as the working directory, type `.\` for Windows, or `./` for UNIX.

If you haven't done so, open a command window and navigate to the folder containing `barsant.pmx`. Execute **PMX** by typing `pmxab barsant` . Alternatively, you may just type `pmxab <return>` and you will be prompted for a jobname, which in this case is just `barsant` . **pmxab** will always generate two files in the working directory: `barsant.pml` is a log file, and `pmxaerr.dat` contains a single integer, 0 if the run was successful, otherwise the line number in the `.pmx` file of the fatal error (useful for batch processing). Also, on successful completion, `barsant.tex` will be placed in the path specified in the setup.

Now you are right where you would be after entering, debugging, and rough-editing the `.tex` file manually. To see the results, process `barsant.tex` just as you would for any MusiXTEX file, running all three passes, and view the `.dvi` file, or go on and run **dvips** to create a postscript file and view that with a postscript viewer such as **GSview**. To make separate parts, run **scor2prt** by typing `scor2prt barsant` . The program will create a new `.pmx` file for each instrument, in this case `barsant1.pmx` and `barsant2.pmx`. You may then process these files like you did the original one to create separate parts.

2 Elements of PMX

2.1 Setup Data in the Input File

To see how the input file is put together, we'll look at `barsant.pmx`. For reference, here are are the first few lines:

```
%----------------%
```

```
%
%  barsant.pmx   Revised 29 June 2002
%
%----------------%
%
% nv,noinst,mtrnuml,mtrdenl,mtrnmp,mtrdnp,xmtrnum0,isig,
  2    2      4       4        0      6      0       0
%
% npages,nsyst,musicsize,fracindent
    1      7     20        0.07
Basso
Recorder
bt
./
```

The lines with % in column 1 are comments. Some special handling of comment lines will be discussed in the section on creating parts from a score in section 3.

The rest of the lines in this example are the *setup data*. Starting in the first non-comment line above,

`nv` (integer≤24) is the total number of staves per system. Each staff may contain either one or two voices, but the total number of voices at any one time may not exceed 24. 2.6

`noinst` (integer≤`nv`) is number of *instruments*. Each instrument has a unique name (see below), and any instrument with more than one staff will have its staves joined with a curly bracket. Usually there is only one staff per instrument and `noinst=nv`. There are two ways to assign more than one staff to one or more instruments. If only the first (lowest) instrument has more than one staff, such as in a score for piano and a solo instrument, simply make `noinst<nv` and any difference will show up in instrument 1, the bottom one in each system. For a more general distribution of staves among instruments, put a minus sign in front of `noinst`, and follow `noinst` with the number of staves in each instrument in succession, separated by spaces. These numbers must add up to `nv` or your computer will explode. For a typical example of keyboard music, see `mwalmnd.pmx`, in which `nv=2` and `noinst=1`, producing two staves per system with a curly bracket at the left.

The number of instruments can be changed as well after the start of the score, but only to a number less than the original one. See section 2.3.12 to learn how to start with a smaller number of instruments and later increase it.

`mtrnuml` is the *logical* numerator of the meter, or the number of beats per measure; `mtrdenl` the denominator. Please note the special considerations in the paragraph after the next. If `mtrnuml` is divisible by 2 or 3, beam grouping will be automatic; otherwise you will have to force all beams using `[...]` as described in section 2.2.12.

`mtrnmp` and `mtrdnp` are the *printed* numerator and denominator. These determine the appearance of the meter in the printed output but have no effect on the internal timing analysis. If `mtrnmp>0` then it and `mtrdnp` are printed literally as the numerator and denominator of the time signature. Please note the special considerations in the following paragraph. If `mtrnmp<0`, then the numerator is abs(`mtrnmp`) and the entire time signature will be printed with a vertical slash through it. If `mtrnmp=0`, then `mtrdnp` determines the printed meter as follows:

`0`	No meter is printed (*blind* meter change)
`1, 2, 3, or 4`	A single digit, between the 2nd and 4th lines
`5`	Cut time (alla breve)
`6`	Common time
`7`	Numeral 3 with a vertical slash

There are special considerations for n/16 and n/1 time signatures (where the latter "1" normally means a whole note). To get n/1 time, use `0` (zero) for `mtrdenl` and `1` for `mtrdnp`. To remember this rule, recall that the printed denominator is taken literally, while the logical denominator can always be represented by the same single digit used for the corresponding time value when entering ordinary notes (see section 2.2.1). So for n/16 time, use `1` for `mtrdenl` and `16` for `mtrdnp`.

If the first bar is a partial bar containing a pickup, `xmtrnum0` is the number of beats in it; otherwise set it to 0. It need not be an integer. The first bar is the *only* bar that can have a different number of beats than the current value of `mtrnuml` (Later we'll see how to change the meter).

`isig` is the key signature, positive integer for sharps, negative for flats.

If `npages`>0, it is the number of pages and `nsyst` is the total number of systems in the entire piece. **PMX** will spread the entire piece horizontally over this number of systems, and vertically over `npages` pages. For proper vertical spacing there should be from about 9 to 16 staves per page. If you specify too many staves for the number of pages, one or more staves may spill over onto an extra sheet. If this happens it will only become obvious when you preview the `.dvi` file. One solution is to use the global option `Ae` (see section 2.3.8); another is to increase `npages` or decrease `nsyst`.

If `npages` is set to 0, then `nsyst` is interpreted as the average number of measures per system. This is useful while building up a file a little at a time. **PMX** will calculate how many systems to use, and spread them over an appropriate number of pages.

`musicsize` is 16, 20, 24, or 29, the height of a staff in points, with 20 considered the default. 2.6

`fracindent` is the indentation of the first system from the left margin, expressed as a decimal fraction of the total line width.

Next come the names of the `noinst` instruments as you want them to appear within the indentation in the first system, one per line, starting with the *bottom* instrument. If you've set `fracindent`=0 and don't want instrument names to appear, you must still leave `noinst` blank lines here. Next comes a single string of `nv` letters or numbers for the clefs, again starting with the bottom staff: `b, r, n, a, m, s, t, f, 8` or digits 0-8 respectively for **b**ass, bar**i**tone, 2.71
te**n**or, **a**lto, **m**ezzo-soprano, **s**oprano, **t**reble, **F**rench violin clef, or octave treble clef. The last line of setup data contains the path to the directory where you want the `tex` file to go when **PMX** creates it. The one in `barsant.pmx` , `./` , represents the current directory in UNIX and some versions of DOS. The path must terminate with `/` or `\` .

2.2 Structure of the Body of the Input File

The rest of the `.pmx` file is the *body* of the input. The basic unit of input from here on is called an *input block* or just *block*, each one representing an integral number of bars. If there is a pickup bar defined by `xmtrnum0` > 0, it must be included in the first block *together with at least one full bar*. If you wish to put a pickup in a separate block, for example at the start of a new movement, set the initial logical meter to fit the pickup bar, then after the pickup bar do a blind meter change as described in section 2.3.3).

There will usually be 4 to 8 bars in a block. 15 is the most allowed. It is good practice to separate the blocks with comment lines that state which bars are represented, as I've done in `barsant.pmx`. It is also advisable, although not required, to separate the bars with the command `|`. Its main functions are to provide visual separation in the input file, and to help isolate input errors: if you put one anywhere except at a bar-end, **pmxab** will stop and show you where it detected the timing error. Otherwise, with several minor exceptions, `|` has no effect.

At the start of each block there may be a few special commands (described starting in section 2.3). Next come the input data for the selected number of bars of the first (lowest in the system) voice in the first staff, followed by either `/` to move to the next staff, or `//` to move to the

next voice on the same staff. Each new voice must start on a new line in the input file, i.e., there should be no further data on the same input line after `/` or `//` . Continue entering other voices, each with *exactly* the same number of bars as the first, terminated by `/` or `//`, until the last (topmost in the system) ends with a `/` and the block is finished. Within a block every voice must have the same number of bars, but every block needn't have the same number of bars as other blocks. The number of voices in a staff can only be 1 or 2, and cannot change within a block, but may vary from block to block.

The data for each voice in each staff are a sequence of commands containing one or more adjacent characters. Commands are separated from each other by spaces. The line-terminating commands `/` and `//` should also naturally be preceded by a space.

2.2.1 Notes

Commands for notes always start with a lower-case letter and, as with all commands, end at the first space. The first letter is the note name (`a-g`). The rest of the characters can be in any order with only a few restrictions. The first digit defines the *basic time value* of the note: `9, 0, 2, 4, 8, 1, 3` or `6` respectively for double-whole, whole, half, quarter, eighth, sixteenth, thirty-second, and sixty-fourth notes. The second digit sets the octave (for reference, octave 4 runs from middle C to the B above). Certain letters may appear after the initial one: `d` for **d**ot; `dd` for double dot; `f, n,` or `s` for **f**lat, **n**atural, or **s**harp (repeat the letter immediately for a double); `u` or `l`, which force the stem direction of any un-beamed note; `e` or `r` to shift the notehead l**e**ft or **r**ight by its own width; `a` (for **a**lone) which inhibits beaming for this note (or, if the first note of an xtuplet, for the entire xtuplet); and `T` to insert a *tremolo* on the stem. The `T` may be followed by a single digit `1`, `2`, or `3` to indicate the number of slashes in the tremolo symbol; `1` is the default if no digit is entered. A single accidental may be immediately followed by `c` to make it **c**autionary, i.e., surround it with parentheses. Alternatively, it may also be followed by `i` to suppress typesetting but still have the M**I**DI processor honor the accidental. Other characters allowed in note commands are `+`, `-`, `.`(period), `,`(comma), `x`, and several special characters following `x`, all to be described below. Between the first letter and the end or `x` if present, non-digits can be in any order with respect to each other and to the digits, with minor exceptions involving shifting dots and accidentals. 2.80

To move a dot from its default location, simply follow the `d` with one or two decimal numbers, each predeced by `+` or `-`. The first is the vertical shift in `\internote`s, the second, the horizontal shift in notehead widths.

Accidentals can be shifted too. One way is to enter `+` or `-` right after the accidental character, then an integer for the vertical shift, then another `+` or `-` followed by the horizontal shift in notehead widths. If you use this method, you *must* enter both numbers. Or, to just shift horizontally, use `<` or `>` followed by the shift in notehead widths. When shifting a sharp to avoid another sharp, a left shift of 0.85 is usually best. When shifting a flat to avoid a flat above it, a left shift of 0.3 is suggested. In chords (see section 2.2.4), if all the notes are in the same voice, **PMX** will automatically shift accidentals if required. This will be disabled for the current chord if any user-defined accidental shifts are entered, unless `A` is entered along with the shift, e.g., `zcsA<.5` . In that case the user-defined shift will be added to the PMX-computed one. Another option that affects accidental positioning in chords is `Ao`, entered in the main note command of a chord. It will force the accidentals in that chord will be posted in the order they come in the source file (starting with the main note), each one as far to the right as it will go without crashing into a notehead, stem, or another accidental.

Dots and accidentals always have to be entered when and if a note calls for them. i.e., they are never carried over from previous notes. On the other hand, the octave only needs to be entered if the note is more than a fourth away from the most recent note in the same voice. This feature lets you go for long stretches in a voice before needing to enter the octave. An alternate

way to jump more than a fourth but less than a twelfth is to type `+` or `-`. In other words, these mean to put the note an octave higher or lower than it otherwise would have gone. Two `+`'s will raise the pitch two octaves above what it otherwise would have been, and so forth. The basic time value is also carried over from the past if it is not re-entered, except for the first note or rest in each voice in an input block, for which it *must* be entered. Therefore, when the melody jumps more than a 4th, using `+` or `-` is often more convenient than using a digit. This is because in order to use the digit, you must first enter the basic time value whether it changes or not.

For example `c44 d e f g a b c c0-` is an ascending quarter-note scale starting on middle C, followed by an octave jump down to a whole note middle C.

Explicit octave numbers can be combined with one or more `+` or `-` . In earlier versions, `+` or `-` was ignored if an octave number was specified. This is a slight backward incompatibility; **PMX** prints a warning when it happens.

Stem length can be **s**hortened or **l**engthened by x `\internote` with the options `S`x or `L`x . x is restricted to the range (0.5,4.0) for shortening and 0.5 to 27.5 for lengthening. The shortening can be made "sticky" by following the number x with `:` . Then every note's stem in the voice will be shortened until one is encountered with the option `S:` . By lengthening a stem enough to span to the next staff and connect with notes there, unflagged staff-spanning chords can be constructed. See section 2.2.4 for further details about staff-spanning chords. 2.73

The first note command in each voice in a block must contain at a minimum the note name or `r` for a rest (see below), and a basic time value. For notes, it is good practice and can simplify editing if in addition an explicit octave is set here. However if it is not, **PMX** will make some assumptions. At the start of the first input block the pitch will be set as if the prior note were middle C. In later blocks **PMX** will use the obvious inheritance rules from the end of the prior block. However, if the number of voices in a staff has changed from the prior block, it is safest to reset the octave at the start of a new block. Duration is never inherited and must be set at the start of each input block.

Dots can be a little tricky, because even though they affect the actual time value, they don't affect the basic time value, and it is only the latter that is "sticky". Therefore, if a note is to be dotted, you always have to enter a `d` (or a period, see next paragraph) somewhere within the command, after the note name, even if the actual time value and octave are the same as the prior note. But the *basic* time value need not be re-entered if it hasn't changed (unless the note is more than a fourth from the prior note *and* you have for some strange reason elected to indicate the octave with a number rather than `+` or `-`). So for example, consecutive dotted half notes, each within a fourth of the previous one, could be most cleanly entered as `cd24 ed gd ed`, whereas `cd24 e` would represent a dotted half note followed by a plain half note (since the basic time value—as defined by the first digit—was a half note all along).

There are two special shortcut rhythmic notations. For normal dotted rhythms (3:1 ratio), if you include a period (`.`) in the note command, it will (a) assign a dot to the note just entered, (b) terminate that note, (c) prepare to receive the next note name *without any space*, and (d) automatically assign a time value to the second note equal to one-third of the first one. No time value may be entered for the second note, but octave and accidental data may. Ornaments and slurs (see below) following this command will apply to the second member. If you need to follow the main note with some modifying command, you can still use the shortcut (`.`) after that command and a space. The main advantage of this shortcut comes if you want to follow one dotted pair with another of the same rhythm; then you needn't enter any explicit time value for *either* member of the second pair. This is possible because after using the shortcut, the basic (inheritable) duration is set to that of the *first* note in the pair, without the dot.

For paired notes with 2:1 rhythmic ratios, the character `,` (comma) behaves similarly to the `.` (period) for 3:1 rhythms.

Xtuplets, or groups of notes with their stems connected, can have from 2 to 24 notes or rests.

Normally they all have the same duration, but there are several options—described below—to change this. The command for the first note of an xtuplet begins exactly like a note or rest command, with the name of the first note in the xtuplet, or `r` if it starts with a **r**est (see subsection below on rests), and an optional time value. However, the actual time value (including a dot if present and a basic duration that may have been inherited from the prior note) now represents the *total* duration of the xtuplet. Next (with no space, as usual) comes `x` followed by either a one- or two-digit integer for the number of notes in the xtuplet, or `T` to initiate a 2-note tremolo, to be further discussed below. The only options allowed immediately following the number are `d` and `n` . `d` signifies that the *first* note of the xtuplet should have a dot and the second, and extra flag. `n` controls the printing of the **n**umber and bracket. If `n` is followed by a blank, then no number will be printed. On the other hand, an *unsigned* integer here is taken as a substitute number to be printed instead of the natural one. If one or two *signed* decimal numbers follow `n` (each starting with `+` or `-`), the first is a vertical shift in `\internote`s, and the second, a horizontal shift in notehead widths. Another suboption to `n` is `f`, to **f**lip the number vertically from its default position. A final suboption to `n` is `s` followed by a signed integer. It applies only to non-beamed xtuplets, for which it tweaks the slope of the bracket above or below the xtuplet. For non-beamed xtuplets, you can further change the appearance of the bracket and number as explain in section 2.3.8. 2.80

The second through last notes of the xtuplet are each then represented by a separate command containing a subset of the characters permitted for ordinary notes or rests: note name or `r` (the only required character), accidental, and octave change character (`+` or `-`). The octave may be given explicitly instead, and any integer will be interpreted as such, as no numerical time value is permitted.

To double the duration of any note in an xtuplet, add the character `D` to the command for that note. This will decrease the expected number of notes in the xtuplet by one. To add a dot to the doubled note (as Bach sometimes did), use `F` instead of `D`. To add a dot to one note and an extra flag to the next, include `d` in the note command, *after* the `x` if it's the first note of the xtuplet as noted above.

As an example, an ascending quarter-note triplet scale would be notated

```
c44x3 d e f4x3 g a b4x3 c d ...
```

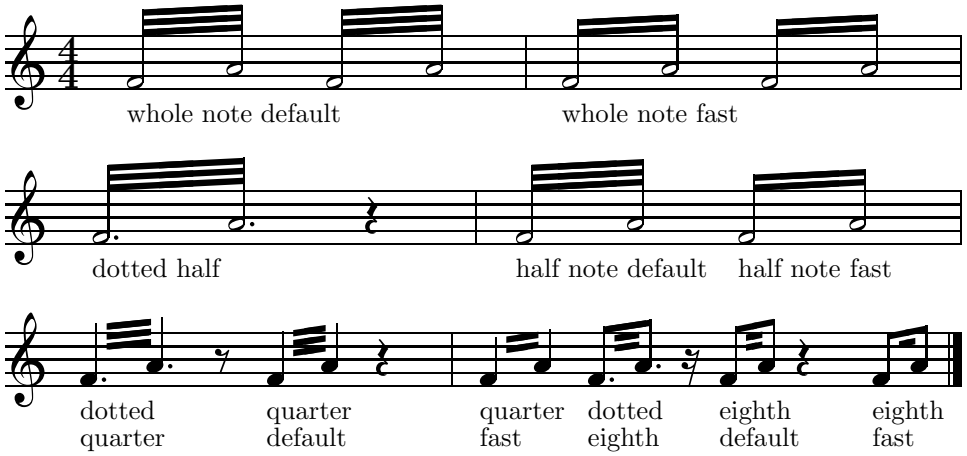
2.2.2 2-note tremolos

A 2-note tremolo is a special case of an xtuplet. It represent a rapid alternation between two notes. It is notated with a pair of notes, either beamed or unbeamed, with the possible addition of from one to three indented, disconnected beams between the two note stems. Like an ordinary xtuplet, it begins with a note name, optional duration and octave level, then the character `x`. The duration applies to the total time value of the two notes, and is currently limited to either a half note (`2`), quarter (`4`), or eighth (`8`). The duration may be dotted. Next comes a `T`. This is optionally followed by one or two integers from 0 to 3. The first indicates the number of ordinary beams connecting the two notes; the second, the number of indented beams. No other options are allowed, and some options are prohibited, such as zero ordinary beams on anything except a quarter or dotted quarter tremolo. If no integers are entered, defaults are assigned: (3,0) for a half, (0,3) for a quarter, and (1,2) for an eighth. After a space, the second note of the tremolo is entered. If the total duration is a half or dotted half, the noteheads will be open. For a whole note tremolo, two consecutive half note tremolos should be used. As with ordinary xtuplets, the horizontal spacing of the notes will always be the correct value for notes with half the duration of the total. 2.80

Either or both members of the tremolo may include chordal notes, using the normal `z` notation described in section 2.2.4. Also, if the tremolo is beamed, the height and angle of the beam may be adjusted after making it a forced beam as described in section!2.2.12. 2.84

Some examples are shown below. When two versions are shown for a given duration, the first is the default. The following **PMX** code generates the example.

```
f24xT a fxT a | fxT20 a fxT20 a /
L2 fd24xT a r4 | f24xT a f2xT20 a /
L3 fd44xT a r8 f44xT a r4 | f44xT02 a fd8xT a r1 f8xT a r4 f8xT11 a /
```



2.2.3 Rests

The command for a rest starts with `r`. Then for a normal rest, in either order come a digit for the basic time value (using same codes as for notes, optional if unchanged from previous value), a `d` if the rest is dotted, and a second `d` if double dotted. The basic time value of a rest affects future notes and rests the same as if it had come from a note, i.e., it applies until another value is entered with a subsequent note or rest in the same voice. The command `rp` represents a full-bar rest notated with a ***p**ause* character (whole rest) regardless of the time signature; in this case no other duration information is needed or allowed. `rb`, followed if necessary by a duration specifier, denotes a ***b**lank* rest, one that occupies space and time but is invisible. This is most often used when there are two voices in a staff and one drops out for some of the duration of the current input block. (See `mwalmnd.pmx` for examples). The option `o` (for **o**ff-center) suppresses centering a full bar rest. If you don't exercise this option, then *all* full-bar rests will be horizontally centered between bar lines, including pauses (`rp`) as well as normal rests that fill the bar. `rm` followed immediately by an integer will generate a ***m**ulti-bar* rest, a special combination of characters between two bar lines with an integer above representing two or more bars of rest. This command will generally only be used in separate parts after having been automatically generated by **scor2prt**. However, it may be used in a multi-line score, provided it is entered for the same number of bars in every staff.

The default vertical position of a rest depends on whether there are one or two voices in the staff. For one voice it is just the MusiXTEX default (approximately centered on the middle line). On the other hand, in the lower voice in a two-voice staff, the rest is lowered `4\internote`, while in the upper voice it is raised `2\internote`. The **PMX** default can be manually overridden by appending `+` or `-` and an integer representing the offset from the *middle* line of the staff (not from the **PMX** default if there are two voices in the staff!). So for example, in a single staff in 3/4 meter, two voices, each with a half note followed by its own quarter rest would be either

```
c24 r4 //
c25 r4 /
```

or equivalently

```
c24 r4-4 //
c25 r4+2 /
```

while

```
c24 r4+0 //
c25 r4b /
```

would produce two notes followed by a single, vertically centered rest.

Another way to override the default vertical positioning of rests is useful in keyboard scores, or in fact any score containing two voices on a staff. The option `K` (for **K**eyboard) in the `A` command generally causes rests to be aligned horizontally with notes in the voice in which they are entered. See section 2.3.8 for a detailed description.

2.2.4 Chords

Chordal notes, which always share a stem and the same time value as the prior note, are symbolized with `z` (for **z**ero time) followed by a note name and optionally an accidental, `+` or `-` as octave indicator, and `e` or `r` for a l**e**ft or **r**ight shift by one notehead width. No basic time value is allowed. If the main note is dotted, then the chordal note will appear with a dot regardless of whether a `d` is entered. The only time a `d` is required in a chordal note command is if the dot's position is to be adjusted; in this case the `d` is required, followed by one or two decimal numbers, each preceded by `+` or `-`. The first is the vertical shift in `\internote`s; the second, the horizontal shift in notehead widths. Any number of chordal notes can follow a single main note. The stem direction of a chord is controlled by the main note, but may be manually overridden with `u` or `l` in the main note command.

When chordal notes are beamed together, the default height and angle of the beam will be determined by the main note on each stem (the one without `z`). If a beam joining chordal notes looks bad, you can usually fix it either by changing which note acts as the main one, or by fine-tuning the beam parameters as described in section 2.2.12.

PMX uses a complex algorithm to automatically position accidentals in chords. If you are unhappy with the result, you can manually tweak the horizontal positions as described in section 2.2.1.

Although there is no dedicated command for it, chords can be made to span from one staff to another using various techniques. The approach will depend on whether the chord is single-stemmed with no flag, single-stemmed with a flag, or beamed. If beamed, it will also depend on whether it is an xtuplet or not. Examples of all the basic possibilities are contained in the sample file `staffcrossall.pmx`. 2.73

For unbeamed, unflagged staff-crossing chords, by lengthening the stem with the `L` option on the main note, it can be made long enough to join with an unflagged single-stemmed note or chord in the next staff. Single-stemmed notes with one or more flags can be joined across staves with a trick discovered by Andre Van Ryckeghem: In one staff create a standard note or chord with the stem pointing away from the other staff. In the other staff, place the chord notes in a one-note forced beam that has been lowered or raised into the first staff (e.g. `[-10 b14 ]`); that will stretch the stem to join the other notes, but with just one note (or chord) in the beam, the crossbar will have zero length and be invisible.

Beamed chords may also span from one staff to another, using joined beams (see section 2.2.12). The general approach is to construct a set of chords (or single notes, if the other chord notes are in the other staff) in each of the two staves, enclose each set in a separate forced beam, and join the two beams with `]j`...`[j` . It is important to remember that the lower staff is processed first. So in most cases, the end of the segment in the lower staff must be joined (using `]j`) to the start of the upper segment (with `[j`). It turns out that for non-xtuplet beamed chords,

in all cases where the chord at the beginning of the beam has a note in the *lower* staff, this works fine provided that the forced beams are of equal duration and cover the same time span, and that positions in either staff with no note are represented with blank rests `\rb` inside the force beam. So for example a set of beamed chords that starts only in the lower staff and ends only in the upper could be represented by

```
{\tt [+28 g83 g g rb ]j /
[jf rb g84 g g ] /}
```

This example highlights some other issues, viz., that the beam height or direction of one or both beam groups may need to be altered. Often this will require trial and error.

Unfortunately this two-group procedure breaks down if the first chord in the beam has no notes in the lower staff. There are tricks to get around this; the user is referred to the file `staffcrossall.pmx` for examples. However, there is a much more straightforward way to define staff-crossing beamed chords that begin in the upper staff: it simply requires defining the beamed group in each staff as an xtuplet within a forced beam. It turns out that the treatment of staff-crossing beamed xtuplets is more robust than for non-xtuplets, and will admit more intuitive coding. So for example, the reverse of the above example, where the beam starts in the upper staff and ends in the lower, could be obtained with

```
{\tt [+28 rb2x4n g3 g g ]j /
[jf g24x4 g g rb ] /}
```

where we used the option `n` to suppress printing the number. As you might expect, more general staff-crossing beamed chordal xtuplets follow the same concepts already described for non-xtuplets, but as noted, they are more robust and admit patterns that start in the upper staff and end in the lower one. `staffcrossall.pmx` also contains examples this approach.

2.2.5 Grace notes

A grace note command starts with a `G`. It is entered in its natural order, normally before the main note, but sometimes after. After `G` and before the note name, comes any combination of the following options: an unsigned integer (which may have 2 digits) representing the number of notes in the grace (default is 1), `m` and a digit for ***m**ultiplicity* (number of flags or beams, default is 1, 0 is allowed), `s` for **s**lur (joining all notes of the grace to the main note; no other **s** is needed on the main note), `x` for a slash (only for single graces), `l` or `u` to force the direction of the stem(s), `X` followed by a decimal number x to insert a gap of x notehead widths between a normal grace and its main note, `A` (for **A**fter) or `W` (for **W**ay-after) to associate the grace note with the *prior* note. 2.6
Next comes the only required character, the first note name. No time value can be entered, but if needed, the octave or an accidental can be given as in a normal note. Second and later notes must follow immediately in sequence, set apart by spaces, likewise without any time value, and without any intervening commands.

Normal or after-graces will be placed *immediately* before or after the main note; way-after's, as far to right as possible before the next note or bar line. If either type of after-grace is slurred, the slur will start on the main note and end on the last one in the grace.

2.2.6 Ornaments

Commands for ornaments are entered *after* their associated note command. The ornaments now available are shake (`ot`), **m**ordent (`om`), "x"- or "+"-shaped ornament symbols (`ox, o+`), pizzicato (`ou`), strong **p**izzicato (`op`), left parenthesis before notehead (`o(`), right parenthesis after notehead (`o)`), upper **f**ermata (`of`), **d**own **f**ermata (`ofd`), staccato (`o.`), tenuto (`o_`), two different se**g**nos (`og` or `oG`), **C**oda (`oC`), arbitrary-length wavy-line **t**rill with *tr* (`oT`), arbitrary-length wavy-line 2.71

trill without *tr* (`oTt`), sforzando (`o>`), duncecap (`o^`), **c**aesura (`oc`), and **b**reath (`ob`). All except the parentheses, staccato, tenuto, and down fermata will normally appear above the staff; the parentheses appear at the level of the note head, and staccato and tenuto just above or below depending on the stem direction. The only difference between staccato and pizzicato is the vertical positioning of the dot.

Either type of trill may immediately include an unsigned decimal number to specify the length of the printed symbol in current `\noteskip`s; the default is 1. Thus `oT0` represents *tr* with no wavy line.

Once the ornament type has been specified, most of them can be raised or lowered from their default position by appending a signed integer to the command, representing the vertical offset in `\internote`s. . A second signed integer specifies a horizontal shift from default in notehead widths.

The caesura and breath marks differ from the others in their default horizontal position, which is 0.5`\noteskip` past the note.

The `og` segno has several special properties. It must be entered in the first (lowest) staff, but will appear above every staff. Its vertical position cannot be altered, but if appended by a number...unsigned if positive...all appearances will be shifted horizontally by that number of points. On the other hand, the `oG` segno has a smaller symbol than `og`, applies only to the note after which it is entered, and can be shifted just as a normal ornament. 2.71

An ornament can be automatically repeated on a series of consecutive notes, provided the notes are all in the same voice and the same input block. To activate this feature, terminate the first ornament command with `:` . Then every note in that voice will have the same ornament until a note is followed by the repeat terminator `o:` .

2.2.7 Editorial accidentals

To place a small sharp, flat, natural, or question mark above the staff, after the affected note enter `oe` followed by `s, f, n` or `?`. You may also put a question mark right after the accidental.

2.2.8 Slurs

By default **PMX** will use MusiXTEX's built-in font-based slurs. But through user intervention it is possible to use either one of two different types of postscript slurs. *Type K* slurs, developed by Stanislav Kneifl, are directly supported by **PMX** and will be the focus of any future **PMX** enhancements. They are globally activated with `Ap` and several global defaults set with other options to the `A` command as described in section 2.3.8. If these are used, so will an alternate set of hairpins (see section 2.2.11). The other postscript slur option is Hiroaki Morimoto's *Type M* slurs. These are not directly supported by **PMX**, but are intended to be fully compatible with the default font-based slurs. To use them, one would use the in-line TEX command `\\input musixpss\` , and be sure *not* to enter `Ap` . From **PMX**'s standpoint they are no different from font-based slurs.

There are some advanced options available only with Type K postscript slurs, and a few obsolete ones only with font-based. At this point the main difference in functionality between the two is that with postscript, **PMX** provides support for true ties, which are shaped and positioned slightly differently from slurs. Future enhancements will probably only work with Type K postscript slurs. Some users do still prefer font-based, possibly because Type K postscript slurs are not visible in some DVI viewers. New users should experiment with the various types of slurs and decide for themselves.

The normal commands for slurs are `(` placed with a space before a note, and `)` placed after. The command `s` is equivalent to *both* of them (!), except that it always follows the affected note. With font-based slurs, `t` is equivalent to `s` but with several minor differences to be explained

later. With postscript slurs, `t` signals to use a true tie. The commands `s` and `t` are *toggles*, turning a slur or tie off if it's already on and starting one otherwise.

A slur or tie may end on a rest, but not start on one. The default ending height in this case will be the same as the starting height, and it may be tweaked as described below.

The first character is optionally followed by a single-character ID code `0-9` or `A-Z`, then by other options described below. ID codes are only needed if two or more slurs are open at the same time within one voice, such as when several chord notes are tied. Using ID codes in such cases tells **PMX** which open slur to close. ID codes cannot be used with font-based `t` slurs.

The rules for finding the default direction and position of a slur are complex; many factors enter into defining visually pleasing values. But there's no need for gory details here; the result will usually satisfy, and if not, it can easily be tweaked. The default direction of curvature can be overridden with `u` (**u**pper), `l` (**l**ower), or equivalently `d` (**d**own). Starting or ending position can be shifted from its default by entering one or two explicitly signed numbers. The first, which must be an integer, represents the vertical shift in `\internote`s; the second, which may be decimal, the horizontal offset in notehead widths. Starting or ending position of a postscript slur or tie can be made to align with the end of the stem of an unbeamed note by using the option `v`. No other options are permitted with `sv`, but any desired position can be forced with the numeric options. 2.71

The shape of the slur may be altered as well. This paragraph deals with font-based slurs, for which the shapes may be less than fully satisfying due to fundamental limitations of MusiXTEX. At the slur termination only, one or three more parameters may follow the ones just described. The first, a signed, nonzero integer, is a vertical adjustment to the mid-height of the slur in `\internote`s. The next two, integers between 1 and 7 following a ":", are alterations to the starting and ending slopes. These numbers are passed directly as arguments of the MusiXTEX macros `\midslur` (if only one is given) or `\curve` (if there are three).

For Type-K slurs, the shape may be changed locally by including `f` in either the slur's starting or ending command to flatten it a bit, or `h`, `H`, or `HH` to increase its curvature and raise or lower its middle by increasing degrees. The default curvature can be altered from normal with new suboptions to `Ap` as described in section 2.3.8. Local curvature tweaks will take precedence over the global default. A special option `n` to the slur command can be used to locally restore the normal curvature if the default curvature has been globally changed.

Another option peculiar to Type-K slurs and ties is to locally override the global setting for automatic height adjustment (to avoid tangencies with staff lines). The global defaults may be changed with the `A` command as described in section 2.3.8. To override the global setting for the current slur or tie only, use the option `p` in the command that starts the slur or tie, followed by `+` or `-` (to turn adjustment on or off), followed by `s` or `t` (for slur or tie).

A dotted slur is activated by including the option `b` (for **b**roken) in the command that starts the slur.

Slurs involving grace notes are specified within the command for the grace (see section 2.2.5).

For font-based slurs, the unique aspect of `t` slurs is that if one starts or ends on the same note as an `s` slur, the former will be moved away from the notehead to avoid a collision. *This only works if neither slur has an ID code.* This feature is only retained for backward compatibility.

The available options should cover most circumstances, but if not, the TEX macros `\isu` etc, defined in `pmx.tex`, can be entered as in-line TEX commands (see section 2.4). These commands have three arguments: slur number, vertical position (pitch, or offset from bottom staff line in `\internote`s), and horizontal offset in notehead widths. When using these commands, you must choose an explicit slur number. Use one large enough to avoid conflicts with **PMX**'s automatic slurs, which are numbered from 0 upward. Also, remember that non-spacing in-line TEX commands such as this one must come *before* the note they apply to, in contrast with the **PMX** slur toggles which may come after.

2.2.9 Ties

With font-based slurs, in **PMX** the only difference between ties and slurs is the default positioning. Ordinary slur ends are centered horizontally above or below the notehead, while tie ends are shifted inboard and closer to the midheight of the notehead. To specify a font-based tie in **PMX**, use a slur command and include the option `t` in it, somewhere after the initial `(` , `)` , `s` or `t` .

With postscript slurs, ties—indicated with `t` or `st`—will have similar differences in endpoint positions, but in addition will have a different shape (somewhat flatter) and will always end at the same height they start. There is also an option to the `A` command that affects ties across line breaks (see section 2.3.8). By default the second part of such ties will be drawn as a complete tie symbol. However, if you want them to be a *half tie*—a special shape that is horizontal at its left end—use the command `Ap+h` at the start of the file.

In addition to the notation options just mentioned, ties may also be indicated with the character { before the starting note and } after the ending note. 2.6

2.2.10 Line-breaking Type K slurs and ties

No special action is required if a slur or tie happens to cross a line break. However, some special, manual adjustments are available for Type K postscript slurs in these cases. The global option `Apl` by itself adjusts several parameters as described in section 2.3.8. Further, if `Apl` has been issued, then case-by-case adjustments for line-breaking Type K slurs and ties are available as suboptions to the slur commands. To tweak the horizontal and vertical positions of the end of the first segment, enter the suboption `s` in the command that starts the line-breaking slur or tie, followed by two signed numbers representing respectively the vertical shift in `\internote`s and the the horizontal shift in notehead widths. To tweak the position of the start of the second segment, follow the above by another `s` and two more signed numbers. The usual curvature options `h`, `H`, `HH`, and `f`, if included in the starting command for a line-breaking slur, will apply only to the first segment, and if in the closing command, to the second segment. If the tweaked slur or tie does not happen to come at a linebreak, the special position tweaks (after `s`) will all be ignored, and the curvature tweaks on the closing note will take precedence as they normally would.

2.2.11 Dynamics

After the affected note, enter `D` followed by one of the following `pppp, ppp, pp, p, ffff, fff, ff, f, mf, mp, fp, sfz, "[`*any text*`]", >,` or `<` . The last two are diminuendo and crescendo, and they are toggles, i.e., the first one of each starts the symbol and the next one ends it. The one surrounded by double quotes is an arbitrary text string no longer than 64 characters, which may include embedded TEX. With any dynamic mark, you can also enter position shifts, vertical as a signed integer representing the number of `\internote`s, then horizontal as a signed number representing number of notehead widths. There can only be one of the letter-groups on each note, but there may also be `D<` and/or `D>` on the same note. These must be entered as separate `D...` commands, and must come in the right order, e.g., 2.76

[*some notes*] `D<` [*more notes*] `D< Dffff D>` [*more notes*] `D>`

Hairpins may span from one input block to the next. 2.7

There are numerous context-sensistive automagic adjustments to the positions of all the dynamic symbols. If you don't like the result you can adjust the position as just described.

Due to MusiXTEX's limitations, there are some restrictions on hairpins when using font-based slurs. They cannot be longer than 68mm, they cannot wrap over a system break, and they must be horizontal. Finally, only certain specific lengths are available so some horizontal

position tweaking may be needed, especially when letter-groups and hairpins are combined. These restrictions are all removed when using postscript slurs.

2.2.12 Beams

For the most part, **PMX** automatically takes care of the details of defining beams: selecting which notes are beamed together, and setting the angle, direction, height, and *multiplicity* (the number of bars along the top or bottom). However, one may define a *forced* beam—which overrides **PMX**'s selection of which notes are beamed together—by surrounding the included notes with `[` and `]`, being certain to separate these commands and their options from the included note commands with spaces. One may also wish to edit certain features of a beam even when **PMX**'s grouping decision would otherwise be acceptable; here again the beamed notes must be set apart with `[` and `]`.

The `[` may optionally be followed immediately by several options.

`u` or `l` will override **PMX**'s selection of the direction of the beam, while `f` will **f**lip it from whatever **PMX** decided.

`j` **j**oins the beam grouping to a prior one started in another system (see below).

One, two, or three consecutive integers, each preceded with `+` or `-` , will affect the beam's appearance. The first integer is an adjustment to the starting level (in `\internote`s) and may range from -30 to 30; the second is a slope adjustment with the same permissible range; the third is an alternate adjustment to the starting level (in beam thicknesses) and may only range from 1 to 3, always acting to increase the stem length. The latter may be used to align consecutive horizontal beams which have internal multiplicity changes. For example, in 2/4 time, `c84 c1 c c c c8` would cause two beams but the first one would be lower than the second; `[+0+0+1 c84 c1 c ] c c c8` would align the tops of the beams with each other. Due to the complexity of **PMX**'s beam analysis procedures, these editing commands may sometimes produce unexpected results, and some iteration may be required to get exactly what you want. For example, `[+0+0+3 cd8 c3 c6 c ] c c c3 cd8` will not produce two aligned beams as desired, because when **PMX** analyzes the first beam, it automatically raises the starting level a bit for another reason, namely, to avoid too short a stem on the 64th notes at the end of that beam. In this case, the user could counteract **PMX**'s internal adjustment by using `[-1+0+3 cd8 c3 c6 c ] c c c3 cd8`.

The option `h` forces the beam to be **h**orizontal.

The character `m` followed by a digit 1-4 forces the **m**ultiplicity of the beam, the number of stem-joining bars.

By default, xtuplets are set apart with their own beam. To beam an xtuplet together with other non-xtuplets, just include it with the other notes in a forced beam.

Rests may also be included within forced beams, provided they are shorter than quarter rests, and of course that they come *between* the first and last notes under the beam.

It's now easy to define a repeating forced beam pattern. If the option `:` (colon) is included in the starting command `[` for a forced beam, then after you end the beam, more beams of the same duration will be forced in that voice, until stopped. They will be stopped at either the next regular forced beam, or the end of the input block for that voice, whichever comes first.

Some users may wish to define beamed groupings with subgroups joined by a single beam. The command `][`, standing alone between two note commands in a forced beam, causes the multiplicity to decrease to unity and immediately increase to its natural value for the next note. For example, `[ c14 c c c ][ c c c c ]` will generate two doubly-beamed groups connected by a single beam.

Related to this is a *single-slope beam group*, which is the same as described in the previous paragraph except that the beam disappears between segments. Segments should be separated by `]-[` standing alone between two notes inside the forced beam.

If there are large jumps in pitch between notes in a beam within a single staff, as a matter of taste you may wish to start the beam for example as an upper one and end it as a lower. **PMX** will never do this automatically, but you can accomplish it by forcing the beam with appropriately modified up/down-ness, starting level, and slope. If you use this technique, there are two details to note: (1) if there are any intermediate multiplicity changes, they will only be handled properly if the initially specified up-down-ness is consistent with the vertical position of the intermediate notes involved, and (2) for proper appearance in crowded scores you may wish to insert hardspace or shifts as described in section 2.3.9. Some examples are included in `most.pmx`.

Beams cannot normally jump staves. But if that is desired, start the beam normally in one staff, and terminate the part of the beam in that staff with `... ]j` . Then resume the beam in the new, adjacent staff with `[j ...` . For staff-jumping beams, it's OK to have just a single note inside one or both of the members. Some adjustment of the beam height and slope may be required. Sometimes the ending section's up-downness must be overridden; you will know this is so if the ending is shifted horizontally from its proper position by one notehead width. Each voice must still have the right number of beats, so you will probably need to fill time with blank rests after the first member of the beam in one staff and before the second member in the other. There can still only be one staff-jumping beam open at a time.

2.2.13 Clefs

A clef change is signaled by `C` followed by a single lower-case letter or digit using the code specified in section 2.1. If clefs come out at the wrong vertical position, refer to the note in `pmx.tex`.

2.2.14 Arpeggios

To set an arpeggio (a vertical wavy line), simply place the command `?` after the commands for both the first and last note. To shift the symbol to the left by x notehead widths, use the option `-`$[x]$. 2.6

2.2.15 Lyrics

2.73

Lyrics depend on the underlying TEX command `\pmxlyr` developed by Dirk Laurie, which is defined in `pmx.tex`. It in turn makes use of the macro package `musixlyr.tex` developed by Rainer Dunker. So to enable lyrics within **PMX**, you will need to ensure that `musixlyr.tex` is installed somewhere in your system where your TEX processor can find it.

Lyrics can be inserted by enclosing them in double-quotes inside the music line just before the first note to which they apply, as in `"us-ing lyr---ics now__ "`. Once the first double quote is encountered, **PMX** will ensure that `musixflx.tex` is input into the TEX file.

Lyrics for several notes can be defined in one go. The lyrics in each input string demand a specific number of notes, depending on the number of syllables, hyphens, and underscores. If there are not enough lyrics, question marks will appear; if too many, the excess syllables will not appear.

The rules for aligning lyrics properly with notes are as follows. Words are separated by whitespace, with any number of spaces counting as one. Syllables within a word that require just one note each are separated by a single hyphen. There are two ways to extend a syllable over two or more notes. If it is the last syllable in a word (like `"now__ "`), follow it with consecutive underscores, one for each extra note, and finally a space. It will be printed with a continuous underscore. To extend a syllable within a word (like `"lyr---ics"`), insert one extra hyphen (with no spaces) for each extra note, and it will be printed with some number of hyphens filling the proper space between syllables. Conversely, a tilde (`~`) between two words (with no spaces) prints a space between them while assigning the last syllable of the first and the first syllable of the

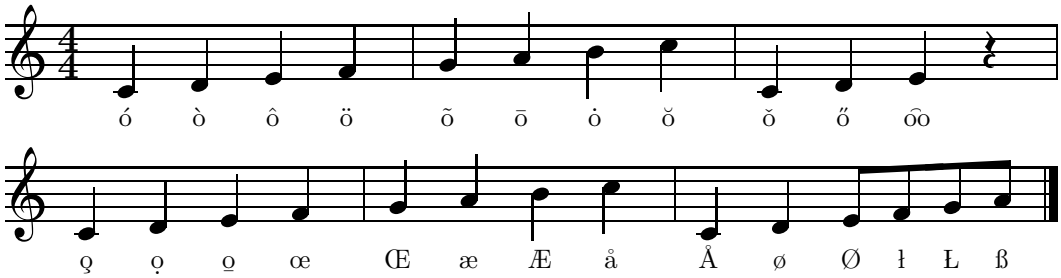
second to a single note[1]. So in the end, in a voice with lyrics, every note must be associated with a syllable, its extension, or two syllables joined with a tilde.

Although underscores within a word or consecutive hyphens at the end may not crash the code, they are not recommended for any foreseen useful purpose.

Accented characters can be included in lyrics or elsewhere in several different ways. Here we provide examples for just one of those methods, one which uses special TEX commands. The following **PMX** input contains most of the available accented characters and leads to the example pictured below.

```
"\'o \`o \^o \"o \~o \=o \.o \u{o} \v{o} \H{o} \t{oo}"@b+4
c44 d e f g a b c c- d e r /

"\c{o} \d{o} \b{o} {\oe} {\OE} {\ae} {\AE} {\aa} {\AA} {\o} {\O} {\l} {\L} {\ss}"
c44 d e f g a b c c- d e8 f g a /
```



By default, lyrics will be placed below the staff where they are entered, half way between that and the next lower staff. You may want to alter the vertical position of a lyrics line, especially if both voices in a staff have lyrics. This is accomplished with the opfion `@`, immediately following the closing quote of the lyrics string with no space. That must be followed by either `a` or `b` for **a**bove or **b**elow the staff, then a signed integer for the number of `\internote`s above or below the default height. This command is "sticky"; it will remain in force for later lyrics in the same voice until altered.

It may also be necessary to allow extra vertical space where the lyrics are positioned. There is no **PMX** command for this, but type 2 inline TEX can be used to insert extra vertical space above any instrument. For example, if the voice is in instrument #2 and lyrics are below that staff, `\\interinstrument=0\internote\\setinterinstrument1{8\internote}\` will add 8 `\internote`s in the space for the lyrics.

Present limitations allow lyrics at upper and lower voices on the bottom two staves of instruments 1 to 4. Elsewhere they are quietly ignored.

Most scores with lyrics will benefit from the type 2 command `\\sepbarrules\`, which stops bar lines from crossing through the vertical space between instruments.

The notation `"us-ing lyr---ics now__ "` is actually shorthand for the inline TEX string `\pmxlyr{us-ing lyr---ics now__ }\`. All the rules given in section 2.4 for Type 1 TEX strings apply. To ensure that the length of all the Type 1 TEX strings belonging to a particular note combined does not exceed 128, remember to account for the nine characters in `\pmxlyr{}`.

This way of entering lyrics is a convenient interface to a small subset of the facilities offered by `musixlyr`. If more advanced features than those supported by `\pmxlyr` are needed, the necessary `musixlyr` macros could be entered as in-line TEX directly into the .pmx file - see the example file `netsoos.pmx` for some of those.

If really advanced features are needed (such as having several verses of lyrics at once), most users would prefer the convenient interface to `musixlyr` via the program **M-Tx** developed by Dirk

[1] This is just an example of using a standard TEX feature within lyrics.

Laurie. It is a pre-preprocessor which produces a `.pmx` file containing the proper in-line TEX commands. Its input language is similar (but not identical) to **PMX** and includes most **PMX** functionality as a subset.

2.3 Commands That Affect All Voices

Most commands that affect all the voices can only appear in the first (lowest) voice in the first (lowest) staff. Most such commands will automatically be transferred from score to parts when separate parts are generated by **scor2prt** (see section 3).

2.3.1 Repeats, double bars, forced single bars

Repeat signs, double bars, and other bar-ending options are signaled by `R` followed by `l, r, lr, d, D, dl, b` or `z` for **l**eft repeat, **r**ight repeat, **l**eft-**r**ight repeat, thin-thin **d**ouble bar, thin-thick **D**ouble bar, thin-thin **d**ouble bar followed by **l**eft repeat, single **b**ar, or blank (invisible) barline. Some of these have peculiarities. `Rb` forces a single bar before a movement break (see section 2.3.12), where otherwise by default there is a double bar. That can be useful for example if you change the number of instruments (via an option in the movement-break command) in the middle of a movement. `Rz` will cause a blank barline at the end of the current system, not necessarily the current bar. It can be used together with blind meter changes if you want to split a bar across a system break. If `Rlr` falls at a system break, **PMX** will automatically split it in two. The command `Rdl` will likewise be split at a system break, but if not at a system break, the `d` will be ignored.

These commands must be in the first voice. It is best only to place them before the first note in an input block or if necessary after the last one; otherwise **scor2prt** may behave erratically. Using two separate `R` commands in succession will cause unpredictable results.

2.3.2 Voltas (first and second endings)

Beginnings and ends of first and second endings are signaled by `V` (for ***v**olta*). If it's the *end* of the volta, add the option `b` (for ***b**ox*) or `x` for *no bo**x***. If it's the *start* of a volta, you can optionally enter any text at all that doesn't include a space and doesn't start with `b` or `x` (most commonly `1` or `2`). A period will automatically be appended to the text. If one volta ends and another starts right away, only a single `V` is needed. Voltas must only be entered in the first voice. If separate parts are to be created from a score using **scor2prt**, then only a single volta is allowed in any given input block, and it must be at the beginning of the block.

2.3.3 Meter changes

Meter can only be changed at the beginning of an input block. A ***m**eter change* command starts with the letter `m`. There are two different ways to complete the command.

Method 1. Enter 4 numbers with no intervening spaces. The four numbers are `mtrnuml`, `mtrdenl`, `mtrnmp`, `mtrdnp` as defined in section 2.1, with the following exceptions for this method only: You must use `o` to represent the number 1; if you enter the digit `1` then **PMX** will interpret that digit and the next as a 2-digit integer, between 10 and 19 inclusive. 19 is the largest number that can be entered with this method. Note that `mtrdenl=0` still represents a whole note.

Method 2. Enter the four numbers verbatim in the order just listed, but separate them with slashes (/).

2.3.4 Fundamentals of key changes and transposition

As explained in section 2.1, the intial key signature, also called the concert key, is specified in the setup data. In order to change the key signature or to transpose (i.e. make the printed notes appear at a different level than where they were entered), use the `K` command. The syntax is `K`[*n*][*k*] where *n* and *k* are explicitly signed digits respectively giving the distance to transpose in `\internotes`, and new key signature. When transposing, you should always use relative accidentals, activated by the separate command `Ar` at the start of the first input block (see section 2.3.8). For example, to transpose a piece in C major to E major you would enter `Ar K+2+4` at the beginning of the first block.

To transpose by a half step to a key with the same letter name, use `K-0`[*k*] where as before *k* is an explicitly signed integer giving the new signature. (Using `-0` instead of `+0` eliminates confusion with a simple key change, see the next paragraph.)

A simple key change can be signalled at the start of any input block. Use the command `K` with *n*=`+0` as the first argument and the new key signature as the second.

If the signature changes from sharps to flats or vice-versa, the default will be to include naturals in the first instance of the new signature. To suppress this behavior, use the option `n` right after `K` . For example, to change from 2 flats to 3 sharps and suppress the naturals, enter `Kn+0+3` . 2.7

The procedures described above will affect all instruments in the score. To change the key of or transpose just a single instrument, use Ki[*m*][*n*][*k*] when *m* is an unsigned integer representing the instrument number, and *n* and *k* are as just described. For more than one instrument, you may immediately repeat everything after `K` (including `i`). This may come either at start of score (right after setup) or at the beginning of any later input block. But if it's later, it must be preceded by a normal (full score) non-transposing key change command `K+0`[*k*]. For example, to change the keys of the second and third instruments to one sharp and two sharps respectively, use `Ki2+0+1i3+0+2` .

2.3.5 More on transposition; "transposing" instruments and example files

In practice, two fundamentally different situations may arise: (1) (Full-score transposition) A score that has been entered in one key is to be completely transposed to a different key and pitch level, usually to force the range to fit different instruments than original; or (2) ("Transposing" instruments) some of the instruments require a part printed in a different key and at a different pitch level than it sounds.

To transpose an entire score from the key specified in the setup data (Case TTA), simply use the `K` command at the beginning of the first input block, as outlined in the previous section.

It gets more complicated when some transposing instruments are involved, because there are three different possibilities: (Case CTS) The transposing instruments can be entered in concert key but printed transposed in the score and in separate parts created with **scor2prt** (see section 3); (Case TTS) They can be entered transposed and printed transposed in both the score and in parts; (Case CCS) They can be entered in concert key and printed in concert key in the score, but printed transposed in parts. Matters are further complicated if there is a later key change. Finally, if a MIDI file is to be produced, then in cases CTS and TTS an additional step involving the transpose option to the MIDI command `I` (see section 4) must be taken.

All of the required commands for all four of these cases are summarized in the table below. Following that, they are discussed a bit further and illustrated in four example files (named [Case]`.pmx`) that are also included in the distribution.

Case	PMX entry	Printed score	MIDI pitch	Initial commands	Later key change
TTA	All B flat major, later key change to B flat minor	All transposed up 2 steps to D major, later to D minor	transposed	`K+2+2` `I`	`K+0-5`
CTS	All B flat major, later key change to B flat minor	Trombone (1) concert; alto sax (2) transposed up 5 to G, later G minor; clarinet (3) transposed up 1 to C, later C minor.	concert	`Ki2+5+1i3+1+0` `IT+0-5-1`	`K+0-5` `Ki2+5-2i3+1-3`
TTS	Trombone (1) concert; alto sax (2) transposed up 5 to G, later G minor; clarinet (3) transposed up 1 to C, later C minor.	Trombone (1) concert; alto sax (2) transposed up 5 to G, later G minor; clarinet (3) transposed up 1 to C, later C minor.	concert	`Ki2+0+1i3+0+0` `IT+0-5-1`	`K+0-5` `Ki2+0-2i3+0-3`
CCS	All B flat major, later key change to B flat minor	All B flat major, later key change to B flat minor; parts printed transposed	concert	`%2K+5+2` `%3K+1+0`	`K+0-5`

Case TTA: Full score transposition. Here the entire score is to be transposed. In the setup data the signature is set to -2. Then the command `K+2+2` says to transpose up 2 steps from the initial key of B flat to D, and put 2 sharps in the key signature. No special attention is needed for the MIDI; it will come out in the transposed key. A later (full-score) key change requires another `K` command, but now the transposition parameter is set to 0 and the new key is the concert key (I guarantee people will be confused by this). In the example the command for the signature change is `K+0-5` , making the new concert key B flat minor with 5 flats, and, considering the initial transposition, causing the score and MIDI to come out in D minor with 2 flats.

Case CTS: Parts all entered in concert key, but some transposed in the printed score. Here, to produce the printed score, parts are all entered in concert key, but instrument-wise transposition is used for the transposed instruments. In the example the alto sax part is entered in B flat but will be transposed up 5 steps in the printed score, to G major. This is brought about with `Ki2+5+1` . Similar logic applies to the clarinet part, while the trombone part is not transposed. If a MIDI file is desired, it will come out in concert key, but only after using the transpose option in the MIDI command to undo the transpositions caused by the K command. In the example the command `IT+0-5-1` does this, "de"transposing each of the three instruments by the necessary number of steps. For a later key change, first the full-score `K` command changes the concert key, then the instrument-wise `Ki command`, with the same transpositions as the initial one, sets the new key signatures for the transposing instruments. Here the signatures to be entered are the transposed signatures, i.e., the ones that will be printed.

Case TTS: Parts entered in respective transposed keys, and printed in those keys in the score. In this method of scoring transposing instruments, parts for transposing instruments are transposed ahead of time and entered exactly as they will appear in the score. So to produce the printed score this way, the pitch does not have to be changed, but the key signatures must be set separately for each transposing instrument using the `Ki` command. In the example, the alto sax is

entered in the key of G so the instrument-wise option for it is `Ki2+0+1` . Note that `+0` means no further transposition is needed before printing, because the part was transposed on entry. Once again, if a MIDI file is desired, it will come out in concert key, but just as in the previous case, you must use the transpose option in the MIDI command `IT` to undo the transpositions caused by the `K` command. For a later key change, the same full-score `K` command as in the previous case is used to change the concert key. Then the instrument-wise `Ki` command, now with `+0` for the transpositions, sets the new key signatures for the transposing instruments, again using the transposed signatures.

Case CCS: Parts entered in concert key, printed in score in concert key, but transposed in separate printed parts. This is the easiest case of all. Nothing special needs to be done for the score, but part-only, full-score transposition commands `%`[instrument number]`K...` should be entered in the score. Then **scor2prt** will generate a transposed part. Of course if a MIDI is made from the score it will come out at concert pitch. For example, to transpose the alto sax part up 5 steps, initially to G major, near the top of the score file enter `%2K+5+1` . Later, where the concert key changes to B flat minor and the alto sax to G minor, enter simply `K+0-5`, making the new concert key B flat minor with 5 flats. When **scor2prt** is invoked to make separate parts, this will be transferred verbatim into all parts, and then **PMX** will internally adjust the signature for each transposed part as required.

Making separate parts. In all of the cases discussed, if the patterns of commands in the table are followed, then separate parts can be made as usual using **scor2prt**. They will automatically come out transposed as desired.

Texts of the transposition sample files:

```
%TTA.pmx
3 2 4 4 4 4 0 -2
1 1 20 .13
Trombone II+III
Trombone I
bbb
.\bs
Tt
TTA
Apr
I
K+2+2
b42 d f b t b t gf df b /
b42 d f b t b t gf df b /
b42 d f b t b t gf df b /
K+0-5
b42 c d e f gs as b /
b42 c d e f gs as b /
b42 c d e f gs as b /
```

```
%CTS.pmx
3 3 4 4 4 4 0 -2
1 1 20 .1
Trombone
Alto Sax
Clarinet
btt
.\
Tt
CTS
Apr
Ki2+5+1i3+1+0
IT+0-5-1
b42 d f b t b t gf df b /
b43 d f b t b t gf df b /
b44 d f b t b t gf df b /
K+0-5
Ki2+5-2i3+1-3
b42 c d e f gs as b /
b43 c d e f gs as b /
b44 c d e f gs as b /
```

```
%TTS.pmx
3 3 4 4 4 4 0 -2
1 1 20 .1
Trombone
Alto Sax
Clarinet
btt
.\
Tt
TTS
Apr
Ki2+0+1i3+0+0
IT+0-5-1
b42 d f b t b t gf df b /
g44 b d g t g t ef bf g /
c45 e g c t c t af ef c /
K+0-5
Ki2+0-2i3+0-3
b42 c d e f gs as b /
g44 a b c d es fs g /
c45 d e f g as bs c /
```

```
%CCS.pmx
3 3 4 4 4 4 0 -2
1 1 20 .1
Trombone
Alto Sax
Clarinet
btt
.\
Tt
CCS
Apr
%2K+5+1
%3K+1+0
I
b42 d f b t b t gf df b /
b43 d f b t b t gf df b /
b44 d f b t b t gf df b /
K+0-5
b42 c d e f gs as b /
b43 c d e f gs as b /
b44 c d e f gs as b /
```

2.3.6 Text

The commands `h` or `l`, when placed in the first column of an input line and followed by a blank or, for `h` only, by a signed integer, stand for ***h**eader* and ***l**ower text*. They will put a text string above or below the *top* staff in the *first* bar of the block where they are entered. The text string must be on a line of its own, immediately following the command. The integer is a vertical shift in `\internote`s.

A *title block* with up to three elements can be defined at the beginning of the first input block. `Tt` signals that the text *on the following line* is to be set as a **t**itle for the whole piece, and it will be centered. `Tc` similarly indicates a **c**omposer's name, to be set below the title and right justified. `Ti` likewise stands for an **i**nstrument name, which will be set above the title,

left-justified. The text for any of these commands can be split over two or more lines by including \\ at the location of the line break.

`Ti` will automatically be invoked by **scor2prt** when it generates parts from a score.

Extra vertical space can be added between the title block and the top system by appending to `Tt` a one- or two-digit number representing the space in `\internote`s. This only works if `Tt` is the *final* title block element entered.

The `D` command can be used to enter arbitrary text as described in section 2.2.11.

Lyrics may be entered as described in section 2.2.15. 2.73

2.3.7 Page numbering, centered header text

If you want pages to be numbered at the top left or right, place the command `P` anywhere within the **PMX** code that represents the first page to be numbered (usually the first or second page). `P` can be followed optionally by the starting page number and/or by `l` or `r`, the latter overriding the default locations of odds on the right and evens on the left. There is also a special option `c` for **c**entered header text. It must be the *last* option in the `P` command. It will define text to be printed at the top of every page *after the first*. If a blank follows `c`, the default header text will be the instrument name entered with the command `Ti` . If any non-blank character except `"` follows `c`, the header text will start with that character and end at the next blank. If `"` follows `c`, the header text will be everything between that and the next `"` (this permits headers containing spaces). The `P` command and its options will be ignored when making parts from a score (since page numbering will usually be different in the score than in the parts), but page numbering (and centered headers) for parts can be still be initiated independently, for example with `%!P2` or `%1P2r` (see section 3).

2.3.8 Overriding certain defaults, or getting the most from PMX

Understanding this section is important if you want to get the most out of **PMX**. In many cases the switches described here represent subtle but significant improvements that have come along since **PMX** was initially developed. Rather than changing the defaults, they are treated as optional in order not to upset the layout of older scores. For example, virtually every new score I create begins with at least `Abple`.

As you may have guessed, it is the command `A` that can be used to override a grab-bag of default settings. The available options affect a wide range of **PMX** features: sizes and interpretation of accidentals, dot positions, space before the first note of every bar, space between staves, slur package selection, vertical positioning of Type K postscript slurs, line-breaking Type-K slurs, curvature of Type-K slurs, naming of parts, vertical positioning of rests in 2-voice staves, brackets for non-beamed xtuplets, and inputting so-called *normal include* files. 2.6

Size of Accidentals. `b` makes all accidentals **b**ig, `s` makes them all **s**mall. By default, big ones are used unless unaltered spacing doesn't provide enough space. Thus the default behavior may cause a mixture of big and small accidentals, and in fact is not recommended.

Relative accidentals. If transposing, then the **r**elative accidental convention must be used, indicated by `r`. This changes the way you enter accidentals. With relative accidentals, the note options `s`, `f`, `n` take on unconventional meanings, now respectively signaling that a note should be raised a half step, lowered a half step, or left alone *relative to the pitch it would have according to the key signature*. So for example, with `Ar`, in the key of B flat major the note command `bs` would cause a B natural to be printed. By contrast, the default is the normal, absolute convention, where the indicated pitch alteration is relative to what the pitch would be if there were no key signature.

Vertical position of dots. If there are staves with two voices, `d` causes dots in the lower one to appear on or *below* center, in contrast with the default.

Gap at start of bar. Use `a` followed by a decimal number to override the default setting for `\afterruleskip`, the space before the first note in a bar. The default in **PMX** is `1\elemskip`, 20 percent smaller than MusiXTEX's.

Space between staves within a system. If **PMX**'s vertical spacing between staves within a system is not pleasing, use `I` or `i` , followed by a decimal number, to apply a scale factor to `\interstaff` . `I` affects all pages, `i` only the current one. Shrinking the space between staves within each system will cause the space between systems to increase, and conversely. These options have no effect if there is only one staff per system.

Equal space between systems. MusiXTEX normally draws a virtual box around each system and inserts equal vertical space between boxes. When objects protrude above the top staff in a system or below the bottom one, this can lead to unequal spacing between the top staff line in one system and the next. If you prefer that the vertical spacing between the staves of consecutive systems be constant for the whole page, use the `e` option of the `A` command. One side benefit of `Ae` is that it will prevent systems from spilling over onto extra pages, regardless of how many systems are put on the page. When using this option, you may occasionally want to force more vertical space between certain systems. There is a TEX macro `\spread` that can be inserted anywhere in the system before the desired wider gap. It has one argument, the desired extra space in `\internote`s.

Stop grouping systems at top in sparse pages. Another command affecting vertical spacing is the `v` option of the `A` command (for **v**ertical). **PMX** normally spreads staves vertically over a full page, unless the white space becomes excessive, in which case it groups all staves near the top of the page. Entering `Av` will suppress this grouping near the top, and ensure that systems will always be spread vertically regardless of how much white space is left. It is a toggle; the second time it is issued, the behavior reverts to the default.

Add extra vertical space before and/or page eject (last resort) As a last resort in getting the right spacing at the top or bottom of a page, the option `V` will insert a vertical skip of the specified number of `\Internote`s before and after the next page eject. It must be followed by `+` or `-`, then a number, then another `+` or `-` and number. 2.8

Make some staves smaller. The `S` option to the `A` command allows you to specify a different size for selected staves and their notes compared to the global value set in the setup data. It is followed by exactly `noinst` characters, one for each instrument, selected from `0`, `-`, `s`, or `t` for normal, small, small, or tiny sizes respectively. 2.7

Postscript slurs. The command `Ap` activates Type K **p**ostscript slurs. To use this you must have `musixps.tex` somewhere that TEX can find it, and `psslurs.pro` somewhere that **dvips** can find it. If these files happen to be missing from your TEX distribution, they can be found here. Several suboptions affecting Type K postscript slurs are described here and in the following paragraphs. First, by default these slurs and ties will not have their vertical positioning tweaked to avoid tangencies with staff lines. To activate this type of adjustment, use one of the suboptions `+s` or `+t` for slurs or ties respectively. (For example, `Ap+s`). Be warned that this may alter the endpoint positions from what one would normally expect. To deactivate the adjustment, use the same command but with `-` . Another suboption of `Ap` affects line-breaking slurs. Normally a full tie is drawn at the start of the second line. However, the suboption `Ap+h` causes the use of ***h**alfties* for the second part, which are flattened at their left-hand end, and require the special font `mxsk` provided with the Type K postscript slur distribution. It may be cancelled with `Ap-h` .

The suboption `l` (e.g. `Apl`) activates some other tweaks and tweaking capabilities for line-breaking Type K slurs and ties. It automatically tweaks the horizontal positions of the end point of the first segment and the start of the second, uses a normal tie character for both segments of a tie, and enables further tweaking of the horizontal and vertical positions of internal endpoints on a case-by-case basis, using options in the initial slur or tie command (see section 2.2.10, and the end of the fourth system in the example file `barsant.pmx`).

Another pair of suboptions to Ap affects the default curvature of Type-K postscript slurs. Ap+c and Ap-c will respectively increase or decrease the default curvature of all slurs to the next level in the sequence f, n, h, H, HH . (Here n stands for **n**ormal.) Several levels may be traversed by repeating the suboption, e.g., Ap+c+c increases the default curvature by two levels. If you try to go outside the allowable range, a warning will be issued, the curvature will be set to f or HH , and processing will continue. See section 2.2.8 for further details.

If your score contains Type K slurs and if you use a program such as **dviselec** to extract single pages from a .dvi file, you should use the suboption h (e.g. Aph) . This will cause the header file psslurs.pro to be written into the postscript file at the top of of every page.

Vertical rest positioning in keyboard scores. The option AK activates special rules for vertical positioning of rests in two-voice staves. By way of background, without this option, rests in two-voice staves have default positions based on a simple rule that is not context-sensitive: those in the lower voice (the one before //) are 4\internotes below their single-voice default positions, and those in the upper line are 2\internotes above the single-voice default. In contrast, the option AK invokes a set of context-sensitive rules to set the default position. The baseline rule is to align the rest in a horizontal line with the next following note in the same bar. If there is no following note in the bar, then it is aligned with the next prior note. If there are simulataneous rests in both voices, the old rule is applied. When the AK option is in force, it only affects places where there are two voices in a staff. It may be toggled on and off at the beginning of any input block, using just AK. When the option is in effect, any user-defined tweaks on the height of a rest will supersede the option for that particular rest, i.e., the tweak will be applied relative to the single-voice default position. When AK is in effect, the option L in a rest command will cause the vertical position of that rest to be based on the *preceding* note, rather than the following one as is the default. 2.6

Names of PMX files for parts. The option N to the A command allows you to specify arbitrary names for the part files generated by **scor2prt**. Follow AN with the part number and the new file base name in double quotes. Immediately follow this with any number of additional part numbers and alternate file base names in quotes. When part files are generated, .pmx will be appended to the requested base name.

Gapped bracket for nonbeamed xtuplets. Non-beamed xtuplets will normally be printed with a bracket above or below, and a number above or below that. If you would like this number instead to be positioned within a gap in the bracket itself, enter AT .You must have tuplet.tex available to your TEX processor. If missing, this file can be found here.

"Include" file. PMX commands in an external file can be included at the start of any input block by designating the file as a *normal include* file, using the command AR[*filename*] . See section 2.7 for details.

Positioning printed pages. For printing on **l**etter or a**4** paper, the command Acl or Ac4 will set the margins of the printed area so it will be properly centered with no further adjustments needed when running **dvips**. 2.7

2.3.9 Extra hardspace, horizontal shifts

Despite the author's best intentions to relieve you of the chore of adjusting *any* horizontal spacing by hand, there may be some occasions where you will want to do it. A command starting with X initiates one of two types of horizontal adjustment: A *shift* moves one or more characters but does not affect any other spacing anywhere; a *hardspace* inserts a fixed amount of space at a particular time and affects the horizontal positions of everything in all staves in the system. If the command includes S, it is a ***s**ingle* shift and affects only the next note or rest. If it includes a : it either starts or terminates a *group* shift. All X commands except group shift terminations must include a decimal number for the size of the offset in notehead widths. If the number is immediately followed by p, then the number represents points, otherwise, notehead widths. If there is no such

number but there is a `:` the command signals a group shift termination. Group-shift commands must occur in start/terminate pairs, and group shifts cannot extend across a bar line.

An `X` command containing neither `S` nor `:` is automatically a hardspace.

Because horizontal spacing in parts will usually differ from that in the score, by default the hardspace command will *not* be copied into parts by **scor2prt**; however the shift commands will be copied. These behaviors can be overridden using the methods to be described in section 3, Alternatively, to help keep **PMX** score files neat and readable, the character `B` can be used within the `X` command to signify that it applies to **b**oth score and part, or `P` for **p**art only.

2.3.10 Minimum spacing between notes in crowded systems

PMX does some special, complex analysis to adjust horizontal spacing in crowded systems. By default, the minimum space between consecutive noteheads is 0.3 notehead widths. In very special situations you may want to change 0.3 to some other fraction. To do so, enter `W.` (decimal point is required) followed by `1-9` to represent the number of tenths of a notehead width to be used as the minimum spacing. Use of this option is demonstrated in the example file `barsant.pmx`.

2.3.11 Page size

The default page size is 740 by 524 pt (10.3 by 7.3 in). To change the height or width, use the special commands `h`[*n*][*u*] or `w`[*n*][*u*] at the beginning of the first input block. Here *n* is a decimal number for the new dimension and *u* defines the units; `i` for inches, `m` for millimeters, and `p` or nothing for points. This command can be used together with `%%` or `%!` (see section 3) to give the parts made by **scor2prt** different page sizes than the parent score.

2.3.12 Line, page, and movement breaks

It is possible to force line, page, or movement breaks anywhere. For a line break, just enter `L`[*n*] at the start of an input block (in the first voice only), and the *n*-th system will start there. To start page *m* at line *n*, enter `L`[*n*]`P`[*m*]. You can't force a page break without first forcing a line break.

To force a movement break, you must first force a line break as above, then enter `M`. If a page break also occurs here, the `P` must precede the `M`. Options following `M` are `+`[*integer*] to insert vertical space in `\internote`s before the break, `i`[*decimal number*] to reset the first-line indentation as a fraction of the line width, and `c` to **c**ontinue bar numbering rather than resetting the bar number to 0. Also, to change the **n**umber of instruments, enter `n`[*integer*], then the number of each instrument in their new order, then a clef-designating character for each staff of each instrument. (An instrument's number is simply its position in the original sequence.) There can never be more than the original number of instruments. In this instance, two-digit instrument numbers must be preceded with `:` (colon). If you want to start with some number of instruments and later increase it, you'll need to insert a dummy page at the beginning with the full set of instruments, then start the second page with a movement break and decrease the number there.

Another option after `M` is `r+` or `r-`, which either forces or suppresses **r**eprinting the instrument names. The default is to print them if the number of instruments changes, but otherwise not.

Immediately after a movement break, any desired meter changes, key changes, or text can be entered in the normal way.

2.3.13 Fractional bars

Often if a piece starts with a pickup, the last bar may not be complete. In such cases, it is usually possible to place the last bar in an input block by itself, headed by a *blind* meter change. For

example, if the meter had been 4/4 and there was a quarter note pickup, leaving 3 beats in the last bar, the last bar might be coded `m3400 cd24 /`.

2.3.14 Stem direction of bass notes

By default **PMX** makes stems go up for middle-line D's in bass clef, but down for notes on the middle line of all other clefs. If you want middle-line bass-clef notes also to have downward stems by default, enter a `B` near the beginning of the file.

2.4 Putting TEX Commands into the PMX File

There are five ways to enter TEX commands into the `.pmx` file. Four of them are *in-line*, where the commands are entered directly; the fifth is by way of an external file.

The four categories of in-line TEX strings differ mainly in where they will appear in the `.tex` file. (A TEX *string* consists of a starting character, a sequence of TEX commands, and a terminal character). In the `.pmx` file, only type 4 TEX strings may wrap over line breaks. All in-line TEX must adhere to the 128-character limit per line, but each line can have more than one TEX command. Type 1 begins with a single `\` and will appear in the `.tex` file right before the TEX command for the next note or rest in the `.pmx` file. Multiple type 1 strings associated with the same note or rest are allowed, although the total length may not exceed 128 characters (so there is generally no reason not to combine all TEX commands for a single note into a single type 1 string).

A type 2 string begins with `\\` and will appear near the top of the `.tex` file, right before `\startmuflex`, regardless of where it appears in the `.pmx` file. A type 3 string starts with `\\\` and will appear right before the `\xbar` or `\alaligne` at the beginning of the current input block, before the first barline of the block. While individual type 2 and 3 strings may not wrap over line breaks in the `.pmx` file, strings of like type on consecutive lines will appear together in the `.tex` file. Types 1, 2, and 3 strings must end with `\` (backslash-space). This means that they may not contain the TEX macro `\` (backslash-space). Finally, each type 2 or 3 string should be isolated on a line of its own, and should be started in column 1.

Type four permits multiple lines of arbitrary text to be entered at the top of the `.pmx` file; they will be transferred verbatim to the top of the `.tex` file. Type four is initiated with `---` alone as the top line of the `.pmx` file. Then follows any text on any number of lines, until the next line starting with `---` terminates the block to be transferred.

The only other distinction among the types of in-line TEX strings arises when **scor2prt** is used to make separate parts (see section 3): types 2-4 will be copied into all parts, while type 1 only goes into its original part.

If you should want to enter a type-1 (note-based) string longer than 128 characters, you could use a series of type-2 or -3 strings to define a TEX macro containing the desired commands.

PMX provides one further option for entering an unlimited set of TEX commands just before `\startmuflex`, and before any Type 2 in-line TEX strings. Simply put the commands into a text file named [*basename*]`.mod` in the texinput directory. It will then automatically be entered with an `\input` command. This feature is retained mainly for backward compatibility; it has been essentially replaced by the various options for in-line TEX strings.

2.5 Figured Bass

Figure commands are entered *after* their associated note commands. They only work in the first (lowest) voice, and in any one other voice. Enter the characters as they would appear from top to bottom, and as you might pronounce them, e.g., `64` or `73`. Flats here are `-` (minus), sharps are `#`, and naturals `n`, *before* the number (if there is a number) (notice the characters are different here

2.6

than in notes). So for example *sharp third* is `#3`, just a sharp is `#`, *six (over) flat five* is `6-5`, and *sharp six (over) 4* is `#64`. In addition to the symbols just described, the following special symbols are available: 2+, 4+, 5+, 6+, 9+ . To use them, you must have the font cmrj in your TEX system, and then just put an `s` after the number. 2.6

The program positions all the figures for each system below the lowest staff of that system, with their tops aligned, and just low enough to clear the lowest beam, notehead, or stem that could interfere. If you would like to change the vertical alignment for the remainder of the staff starting at a figure after the first, append `v`[*n*], where *n* is an integer representing the vertical shift in `\internote`s, which may have a minus sign. 2.71

If you want a figure to align horizontally in the second tier, insert the placeholder figure `_` (underscore) before the one you want lowered. This is equivalent to lowering the figure stack by 4 `\internote`s. If you want to *raise* the entire stack by an integral number of `\internote`s, append `+` and the number. This can be combined with the placeholder figure `_` to provide full control over the vertical position of the stack. 2.6

Sometimes you may need to enter a figure when there's no bass note sounding. To do this, just after the most recent bass note enter `x`, followed by a two single digits (the first is a repeat count; the second a time value, i.e., `2,4,8,1,` or `3`), immediately followed by a figure symbol as defined in the previous paragraph. This will offset the figure from the associated note by the specified time value. For example, if the lowest voice contained `c03 x3465`, there would be a whole-note c, and 3 quarter notes later a figure 65 below the staff.

There is also a *continuation* command, a zero followed immediately by another unsigned number. This produces a horizontal line under the bass note, starting just to the left and extending to the right by the given number of `\noteskip`s. The height and length of the line are set by the current note's level and `\noteskip` respectively. These can be mixed in with other figures to produce vertical stacks. If another figure follows in the same command, use `:` as a separator. If `\noteskip` changes or a note drops below the starting level before the line ends, it is possible to trick **PMX** by entering separate `0`[*n*] commands under each consecutive note; **PMX** will automagically join them together at the same height (thanks to Werner Icking for this idea).

If there are figured bass commands in a `.pmx` file but you want them to be ignored, then enter the command `F` at the beginning of the body of the file. This feature would most often be used in the form `%1F` (see section 3), which makes a separate bass part with no figures.

Figured bass commands will not be altered in any way under transposition. There is no universal set of interpretations of figured bass symbols, so no automatic transposition is possible.

2.6 Macros

A **PMX** macro is a single command that stands literally for any any string of characters that may occur in the input file (sorry, no variables). It may be useful if you need to repeat the same string later. There is no practical length limit.

To ***r**ecord* a macro, type `MR`*n* where *n* is between 1 and 20. Everything you then type will be processed normally as well as stored, until you enter the command `M`. The next time you need to enter the same string, just type `MP`*n* to **p**lay back the macro.

To just **s***ave* a macro without having **PMX** process it as you enter it, start it with `MS`*n*.

Macros can be redefined at will. **PMX** will print a warning whenever this occurs.

If you use macros and want to make separate parts, some care is necessary. **Scor2prt** will only transfer `MR` macros into the part where they originated, but will transfer `MS` macros into all parts.

2.7 Include Files

Include files are separate text files containing arbitrary (but contextually appropriate) sequences

of valid **PMX** commands. By using the techniques described in this section, the commands in an include file can be inserted at any desired place in the virtual **PMX** file that the code processes. They will always be syntax-checked.

There are two types of include files, *global* and *normal*. There can only be one global include file and it must be named `pmx.mod`. If activated, its contents will always be inserted right after the setup data. To activate it, two conditions must be met: (1) an environment variable `PMXMODDIR` must be defined to contain a valid path, ending with `/` or `\` ; (2) a file named `pmx.mod` must be present in the directory so defined. If `PMXMODDIR` is not set, or if it is defined but there is no file `pmx.mod`, then processing will proceed as usual.

Normal include files can have any name and do not require any environment variable to be set. They are activated by the PMX command `AR[`*filename*`]` , placed in the .pmx file at the location where the included lines are to go. It will generally only make sense to place this command at the beginning of an input block. **PMX** will first check for the file as pointed to by [*filename*], which may contain a complete or partial pathname preceding the actual file name. If [*filename*] is not found, then **PMX** will look for `%PMXMODDIR[`*filename*`]`, i.e., it will check the directory defined by `PMXMODDIR` if `PMXMODDIR` has been set. However, it is not necessary to define `PMXMODDIR` to use a normal include file. There may be any number of normal include files. The same file may be used multiple times. Include files cannot contain references to other include files via the `AR` command; if you try to do that your computer will explode. The following information regarding all activated include files will be printed both to the screen and to the .pml file: notice of opening or closing, echo of the contents, error messages pertaining to syntax errors in the included **PMX** commands, and an error message if **PMX** cannot find a referenced normal include file. In the latter two cases **PMX** will stop.

2.8 Batch Processing

Due to the number of different programs that must be run in sequence to produce a printed sheet of music with the MusiXTEX system, most users prefer to use a batch script to control the process. Since batch commands are platform-dependent we will not provide examples here, but will mention several **PMX** features that can facilitate batch processing.

First, whenever **pmxab** terminates due to a syntax error, the exit code is set to 1. There are various ways of detecting this with batch commands, then acting accordingly. Second, **pmxab** always writes a file `pmxaerr.dat` containing a single number: 0 if it exited normally, otherwise the line number in the `.pmx` file where the syntax error was. With advanced batch programming techniques, this file can be opened and read, and if there was an input error, a text editor can be opened and the input point placed on the line with the error.

There have been several requests to allow **PMX** to keep running even after it detects an input error. This has not been done because in many cases any error messages after the first one would be meaningless, or worse, uncorrected errors could cause crashes. In any event, all the output from **pmxab** will be stored in the log file [*jobname*]`.pml`.

3 Making Parts from a Score

Separate parts can be made by running **scor2prt** and entering the basename when prompted. The program will create `noinst` separate `.pmx` files, one for each instrument. By default the files will be named [*basename*][*n*]`.pmx`, where [*n*] is the sequential position of the instrument. If desired, part file names can be customized with `AN` as described in section 2.3.8.

In this section we describe how to control the layout of the parts separately from that of the score, but by using commands that are placed in the `.pmx` file for the score. This eliminates the

need for ever editing the `.pmx` files for the parts separately. You can make all corrections in the file for the score, and then re-run **scor2prt**.

Normally all lines starting with `%` in the parent `.pmx` are transferred into all the parts. However, if a line has `%%` in columns 1-2, both it *and the following line* will be ignored when making parts. If the ignored line contains only `h` , `l` , `Tc` , `Ti` , or `Tc` to start, then one additional line will be ignored.

Conversely, if a line begins with `%!` then it will be ignored as usual in creating the parent `.tex` file, but after stripping the first 2 characters the rest will be put in the `.pmx` file for *all* the parts.

To enter a line into the score file that is only to be transferred to one part, begin the line with `%`*h*, where *h* is an *extended hexadecimal digit* representing the part number from 1 to 24 (`1,2,...,9,a,b,c,...,n,o`). The first two characters will then be stripped and the rest transferred to the desired part. For example, to force a line break to system 15 and a page break to page 2 in part 11 only, enter `%bL15P2`. The use of the extended hex digits `a-o` creates a potential incompatibility with prior versions. To minimize this, the character after "`%`" will *only* be interpreted as a part number if it represents a number less than or equal to `noinst`; otherwise the entire line will be treated as an ordinary comment and transferred to all parts as a comment. 2.6

Although only permitted in the first voice in the score, the following commands with all their options will automatically be copied into all parts (unless the preceding line has `%%`): `m, V, R, A, h, w, K`. Literal TEX strings of types 2-4 will also be copied into all parts, while type 1 will only go into its original part.

User-defined hardspaces (`X` without `:`) are handled specially. By default they are not copied into parts. There are two ways to circumvent this. One way to insert hardspace *x* into part *n* is to place in the score, on a line of its own, the command `%`[*n*]`X`[*x*] . The other way is with options in the `X` command in the score: `B` causes the hardspace to be used in **b**oth score and parts; `P` puts it into the **p**art but not the score.

Instrument-wise transposition commands (see section 2.3.4) are also handled specially. When **scor2prt** encounters `Ki`[*n*] (for instrument *n*) in the score, it transfers the transposition information (transposition amount and key signature) for that instrument into the corresponding part, replacing `Ki` by `K` and keeping only the information for instrument `n`. 2.7

Lateral shifts (`X`[...]`:`) will be handled normally, staying with their original voice.

By default the total number of systems in each part will be the same as in the score. If you want to override this, there is a command `S`[*n*] (where *n* is the desired number of systems), which can only appear at the beginning of the first input block. This can be used after `%!` to affect all the parts, or after `%`[*h*] to affect just part *h*. **Scor2prt** will also compute how many pages it thinks each part should have, and enter that in the startup data for that part. If you wish to override that, then in the `.pmx` file for the score, insert for example `%3S14P2` to force the third part to have 14 systems and 2 pages (you cannot override the number of pages without first overriding the number of systems).

A musicsize of 20 is the default in all parts. This may be overridden with the option `m` in the command `S`; e.g., `%2S15m16` .

As already noted, a `P` command for page numbering in the parent file is ignored when making parts. To initiate page numbering in the parts, use for example `%!P` anywhere within the **PMX** code representing the first page of the parts (from TEX's standpoint the command must occur between the beginning and end of the page on which the numbering is to begin). It will often be useful in this case to use the option `c` , which by default causes the instrument name to be centered in small type at the top of every page after the first.

Note the distinctions among the various usages of `P`: as an option with `S`, it sets the total number of pages in a part; as an option with `L`, it forces a page break; and as a command on its own, it controls page numbering and centered headings.

MIDI commands, i.e., those starting with `I`, will never be copied into parts, unless they are in a special comment line as just described.

One function of **scor2prt** is to condense consecutive bars of rest into a single group of special printed characters with a number above it. The command `rm` defines such a **m**ulti-bar **r**est as described in section 2.2.3. **Scor2prt** will automatically insert `rm` commands into the `.pmx` files for the parts where appropriate. However, for this feature to work, the *first* full-bar rest in the sequence *must* have its duration explicitly defined in the parent `.pmx` file, either with a digit or with `p`. I.e., the feature will not work if the first rest in the sequence inherits its duration from the previous note.

Using the special **PMX** commands listed in this section, augmented where needed with literal TEX commands, it is possible to store *all* the information for both the score and the parts in a single `.pmx` file. This greatly simplifies the editing process, since both the score and the part can be corrected at once, and parts need not be re-edited each time they are regenerated from the score.

4 Making MIDI Files

PMX has an elementary capability to create MIDI files. It is intended mainly to aid in editing scores, so it does not have advanced facilities one would want for making musically satisfying sound files.

As of version 2.6, **PMX** can only generate MIDI files for scores with 15 or fewer voices. 2.6

Entering the command `I` before any notes have been entered will cause a MIDI file [*jobname*]`.mid` to be generated in the current directory. Options may follow, without spaces. They are defined in the following paragraphs. Multiple options can be combined in one `I` command. `I` commands can appear later in the file as well, but only at the start of an input block. Sometimes the order of the options matters, determining for example whether or not a user-defined pause is included inside a macro block.

`t`*x* sets the tempo to *x* quarter notes per minute. Default is 96. You can change tempos as often as you like, but only at the start of an input block (as with all MIDI commands).

`i`*i1i2...in* assigns MIDI instruments *i1,i2,...,in* to the respective **PMX** instruments. The default is harpsichord, of course. If you use this option, you must specify *all* instruments. Each *in* is either a 2-letter abbreviation or an integer between 1 and 255. Acceptable abbreviations are listed below. Numbers and pairs of letters may be mixed, but consecutive pairs of numbers must be separated by `:` (colon) . This option can only be exercised once per file. Also, the number of instruments cannot change during a piece.

The number of arguments following suboption `i`, as well as the next three described suboptions, must in fact equal the number of *instruments*. Before version 2.7, it was the number of *staves* (despite the incorrect description in the manual!) These numbers may differ and this creates a backward incompatibility. Hoping this won't cause too much distress, I've enhanced the real-time error messages. 2.7

`v`*i1*:*i2*:...:*in* assigns MIDI velocities to each instrument. The colons are required. Values may range from 1 to 127. The default is 127.

`b`*i1*:*i2*:...:*in* assigns MIDI balances to each instrument. The colons are required. Values may range from 1 to 128. The default value is 64, which represents the center. Smaller numbers favor the left stereo channel; larger ones the right.

`T` allows transposing any instrument by a selected number of steps (`\internote`s). It must be followed by exactly `noinst` signed integers representing the amount of transposition for each instrument in order. In practice it is useful in two situations (1) To transpose a MIDI output up or down by one octave (7 `\internote`s); and (2) when a transposing instrument is printed in the transposed key in the score after having issued `Ki`, to undo the transposition in the MIDI. 2.7

`M` initiates a macro operation. This is used for repeats, da capo's, etc. Macros must have ID numbers between 1 and 20. Operations are start record macro *i*: `MR`*i* ; end recording: `M` ; and playback (insert) macro *i*: `MP`*i* . Only one macro can be active at a time, recording or playing but not both. If you try nesting or overlapping macros, your computer will become psychotic.

`p`*x* inserts a pause of *x* quarter notes. Decimals are allowed, but will be rounded to the nearest sixteenth note.

`g`*i* sets the MIDI gap to *i* MIDI clock tics. This is a silence inserted at the end of every note, while decreasing the sounding duration by the same amount. The default is 10, which is 2/3 of a 64th note.

The MIDI module does not recognize graces, ornaments, repeats, voltas, or segnos. The only ties that are recognized are those using `s` or `(` alone, with no explicit ID number. Key signatures, time signatures (meter) and instrument names will be written into the MIDI file, the latter as track names. This will have no effect whatsoever on audible output but will affect on-screen appearance of some MIDI file players and editors. Location of the **PMX** key-change and meter-change commands relative to MIDI macro delimiters in the source will affect (in the obvious way) how these data are passed to such programs.

The MIDI file generator does not yet support changing the number of instruments in midstream. Doing so will cause unpredictable results.

The instruments are a subset of "The General MIDI Instrument Specification." Of course how they sound depends on your hardware and software. Instruments not listed below can still be used but must be specified by number. The numbers listed here are from the 1-128 range; when passed to the MIDI file they are reduced by one.

`pi` Acoustic Grand Piano (1)
`rh` Rhodes Piano (5)
`ha` Harpsichord (7)
`ct` Clavinet (8)
`ma` Marimba (13)
`or` Church Organ (20)
`gu` Acoustic Nylon Guitar (25)
`ab` Acoustic Bass (33)
`vl` Violin (41)
`va` Viola (42)
`vc` Cello (43)
`cb` Contrabass (44)
`vo` Synth Voice (55)
`tr` Trumpet (57)
`tb` Trombone (58)
`tu` Tuba (59)
`fr` French Horn (61)
`so` Soprano Sax (65)
`al` Alto Sax (66)
`te` Tenor Sax (67)
`bs` Baritone Sax (68)
`ob` Oboe (69)
`ba` Bassoon (71)
`cl` Clarinet (72)
`fl` Flute (74)
`re` Recorder (75)

5 Limits

For simplicity in writing the program, **PMX** has numerous variables with fixed dimensions. In most cases there are no checks against these limits (hey, I've got more important things to program), so occasionally there may be hangups due to exceeding a dimension. Any of these can potentially be increased by making a request via the mailing list. However, before making such a request, try working around the problem by breaking the input into smaller blocks.

5.1 Limits on quantities that a user can control

(The user can control the *number* of these items, but cannot control the *limit on the maximum number* of them.)

128 characters per input line.
24 staves. 2.6
2 voices per staff.
24 voices per system. 2.6
125 systems.

600 bars.
40 forced line breaks.
10 forced page breaks.
18 key changes.
75 pages. 2.78
600 notes per input block. 2.6
15 bars per input block.
101 slurs per input block.
74 figures (figured bass) per input block.
37 grace note groups per input block.
74 notes in grace note groups per input block.
52 literal TEX strings per input block.
6 voltas per input block.
24 trills per input block. 2.6
62 chordal notes (non-spacing) per input block.
8 beams per voice per bar.
40 forced beams per voice per input block.
10 clef changes per voice per input block.
24 notes per beam.
24 notes per xtuplet.
41 text-dynamic strings per input block.
9600 lines in input file 2.78

5.2 Limits not under immediate user control

131072 bytes in the entire input file 2.78
20 `\notes` groups per bar.
20 inserted standard anti-collision spaces (not xtuplet or end-of-bar) per bar.
20 inserted anti-collision spaces within xtuplets per bar.
19 inserted anti-collision end-of-bar hardspaces per system.
83 inserted anti-collision end-of-bar hardspaces.
400 inserted standard anti-collision spaces per system.
100 inserted anti-collision spaces within xtuplets per system.
1000 inserted standard anti-collision spaces.
200 inserted anti-collision spaces within xtuplets.
24576 bytes of MIDI output data per voice.

6 Closing Notes

6.1 About the Example Files

`most.pmx` contains examples of most of the **PMX** commands, and a few programming tricks, including examples in the last line of beam groups whose notes vary widely in pitch. The printed output displays the **PMX** commands near to the resulting typeset characters. It is more useful to look at the printed output rather than the source file, since the file is littered with in-line TEX needed to output the text strings representing the **PMX** commands. **WARNING:** Do not try to play this music; it could be hazardous.

`barsant.pmx` contains the first movement of a recorder sonata by the Italian Francesco Barsanti (1690-1772). It demonstrates many of **PMX**'s strong points in a "battlefield" situation: figured bass, complex beaming patterns, xtuplets, and automatically adjusted horizontal and vertical spacing in crowded scores. In fact, this single-page score pushes the limits of vertical and

horizontal crowding. To get the final result, it makes subtle adjustments using various available options: `Ae` for equal space between systems, `AI1.1` to increase the vertical space between staves in a system, `Apl` to activate postscript slurs and special treatment of line-breaking slurs/ties (note slur at end of fourth system), and `W.5` to increase minimum space between noteheads so the 64th notes don't touch each other. This is also a good score to try making parts with **scor2prt**. A special command `%2S9` is used to increase the number of systems in the recorder part.

`mwalmnd.pmx` is an Allemand for harpsichord by the German Matthias Weckmann (1616-1674). It uses many techniques peculiar to keyboard scores, most notably two voices per staff.

`netsoos.pmx` is an example with lyrics, including several inline TEX commands to enhance the layout. 2.73

`staffcrossall.pmx` contains examples of staff-crossing chords. Some are single-stemmed, some are beamed non-xtuplets, and finally beamed xtuplets. 2.74

6.2 A Benign Bug

When TEX'ing the output of **PMX** you will usually get an `Underfull \vbox` message at the end of each page. This is due to my using `\eject` at the end of every page, which automatically spaces the systems vertically without having to fiddle with `\staffbotmarg`. As far as I know, the warning is benign, and may be ignored.

6.3 Where to Get Help

The main home of **PMX** on the internet is the software section of the Werner Icking Music Archive. This site also links to a mailing list devoted to MusiXTEX and related programs including **PMX** . The denizens of this list are always willing to answer questions about any aspect of the software. New users are strongly advised to take advantage of this resource.

6.4 Acknowledgments

To Daniel Taupin, Ross Mitchell, and Andreas Egler for creating MusiXTEX; to Olivier Clary for suggesting a crucial modification in the note-entry scheme; to my colleague John DiPol (a non-musician!) for the idea of using binary masks to define beam groupings; to Joel Hunsberger for unraveling some deep MusiXTEX tangles; to Dirk Laurie for making **PMX** accessible to vocal music by creating `\pmxlyr` and **M-Tx**; to Stanislav Kneifl and Hiroaki Morimoto for developing the postscript slur packages; to Christian Mondrup, Andre Van Ryckeghem, Christof Biebricher, Joerg Anders, Olivier Vogel, and other denizens of the TeX-music mailing list for first-class bug-finding and support in responding to queries about **PMX** on the mailing list; to Luigi Cataldi, Olivier Vogel, Christof Biebricher, and Cornelius Noack for producing translated and enhanced **PMX** tutorials; and to Bob Tennent for maintaining the software section of the web site. Finally, I want to mention again the invaluable contributions by Werner Icking: his exhaustive beta testing, uncanny bug-finding, continuing encouragement, and promotion of **PMX** right up until his sudden and premature departure from this earthly realm.