MaxScore – music notation for Max/MSP

Created by Nick Didkovsky with Georg Hajdu

This manual was updated Friday, April 25, 2008

Introduction

Thank-you for downloading MaxScore, the music notation object for Max/MSP. MaxScore was written in Java and uses Java Music Specification Language (JMSL) as its notation and playback engine, but requires no Java programming to use. While MaxScore is free, it requires a JMSL license which can be obtained at www.algomusic.com. MaxScore was commissioned by "Bipolar - German-Hungarian Cultural Projects." Bipolar is an initiative of the Federal Cultural Foundation of Germany.

What is MaxScore?

MaxScore provides you with common music notation directly in the Max/MSP environment. MaxScore is a Max object which accepts messages that can create a score, add notes to it, perform it, save it, load it, and export the score to popular formats for professional publishable results. MaxScore currently exports to MusicXML so you can load your scores into Finale. MaxScore also exports to the GNU LilyPond automated engraving system.

MaxScore is more than a notation tool. It is an interactive performance object. MaxScore can play back a score and drive your MSP patches through a well-defined instrument interface. Scores can be created and modified in real-time. You can add notes explicitly by defining their properties (specifying a quarter note triplet as duration and middle C as pitch), or generate an arbitrary stream of musical events and use MaxScore's Transcriber to notate them automatically.

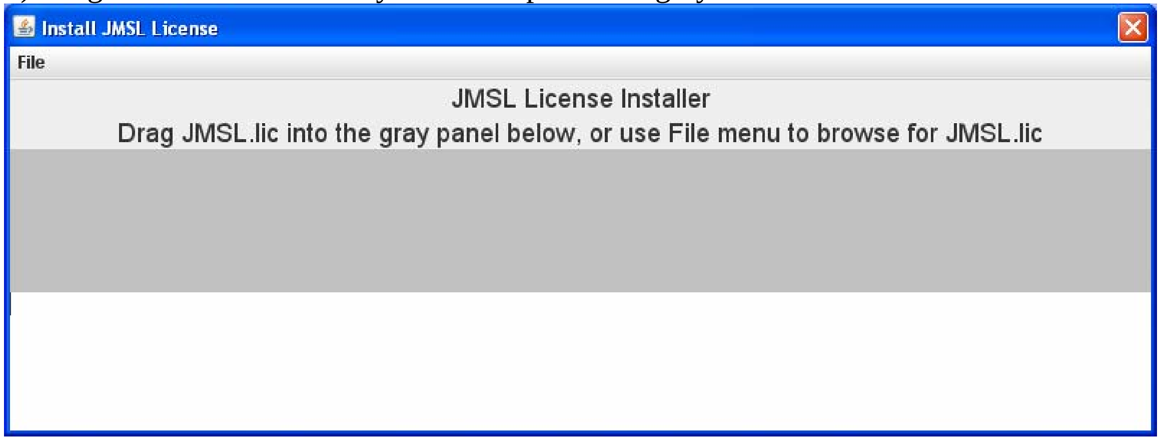
MaxScore was programmed in Java Music Specification Language by Nick Didkovsky but requires no Java programming to operate. While MaxScore is freely available to the public, it requires a JMSL license to run, available at www.algomusic.com

Installation

MaxScore comes bundled with four “jar” files, a Max helpfile, and auxiliary files. Run the installer appropriate for your platform. Be sure to choose your MaxMSP installation directory when using the Windows installer. The Mac installer will look for the folder “MaxMSP 4.6”

Licensing

1) First request a JMSL license at
http://www.algomusic.com/algomusiclicense/request_jmsl_demo_license.html
2) Your license will arrive by email a few seconds after you request it. Save the license file that is attached to the email to your desktop, named *JMSL.lic*.
3) Then double click on JMSL_License_installer.jar . A window like the one shown below will open.
4) Drag the license file from your desktop into the gray area in the installer window.

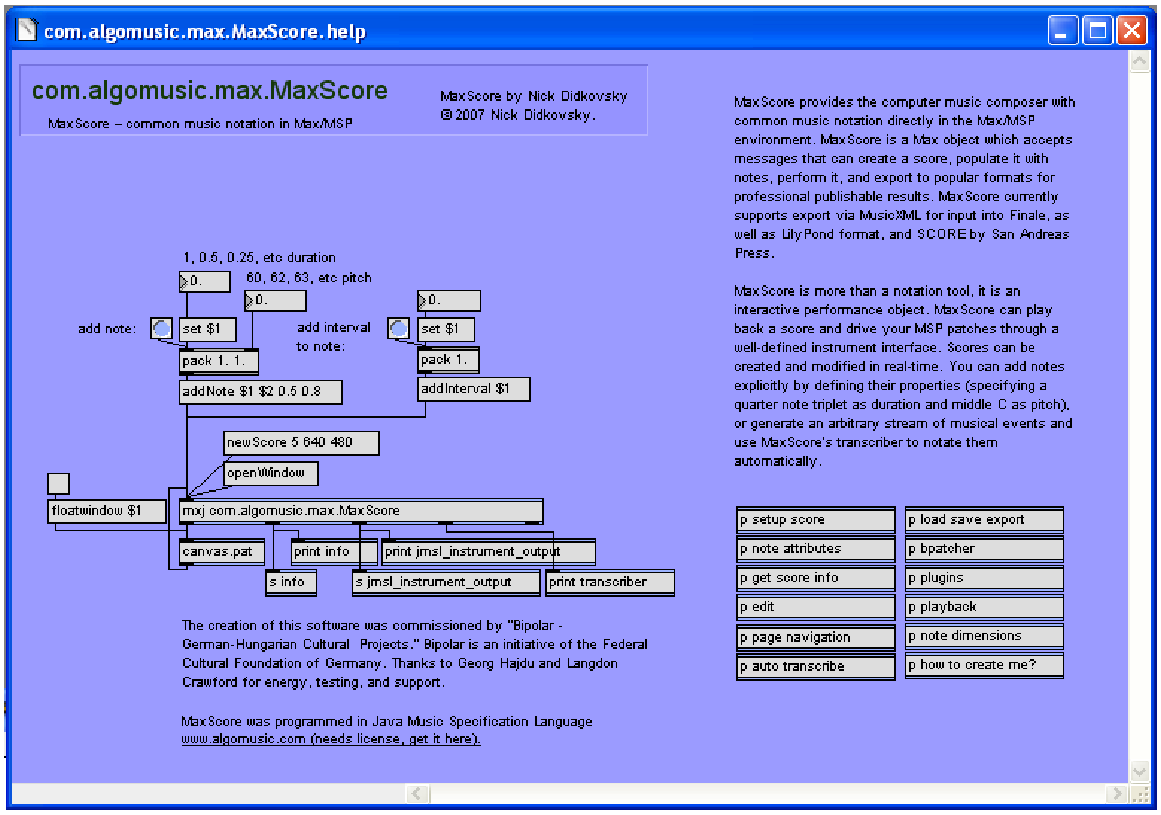


JMSL License Installer
Drag JMSL.lic into the gray panel below, or use File menu to browse for JMSL.lic

The JMSL license installer. Drag the file named JMSL.lic to the gray area.

Quick start

Let’s make sure your copy of JMSL is licensed, that the jars are in the right place, and that Max/MSP can use MaxScore.
1) Launch Max/MSP as you normally do. Open a new window (File -> New->Patcher) and create a new empty object box. Type “maxscore” (no quotes) and type esc before activating the object. Alternatively, choose “mxj com.algomusic.max.MaxScore” from the New Object menu or File -> Open the file called com.algomusic.max.MaxScore.help. For more information on creating new objects, please refer to Max46Fundamentals.pdf.



MaxScore provides the computer music composer with common music notation directly in the Max/MSP environment. MaxScore is a Max object which accepts messages that can create a score, populate it with notes, perform it, and export to popular formats for professional publishable results. MaxScore currently supports export via MusicXML for input into Finale, as well as LilyPond format, and SCORE by San Andreas Press.

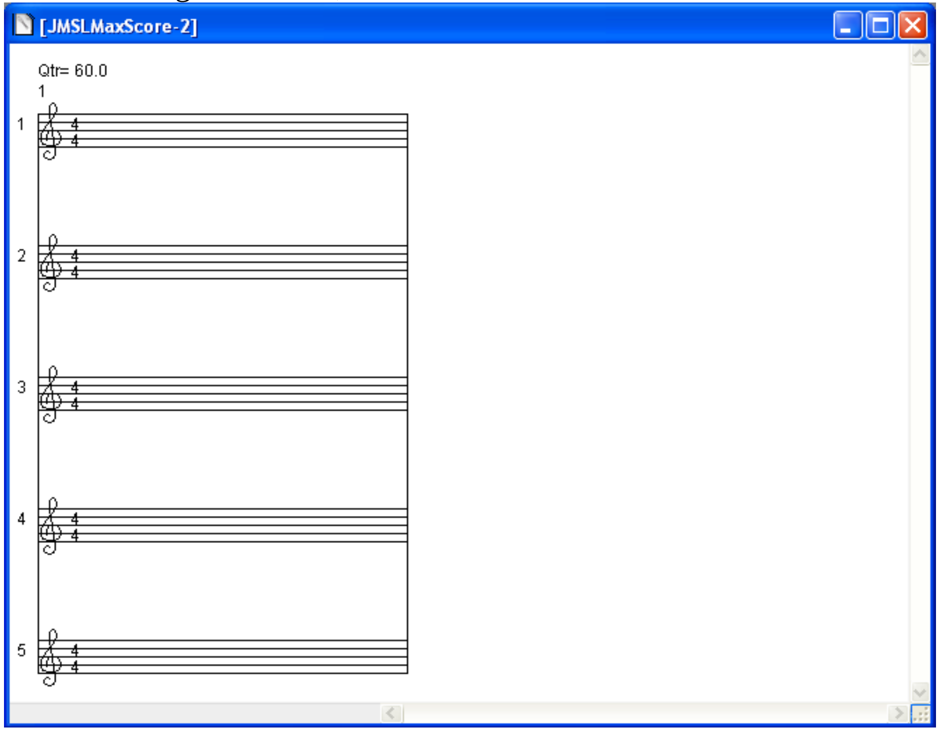
MaxScore is more than a notation tool, it is an interactive performance object. MaxScore can play back a score and drive your MSP patches through a well-defined instrument interface. Scores can be created and modified in real-time. You can add notes explicitly by defining their properties (specifying a quarter note triplet as duration and middle C as pitch), or generate an arbitrary stream of musical events and use MaxScore's transcriber to notate them automatically.

The creation of this software was commissioned by "Bipolar - German-Hungarian Cultural Projects." Bipolar is an initiative of the Federal Cultural Foundation of Germany. Thanks to Georg Hajdu and Langdon Crawford for energy, testing, and support.

MaxScore was programmed in Java Music Specification Language www.algomusic.com (needs license, get it here).

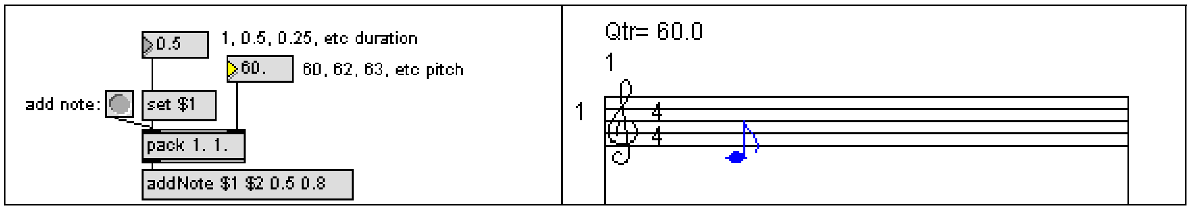
MaxScore's Setup Score help patch

2) Click on the message box that reads "NewScore 5 640 480". A new score will open that has width and height 640x480, with five staves.



New MaxScore with five staves

3) Add Notes
Note entry using the mouse: Right click anywhere in the staff to add a couple of notes. Click-drag a rectangle around these notes. Tap the letter “c” on your computer keyboard to copy these notes. Click anywhere on an empty staff. Tap the letter “v” on your computer keyboard to paste them.
Note entry using the addNote message: In the number boxes (see screen shot below), enter 0.5 for an eighth note duration, and 60 for pitch in the number boxes. Bang addNote. Middle C 8th note will appear in the score.



MaxScore’s addNote message (left) and result (right)

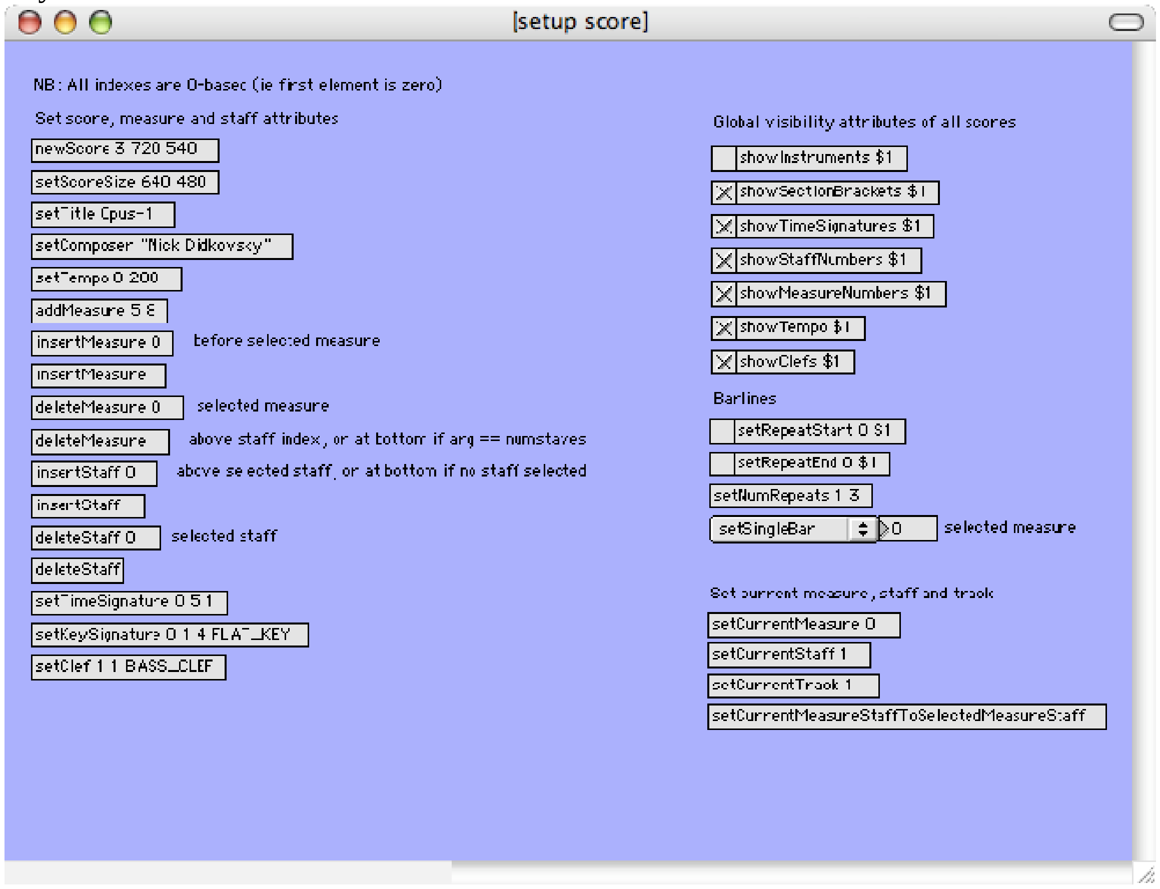
4) That’s it***! If you want to use MaxScore in your own patches make sure you connect it to canvas.pat as is shown in the help file***

Catalog of MaxScore Messages

Here we present a catalog of messages you can send to MaxScore to create, perform, export, and save your scores. The catalog is broken down by the help files.

p setup Score

This window contains MaxScore messages that control various properties and appearance of your score.



[setup score]

NB: All indexes are 0-based (ie first element is zero)

Set score, measure and staff attributes

```
newScore 3 720 540
setScoreSize 640 480
setTitle Opus-1
setComposer "Nick Didkovsky"
setTempo 0 200
addMeasure 5 8
insertMeasure 0        before selected measure
insertMeasure
deleteMeasure 0        selected measure
deleteMeasure          above staff index, or at bottom if arg == numstaves
insertStaff 0          above selected staff, or at bottom if no staff selected
insertStaff
deleteStaff 0          selected staff
deleteStaff
setTimeSignature 0 5 1
setKeySignature 0 1 4 FLAT_KEY
setClef 1 1 BASS_CLEF
```

Global visibility attributes of all scores

```
☐ showInstruments $1
☒ showSectionBrackets $1
☒ showTimeSignatures $1
☒ showStaffNumbers $1
☒ showMeasureNumbers $1
☒ showTempo $1
☒ showClefs $1
```

Barlines

```
☐ setRepeatStart 0 $1
☐ setRepeatEnd 0 $1
setNumRepeats 1 3
setSingleBar  0   selected measure
```

Set current measure, staff and track

```
setCurrentMeasure 0
setCurrentStaff 1
setCurrentTrack 1
setCurrentMeasureStaffToSelectedMeasureStaff
```

newScore

args:

1. int numStaves
2. int width
3. int height

Creates a new MaxScore. You should send the openWindow message if a new window does not open

setScoreSize

args:

- int width
- int height

Resizes score.

setTitle
args:
- string title

Sets the title of your score. Title is included in exported MusicXML and LilyPond formats.

setComposer
args:
- string composer's name

Setting the composer of the current score will be reflected in MusicXMLand LilyPond output.

setTempo
args:
1. int measure index
2. int beats per minute

Sets the tempo of the indicated measure to the indicated beats per minute. Measures added after this measure will inherit this tempo. To change the tempo of a range of measures, iterate through them and send this message for each

addMeasure
args:
1. int time signature numerator
2. int time signature denominator

Adds an empty measure with the specified time signature to the end of the current score

insertMeasure
args:
- none

Inserts an empty measure before the currently selected measure

insertMeasure
args:
- int measure index

Inserts an empty measure before the indicated measure. Measure index is 0 based (ie measure 0 is the first measure). New measure inherits the properties of the measure before it. So if you insert a measure before measure 1 the new measure will take on the time signature and tempo of measure 0

deleteMeasure
args:

- none

Deletes the currently selected measure**.**

deleteMeasure

args:
- int measure index

Deletes the indicated measure. Measure index is 0 based (ie measure 0 is the first measure).

insertStaff

args: none
Inserts a new staff above selected staff index, or at the bottom of the system if no staff is selected.

insertStaff

args:
- int staff index

Inserts a new staff above indicated staff index, or at the bottom of the system if staffindex equals number of staves. Staff index starts with zero being the top staff.

deleteStaff

args:
- none

Deletes the currently selected staff in every measure of score**.**

deleteStaff

args:
- int staff index

Deletes indicated staff of every measure on score. Staff index starts with zero being the top staff.

setTimeSignature

args:
1. int measure index
2. int time signature numerator
3. int time signature denominator

Sets the time signature of the indicated measure to the indicated time signature. Measure index is 0-based (first measure is 0).

setKeySignature

args:
1. int measure index
2. int staff index

3. int numaccidentals
4. string FLAT_KEY | SHARP_KEY

Sets the key signature of the speficied staff in the specified measure. Key signature is specified by the number of accidentals, and whether the accidentals are sharps or flats (for example the key of D major or B minor would be specified as 2 SHARP_KEY)

setClef

args:

1. int measure index
2. int staff index

string clef type (TREBLE_CLEF, ALTO_CLEF, TENOR_CLEF, BASS_CLEF)
Set the clef for the indicated staff of the indicated measure.
Measure index is 0 based (first measure is measure 0)
Staff index is 0 based (top staff is 0)

showInstruments

args:

- boolean true | false

Show or hide the names of the instruments normally printed above the staff.

showSectionBrackets

args:

- boolean true | false

Show or hide the << >> brackets which are printed above measures indicating a section. A section is a subrange of measures which can be performed using section play

showTimeSignatures

args:

- boolean true | false

Show or hide the time signatures normally printed on each staff.

showStaffNumbers

args:

- boolean true | false

Show or hide the staff numbers normally printed on each staff.

showMeasureNumbers

args:

- boolean true | false

Show or hide the measure numbers normally printed above the first measure of each page.

showTempo

args:

- boolean true | false

Show or hide the tempo normally printed above each measure

showClefs

args:

- boolean true | false

Show or hide the clefs normally printed on each staff.

setRepeatStart

args:

1. int measure index
2. boolean true | false

For the indicated measure set or unser repeat start ||:
Measure index is 0 based (first measure is measure 0)

setRepeatEnd

args:

1. int measure Index
2. boolean true | false

For the indicated measure set or unset repeat end :||
Measure index is 0 based (first measure is measure 0)

setNumRepeats

args:

1. int measure index
2. int num repeats

Set the number of repeats for the specified measure. Only makes sense if the measure has repeat end
Measure index is 0 based (first measure is measure 0)

setSingleBar

args:

- int measure index

Set the barline of the indicated measure to a single bar
Measure index is 0 based (first measure is measure 0)

setDoubleBar

args:

- int measure index

Set the barline of the indicated measure to a double bar
Measure index is 0 based (first measure is measure 0)

setPeriodDoubleBar

args:

- int measure index

Set the barline of the indicated measure to a period double bar (one thin one thick)

Measure index is 0 based (first measure is measure 0)

setBarNone

args:

- int measure index

Make the barline of the indicated measure invisible
Measure index is 0 based (first measure is measure 0)

setCurrentMeasure

args:

- int measure index

Sets the current measure as specified by the value passed to it. Note that the first measure is measure 0, the second measure is measure 1, etc. The current measure, current staff, and current track is where the addNote message will put its next note.

setCurrentStaff

args:

- int staff index

Sets the current staff as specified by the value passed to it. Note that the top staff is 0, the second staff is 1, etc. The current measure, current staff, and current track is where the addNote message will put its next note.

setCurrentTrack

args:

- int track index

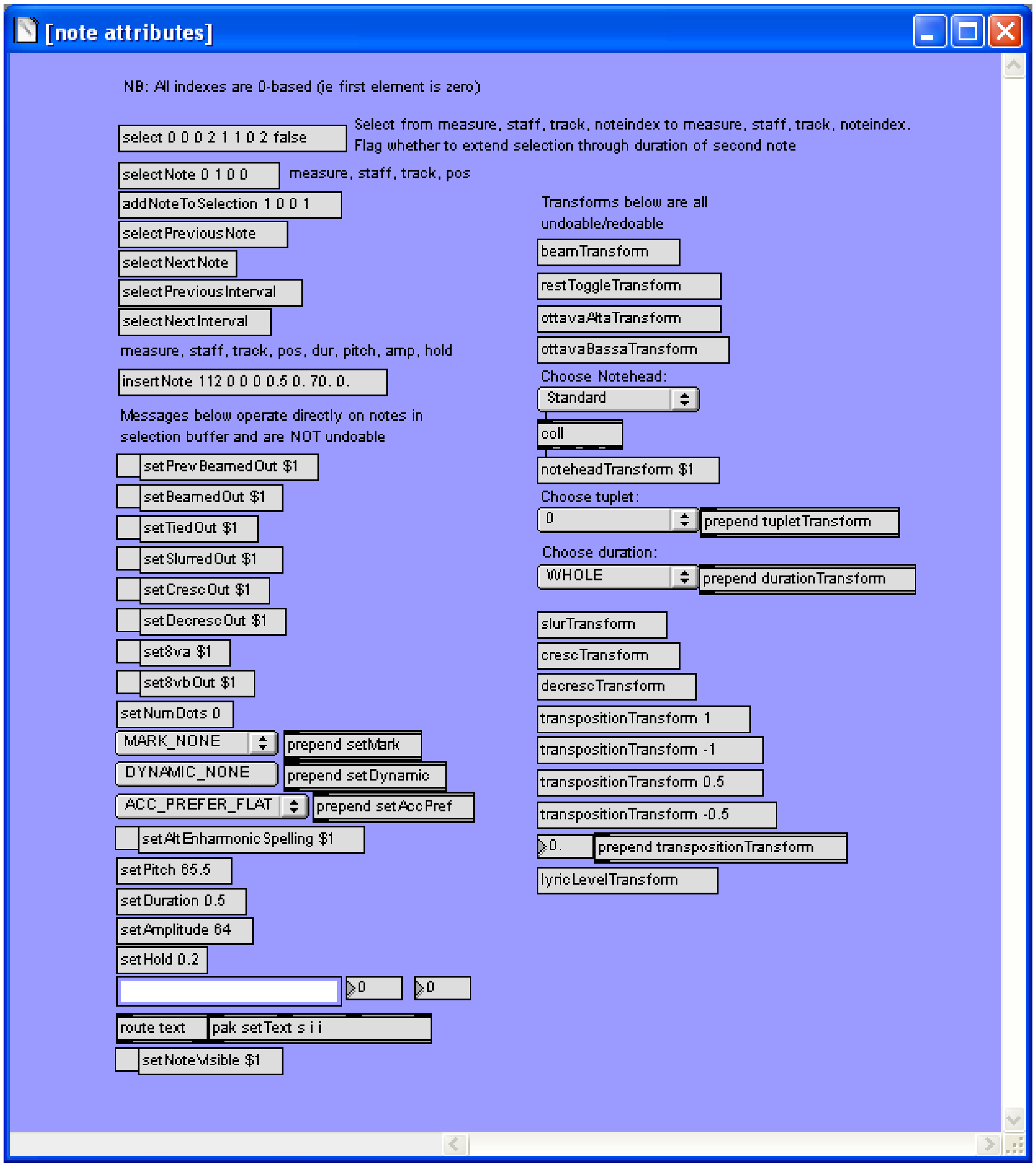
Sets the current track as specified by the value passed to it. By default, every staff has two tracks, 0 and 1. Think of track 0 as the default track for one-part music, and think of track 1 as the second part (for example, put hihat notes for drumset in track 1 and its stems will go up; put kick and snare in track 0 and their stems will go down). The current measure, current staff, and current track is where the addNote message will put its next note.

setCurrentMeasureStaffToSelectedMeasureStaff

args: none
The Selected measure and staff is where the user clicks with the mouse. Remember the addNote message adds notes the the end of the current track of the current staff of the current measure. This is different from the Selected measure, staff, and track. This message provides you with a short cut to click anywhere in the score and make that the current location.

p note attributes

This window contains maxScore messages that allows you to select notes and apply operations to them. Notes can be selected by hand clicking on a note, or by dragging a rectangle around notes, or by clicking on a note and shift-clicking on another note to select all notes between them.



[note attributes]

NB: All indexes are 0-based (ie first element is zero)

select 0 0 0 2 1 1 0 2 false

Select from measure, staff, track, noteindex to measure, staff, track, noteindex.
Flag whether to extend selection through duration of second note

selectNote 0 1 0 0

measure, staff, track, pos

addNoteToSelection 1 0 0 1

selectPreviousNote

selectNextNote

selectPreviousInterval

selectNextInterval

measure, staff, track, pos, dur, pitch, amp, hold

insertNote 112 0 0 0 0.5 0. 70. 0.

Messages below operate directly on notes in
selection buffer and are NOT undoable

setPrevBeamedOut $1

setBeamedOut $1

setTiedOut $1

setSlurredOut $1

setCrescOut $1

setDecrescOut $1

set8va $1

set8vbOut $1

setNumDots 0

MARK_NONE | prepend setMark

DYNAMIC_NONE | prepend setDynamic

ACC_PREFER_FLAT | prepend setAccPref

setAltEnharmonicSpelling $1

setPitch 65.5

setDuration 0.5

setAmplitude 64

setHold 0.2

0 | 0

route text | pak setText s i i

setNoteVisible $1

Transforms below are all
undoable/redoable

beamTransform

restToggleTransform

ottavaAltaTransform

ottavaBassaTransform

Choose Notehead:

Standard

coll

noteheadTransform $1

Choose tuplet:

0 | prepend tupletTransform

Choose duration:

WHOLE | prepend durationTransform

slurTransform

crescTransform

decrescTransform

transpositionTransform 1

transpositionTransform -1

transpositionTransform 0.5

transpositionTransform -0.5

0. | prepend transpositionTransform

lyricLevelTransform

select

int: from measure index
int: from staff index
int: from track index
int: from note index
int: to measure index
int: to staff index
int: to track index
int: to note index
boolean spanToEndOfDuration

Selects notes between and including the “from note” and the “to note”. If flag is true, the selection will be extended through the full duration of the “to note”. So for example if the “from note” is in the top staff which contains numerous quarter notes, and the “to note” is a whole note in the second staff, setting this flag to true will include all the quarter notes in the top staff that span the whole note. Setting this flag to false will cause the selection to end with the whole note and the quarter note in the staff immediately in line with it.

selectNote
args:

1. int: measure
2. int: staff
3. int: track
4. int: pos

Selects note in specified measure, staff, track, and position. For example, passing args 0, 1, 0, 0 would select the first note in measure 1 of second staff.

addNoteToSelection
args:

5. int: measure
6. int: staff
7. int: track
8. int: pos

Adds specified note to selection buffer. See selectNote for args example. This differs from selectNote in that addNoteToSelection adds the specified note to the selection buffer, while selectNote first clears the buffer then adds the specified note.

selectNextNote
args: none
Assuming a note is currently selected, this command deselects it and selects the next note in staff. If there are no more notes in the current staff, you will see the message “current note has no next note” in the Max window.

selectPreviousNote
args: none
Assuming a note is currently selected, this command deselects it and selects the previous note in staff. If there are no notes preceding the note in the current staff (ie it is the first note of measure 1, for example), you will see the message “current note has no previous note” in the Max window.

selectPreviousInterval
args: none
Assuming a note is currently selected and it is part of a chord, this command deselects it and selects the next note below it in the chord. If there are no more notes in the current chord, you will see the message “root, no previous interval” in the Max window.

selectNextInterval
args: none
Assuming a note is currently selected and it is part of a chord, this command deselects it and selects the next note above it in the chord. If there are no more notes above it in the current chord, you will see the message “Top of chord, no next interval” in the Max window.

insertNote
args:

1. int: measure
2. int: staff
3. int: track
4. int: pos
5. float dur
6. float pitch
7. float amp
8. float hold

Inserts a note before the indicated location. For example if the first four args are 0, 0, 0, 0, the new note will show up as the first position of the top staff of the first measure. The last four arguments, follow standard convention where dur 1.0 is a quarter note, 0.5 is an eighth note, 0.333 is a triplet, etc. Pitch 60 is middle C. Fractional pitches like 60.5 are ok. Amp is usually 0..1 unless you are using MIDI. Hold is sustain time in seconds (this is different than duration. It defined staccato versus legato performance for example; literally how long the note is “held”).

setPrevBeamedOut
args:
Boolean true|false

Assuming a note is selected, if true, beams previous note to selected note. If false, unbeams.

setBeamedOut
args:
	Boolean true|false

Assuming a note is selected, if true, beams selected note to next note. If false, unbeams.

setTiedOut
args:
	Boolean true|false

Assuming a note is selected and the next note has the same pitch as the selected note, if arg is true, ties selected note to next note. If false, unties.

setSlurredOut
args:
	Boolean true|false

Assuming a note is selected, if arg is true, slurs selected note to next note. If false, unslurs.

setCrescOut
args:
	Boolean true|false

Assuming a note is selected, if arg is true, makes crescendo symbol from selected note to next note. If false, removes cresc hairpin. This is a notation only, it does not affect the amplitudes of the selected notes.

setDecrescOut
args:
	Boolean true|false

Assuming a note is selected, if arg is true, makes decrescendo symbol from selected note to next note. If false, removes decresc hairpin. This is a notation only, it does not affect the amplitudes of the selected notes.

set8va
args:
	Boolean true|false

Assuming a note or notes are selected, if arg is true, puts ottava alta symbol over selected notes. If false, removes ottava alta symbol. This will cause the instrument to receive a pitch 12 steps higher than written.

set8vb

args:

Boolean true|false

Assuming a note or notes are selected, if arg is true, puts ottava bassa symbol over selected notes. If false, removes ottava bassa symbol. This will cause the instrument to receive a pitch 12 steps lower than written.

setNumDots
args:

int numDots

Assuming a note or notes are selected, set the number of dots on this note to 0, 1, or 2. Affects playback duration.

setMark
args:

String, one of:
MARK_NONE
MARK_ACCENT
MARK_STACCATO
MARK_TENUTO
MARK_WEDGE
MARK_ACCENT_STACCATO
MARK_ACCENT_TENUTO
MARK_WEDGE_STACCATO
MARK_FERMATA
MARK_HARMONIC
MARK_TRILL,
MARK_TRILL_FLAT
MARK_TRILL_SHARP
MARK_TRILL_NATURAL
MARK_MORDANT
MARK_INVERTED_MORDANT
MARK_FERMATA

Applies indicated mark to selected notes.

setMark
args:

int code, an integer which is one of:
MARK_NONE = 0
MARK_ACCENT = 1
MARK_STACCATO = 2
MARK_TENUTO = 3
MARK_WEDGE = 4
MARK_ACCENT_STACCATO = 5

MARK_ACCENT_TENUTO = 6
MARK_WEDGE_STACCATO = 7
MARK_FERMATA = 8
MARK_HARMONIC = 9
MARK_TRILL = 10
MARK_TRILL_FLAT = 11
MARK_TRILL_SHARP = 12
MARK_TRILL_NATURAL = 13
MARK_MORDANT = 14
MARK_INVERTED_MORDANT = 15

Applies indicated mark to selected notes.

setDynamic
args:
String, one of:
DYNAMIC_NONE
DYNAMIC_PPP
DYNAMIC_PP
DYNAMIC_P
DYNAMIC_MP
DYNAMIC_MF
DYNAMIC_F
DYNAMIC_FF
DYNAMIC_FFF

Applies indicated dynamic to selected notes.

setAccPref
args:
String, one of:
ACC_PREFER_FLAT
ACC_PREFER_SHARP

Set the preference that all notes in Selection Buffer should use to spell accidentals. This does not actually make the note sharp or flat. It tells the note which to choose in the event it needs to be spelled with an accidental (for example if C is transposed up a half step, should it be spelled with C# or Db).

setAltEnharmonicSpelling
args:
boolean true|false
Determines if notes in selection buffer should use alternate enharmonic spelling (ex E#, F##, Cb, Fbb, etc).

setPitch

args:

float pitch

Assign indicated pitch to all notes in selection buffer. Pitch can be fractional such as 60.123 (123 cents above middle C)

setDuration

args:

float duration

Assign indicated duration to all selected notes. Whole note = 4.0, half note = 2.0, quarter note = 0.5, eighth note=0.25, quarter note triplet = 0.333, 8th note quintuplet = 0.2, etc

setAmplitude

args:

float amp

Assign indicated amplitude to all selected notes. No restrictions on units. Yo might use 0..127 for MIDI or 0..1 for patches. Interpretation depends entirely on the instrument which will play this note.

setHold

args:

float holdTime

Specifies how long a note sustains. This is different from duration. A quarter note (duration 1.0) might have a very short staccato hold time of 0.1, or a long hold time greater than 1.0. holdTime does not interfere with rhythm. It is simply how long a note sounds once it is started.

setText

args:

string text
int xOffset
int yOffset

Sets specified text near selected notes. X and Y offset specify how far away from the note the text appears.

setNoteVisible

args:

boolean true|false

Sets selected notes visible or invisible.

beamTransform

args: none

Toggles Beam Transform on selected notes. Transforms are undoable/redoable.

restToggleTransform

args: none

Toggles Rest Transform on selected notes. Transforms are undoable/redoable.

ottavaAltaTransform

args: none

Toggles 8va on selected notes. Transforms are undoable/redoable.

ottavaBassaTransform

args: none

Toggles 8vb on selected notes. Transforms are undoable/redoable.

noteheadTransform

args:
string, one of:
NOTEHEAD_STANDARD
NOTEHEAD_X
NOTEHEAD_DIAMOND
NOTEHEAD_TRIANGLE
NOTEHEAD_INVERTED_TRIANGLE
NOTEHEAD_X_DIAMOND
NOTEHEAD_SLASH

Applies specified notehead to selected notes.

tupletTransform

args:
int, one of: 0, 3, 5, 7, 9, 11, 13, 17, 19

Applies tuplet to selected notes, where 0 removes tuplet and restores note to its binary value.

durationTransform

args:
string coreduration, which is one of:
WHOLE
HALF
QUARTER
EIGHTH
SIXTEENTH
THIRTYSECOND
SIXTYFOURTH
ONEHUNDREDTWENTYEIGHTH
TWOHUNDREDFIFTYSIXTH

Change the core duration of a note. This does not affect whether it is a tuplet or has dots. Just changes the “core duration”.

slurTransform

args: none
Toggles slur applied to selected notes.

crescTransform
args: none
Toggles cresc applied to selected note. This differs from setCrescOut in that the crescTransform terminates the cresc on the final selected note.

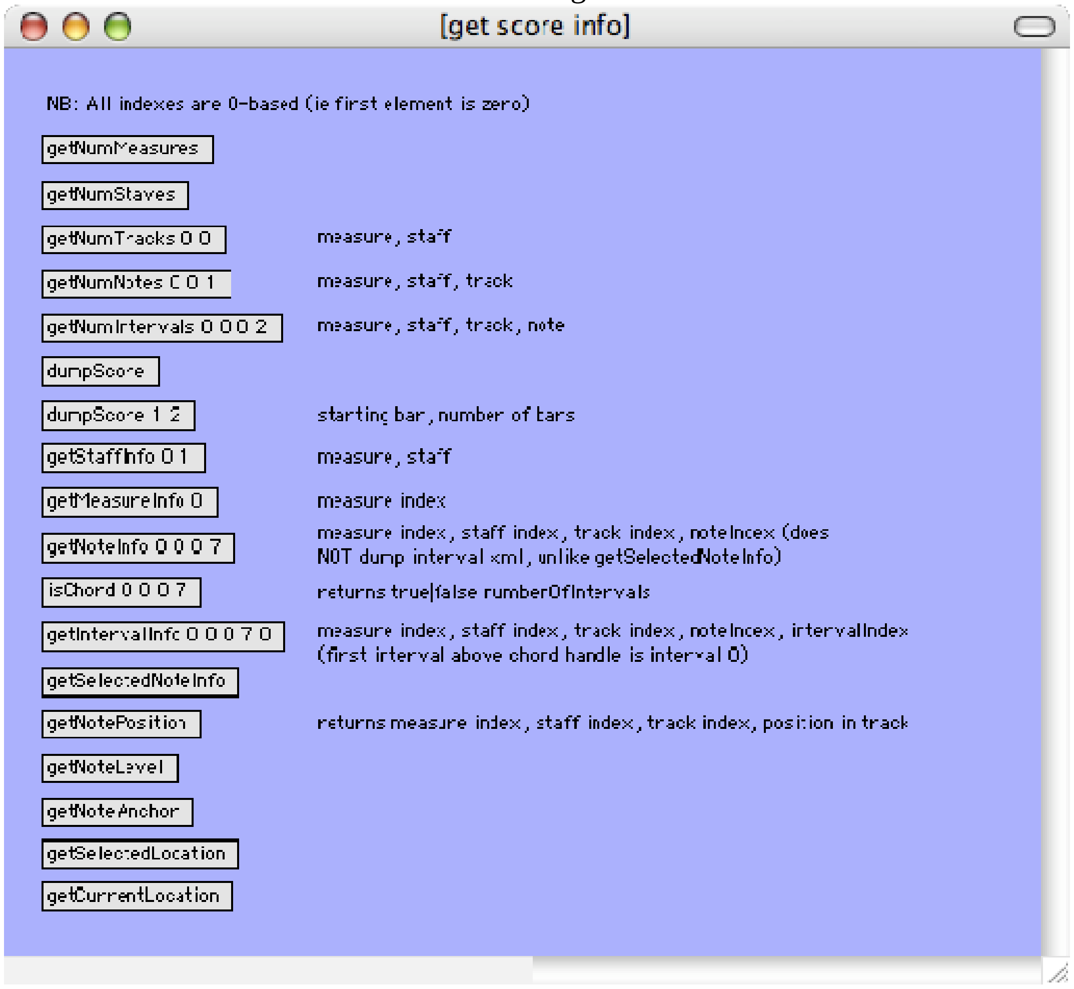
decrescTransform
args: none
Toggles decresc applied to selected note. This differs from setDecrescOut in that the decrescTransform terminates the decresc hairpin on the final selected note.

transpositionTransform
args:
float steps
Transposes selected note by indicated amount. 12 transposes up an octave. -12 transposes down an octave.

lyricLevelTransform
args: none
Applies a transform which sets the x,y offsets of text of each selected note to values which are below the note and below the staff, so all are vertically at the same level. If a note is too low below the staff, the text will be pushed down below it and will not align with other text.

p get score info

This window contains maxScore messages that allows you to query your score for various attributes and information. All info gets sent out the second outlet.



[get score info]

NB: All indexes are 0-based (ie first element is zero)

getNumMeasures	
getNumStaves	
getNumTracks 0 0	measure, staff
getNumNotes 0 0 1	measure, staff, track
getNumIntervals 0 0 0 2	measure, staff, track, note
dumpScore	
dumpScore 1 2	starting bar, number of bars
getStaffInfo 0 1	measure, staff
getMeasureInfo 0	measure index
getNoteInfo 0 0 0 7	measure index, staff index, track index, noteIndex (does NOT dump interval xml, unlike getSelectedNoteInfo)
isChord 0 0 0 7	returns true|false numberOfIntervals
getIntervalInfo 0 0 0 7 0	measure index, staff index, track index, noteIndex, intervalIndex (first interval above chord handle is interval 0)
getSelectedNoteInfo	
getNotePosition	returns measure index, staff index, track index, position in track
getNoteLevel	
getNoteAnchor	
getSelectedLocation	
getCurrentLocation	

getNumMeasures

args: none
Sends the number of measures out the second outlet.

getNumStaves

args: none
sends the number of staves out the second outlet.

getNumTracks

args:

1. int measure index
2. int staff index

Sends the number of tracks of the indicated staff of the indicated measure out the second outlet.
Measure index is 0 based (first measure is measure 0)

Staff index is 0 based (top staff is 0)

getNumNotes

args:

1. int measure index
2. int staff index
3. int track index

Sends the number of notes contained in the indicated track of the indicated staff of the indicated measure out the second outlet
Measure index is 0 based (first measure is measure 0)
Staff index is 0 based (top staff is 0)
Track index is 0 based (first track is 0)

getNumIntervals

args:

1. int measure index
2. int staff index
3. int track index
4. int note index

A chord is represented as a Note (called a handle of the chord) with some number of intervals. A triad is a note with two intervals, for example.
This message sends the number of intervals contained in the indicated note of the indicated track of the indicated staff of the indicated measure out the second outlet
Measure index is 0 based (first measure is measure 0)
Staff index is 0 based (top staff is 0)
Track index is 0 based (first track is 0)
Note index is 0 based (first note is 0)

dumpScore

args: none
Dump xml note list (with all its attributes) from Score to the Max environment. Format is XML with all measure and note attributes specified. Same as JMSL Score's file format (ie this is JMSL's own XML format, **not** MusicXML)

dumpScore

args:

1. int start measure index
2. int num measures

Dump xml listing about specified measure range in XML format. Note that indexes are zero-based (ie first measure is 0), so getMeasureInfo(1, 3) would dump info for the second, third, and fourth measures.

getStaffInfo

args:

1. int measure index
2. int staff index

Sends staff properties out second outlet in XML format.
Measure index is 0 based (first measure is measure 0)
Staff index is 0 based (top staff is 0)

getMeasureInfo

args:

- int measure index

Sends measure properties out second outlet in XML format. This is in JMSL's own XML format (not MusicXML)
Measure index is 0 based (first measure is measure 0)

getNoteInfo

args:

- int measure index
- int staff index
- int track index
- int note index

Sends xml note description to second outlet. Does not include interval info if it is a chord.

isChord

args:

- int measure index
- int staff index
- int track index
- int note index

Is specified note a chord? Sends true|false numIntervals to second outlet

getIntervalInfo

args:

- int measure index
- int staff index
- int track index
- int note index

Sends XML info for specified interval in chord out second outlet. 0 gives first interval itself (not the chord root). So in a two note chord, getNoteInfo gives you the chord handle while passing interval 0 to getIntervalInfo gives you the second note.

getSelectedNoteInfo

args: none
Dump note info for each note in selection buffer. Follows JMSL Score's native XML format. If a Note is a handle to a chord it will print all its intervals' xml info as well.

getNotePosition

args: none
Get position of selected note(s): measure index, staff index, track index, position of note

in track. All indexes are zero based so first measure is 0, top staff is 0, first note is 0. For example, first note in top staff of first measure returns 0 0 0 0. Sends out second outlet.

getNoteLevel

args: none
Sends vertical level of selected notes out second outlet

getNoteAnchor

args: none
Sends drawing anchor (x, y) of selected notes out second outlet. x and y are relative to (0, 0) at top left of score canvas

getSelectedLocation

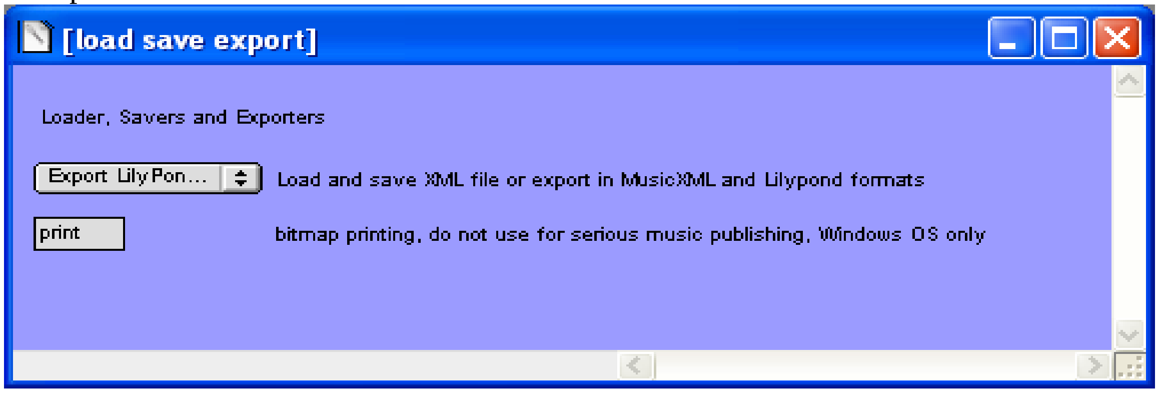
args: none
Get the measure index and staff index of score's selected measure and selected staff. This is where the user clicked. Sends -1 -1 if there is no currently selected measure and staff. Sends out second outlet.

getCurrentLocation

args: none
Get the measure index and staff index of score's current measure and current staff. This is where the next addNote() will go. Sends out second outlet.

p load save export

This window contains MaxScore messages that allows you to load and save score as well as export to external music notation formats.



Loader, Savers and Exporters

Load and save XML file or export in MusicXML and Lilypond formats

bitmap printing, do not use for serious music publishing, Windows OS only

loadScore

args:

- string filename

Load a score from a file in JMSL's own XML format (not MusicXML)
Specify full path and filename, quote if path contains spaces.

saveScore

args:

- string filename

Save current score in XML format (JMSL's own XML format, not MusicXML). Specify full path and filename. If you specify a filename that ends with .zip then your XML score will zip compressed, which MaxScore can also load back in with loadScore (as long as .zip is specified in the filename of course). Zipping an XML file compresses it greatly because there is so much redundancy in XML

saveMusicXML

args:

- string filename

Save current score in Recordare's MusicXML format. You can load MusicXML into Finale and Sibelius for professional quality music publishing.

saveLilyPond

args:

- string filename

Save current score in LilyPond format for professional quality music publishing. For more information about LilyPond, visit www.lilypond.org

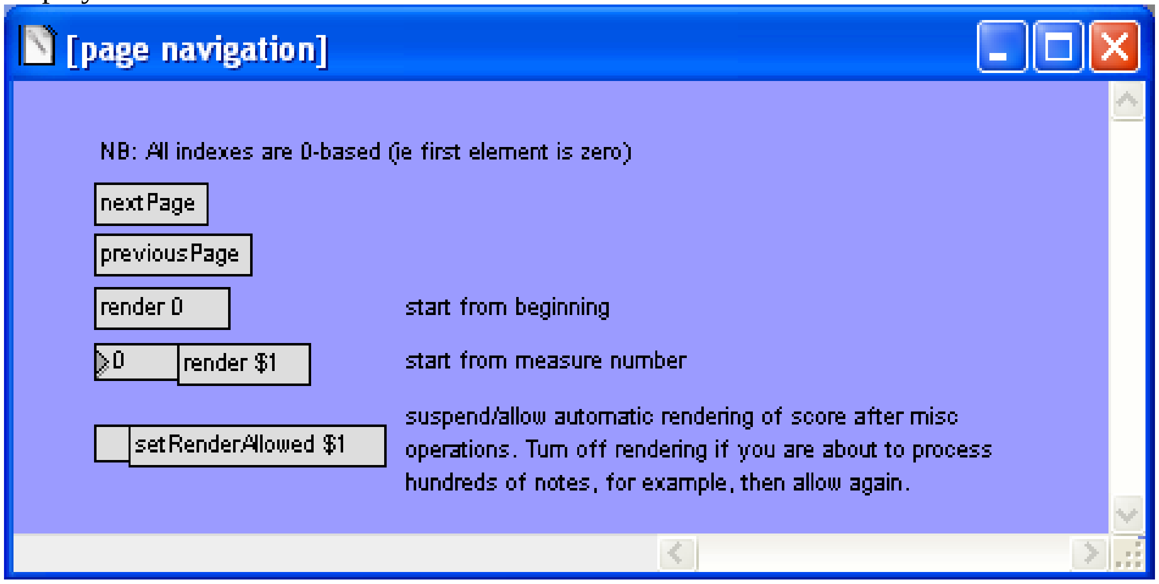
print

args: none

Prints a bitmap image of score to printer. Works under Windows only.

p page navigation

This window contains MaxScore messages that allows you to navigate through the score (turn pages, etc). Pages are not absolute locations in MaxScore. You can scroll through a score on e measure at a time, and an entire page will be laid out with that measure as the first one on the page. Rendering and layout is based on the first measure requested to be displayed.



NB: All indexes are 0-based (ie first element is zero)

nextPage

previousPage

render 0 — start from beginning

0 render $1 — start from measure number

setRenderAllowed $1 — suspend/allow automatic rendering of score after misc operations. Turn off rendering if you are about to process hundreds of notes, for example, then allow again.

nextPage

Renders a page of the current score starting at the next unrendered measure. For example if the current page displays measures 100..200, the nextPage message will display a new page starting with measure 201, containing as many measures as will fit on that page.

previousPage

Renders a new page whose last measure is the one preceding the first measure of the current page. For example, if the current page displays measures 100..200, the previousPage message will display a new page ending with measure 99. The first measure of that new page depends on how many measure fit on that page.

render

args:

- int measure index

Layout and render a page starting with specified measure. Measure index is 0-based (ie first measure is 0)

setRenderAllowed

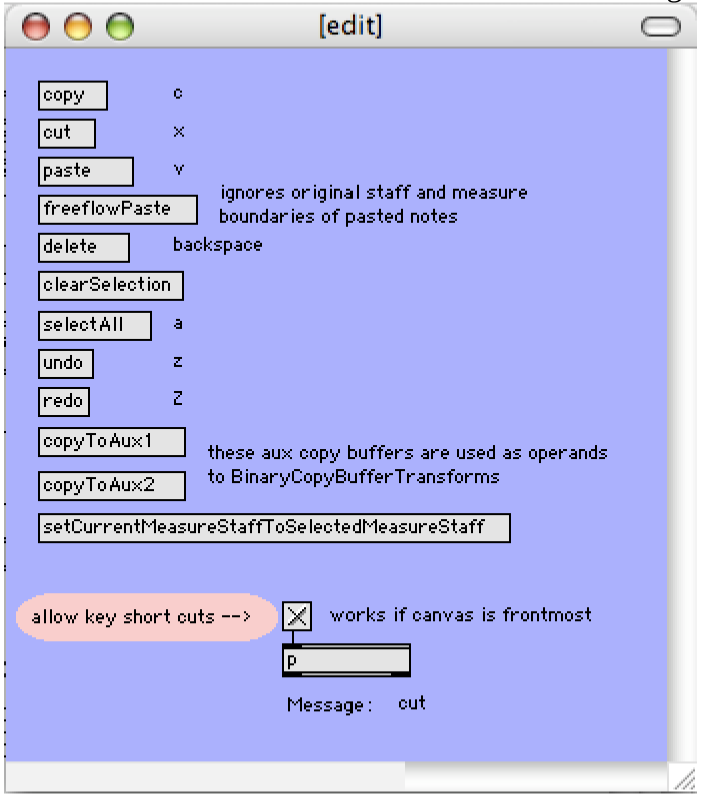
args:

- Boolean true| false

Enables or disables internal render commands which occur automatically at the end of many MaxScore messages. Disable if you intend to perform operations on hundreds of notes, for example. Then enable when you are done with your operations and call render to see the new state of your score.

p edit

This window contains MaxScore messages that copy and paste notes in a variety of flavors. This window also demonstrates how to assign keystrokes to actions.



ignores original staff and measure boundaries of pasted notes

these aux copy buffers are used as operands to BinaryCopyBufferTransforms

works if canvas is frontmost

copy

args: none
Copy notes in selection buffer to copy buffer. Note that UnaryCopyBufferTransforms operate on notes found in this buffer

cut

args: none
Copy notes in selection buffer to copy buffer and remove them from score

paste

args: none
Paste notes from copy buffer into selected location of score. The Selected location is the measure and staff where the user clicks with the mouse. Relative measure and staff boundaries of the copied notes are maintained after pasting.

freeflowPaste

args: none
Paste notes from copy buffer into selected location of score. The Selected location is the measure and staff where the user clicks with the mouse. Copied notes that were originally in different measures and staves will be pasted as a contiguous stream without preserving original measure and staff boundaries.

delete

args: none
Deletes selected notes. Does not put them in copy buffer.

clearSelection

args: none
Removes all selected notes from the selection buffer (is deselects all notes).

selectAll

args: none
Selects all notes in score.

copyToAux1

args: none
Copy notes in selection buffer to auxiliary copy buffer 1. BinaryCopyBufferTransforms operate on notes found in this buffer and in aux buffer 2, using notes contained in these two copy buffers as operands.

copyToAux2

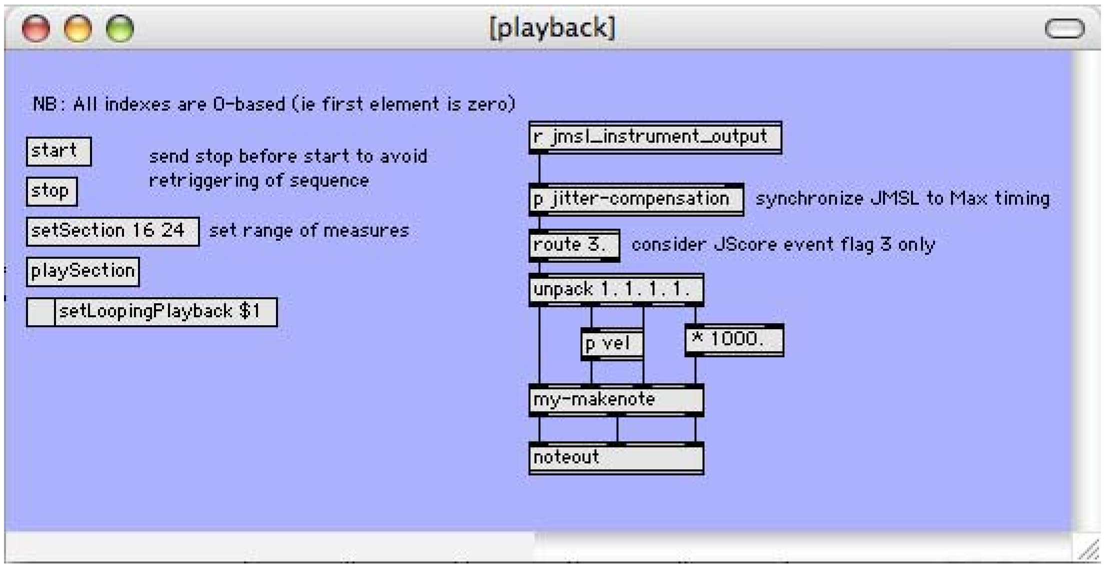
args: none
Copy notes in selection buffer to auxiliary copy buffer 1. BinaryCopyBufferTransforms operate on notes found in this buffer and in aux buffer 2, using notes contained in these two copy buffers as operands.

setCurrentMeasureStaffToSelectedMeasureStaff

args: none
The Selected measure and staff is where the user clicks with the mouse. Remember the addNote message adds a note to the end of the *current* measure, staff, and track. This is different from the s*elected* measure, staff, and track. This message provides you with a short cut to click anywhere in the score and make that *selected* location the *current* location.

p playback

This window contains MaxScore messages that control playback of a Score. *For n important discussion of the "jmsl_instrument_output" portion of this window, see the next section entitled "note dimensions".*



start

args: none
Begins score playback starting with the first measure.

stop

args: none
Finishes score playback.

setSection

args:

- int start measure index
- int end measure index

A section is a subrange of measures of a score which can be played back independently. Measure index is 0-based (ie measure 0 is the first measure). << >> will appear over the first and last measures of a section.

playSection

args: none
Beings section playback (a section is a subrange of measures specified with the setSection message)

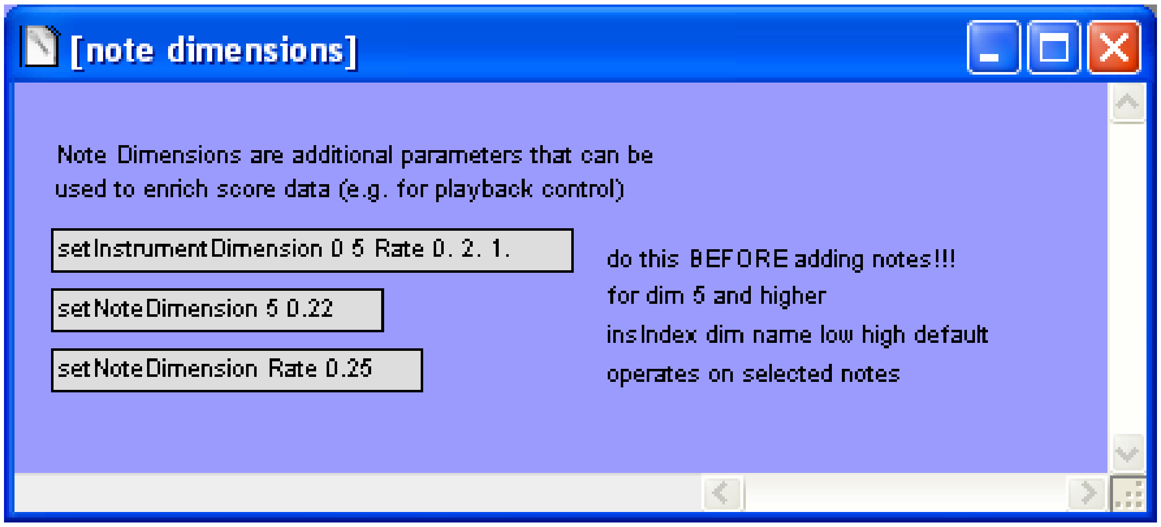
setLoopingPlayback

args:

- Boolean true | false

Sets playback behavior to loop or not. When playing a section, the section will looping if this is true. When playing back a score, the entire score will loop if this is true.

p note dimensions



Note Dimensions are additional parameters that can be used to enrich score data (e.g. for playback control)

```
setInstrumentDimension 0 5 Rate 0. 2. 1.
setNoteDimension 5 0.22
setNoteDimension Rate 0.25
```

do this BEFORE adding notes!!!
for dim 5 and higher
insIndex dim name low high default
operates on selected notes

A Note contains an array of floating point numbers, or "dimensions". Dimensions are indexed 0, 1, 2, 3… The meaning of first 5 dimensions is fixed as follows:
0 – duration
1 – pitch
2 – amplitude
3 – hold time
4 - sustain/play flag

Dimensions numbered 5 and higher are user-definable and could be used to control custom inlets of your MSP patches, for example.

When a Note is played, it passes its data to the MaxScoreInstrument assigned to its staff. This MaxScoreInstrument in turn passes the data to the main MaxScore object, which massages it and sends its third outlet (which in the Help File is connected to "s jmsl_instrument_output"). The first value of the array passed to the outlet is the instrument index. Each instrument has a unique index, so you can inspect this value and thereafter route the rest of its data appropriately, since you know which staff the data came from.

Below is the **format of the Note data as it is passed out of MaxScore's third outlet** as an array of floats.

0)	instrument index
1)	timestamp in MaxClock time (the 'on-time' of this note)
2)	pitch
3)	amplitude
4)	hold time in seconds (ie sustain time ie Max "duration")
5)	event flag (describes state of this event, see discussion below)
6...n)	any additional dimensions

Some comments:
Note that the **timestamp** will be a precise value a little in the future. This is useful to absorb wiggle of the underlying scheduler which is implemented using Java threads. See the “jitter compensation” object in the “p playback” window for an example on how to utilize this feature

The **event flag** specifies the state of the note event. It is not set by the user, it is set at runtime by MaxScoreInstrument. The event flag can be one of four values, as is shown below:

MAX_INS_OFF = 0;
MAX_INS_ON = 1;
MAX_INS_UPDATE = 2;
MAX_INS_PLAY = 3;

The “update” and “play” values are most important. When a note is played, this flag will be set to 3. When a note is tied in, however, this value will be 2 (ie it is being updated). What does updated mean? You might for example have dimension 6 assigned to a filter cutoff frequency. Imagine you have two quarter notes, with the first tied to the second. Sine they are tied, each has the same pitch but they might have different amplitudes (dimension 3) and different cutoff frequencies (dimension 6). The first note is played, and its event flag will be 3 (max_ins_play). When the next, tied-in note is played, it will send the event flag 2 (max_ins_update). This will notify you not to re-attack your MSP patch, but only to update its amplitude and cutoff frequency inlets. Notice that the first quarter note will send a hold time (ie sustain time) for the full duration of itself plus the durations of the tied-in note after it (in ths case a value of 1.8 would be reasonable, which is 1.0 for the first quarter plus 0.8 for the second quarter, holding its tone just a little short of the full duration).

setInstrumentDimension
args:

int instrument index
int dimension
String dimension name
float low limit
float high limit
float default

Each staff of a score is assigned its own MaxScoreInstrument. This message lets you set the dimensions of a MaxScoreInstrument. Call this message to set up your dimensions before adding notes to the staff. For example, passing the following args…

```
0 5 Rate 0. 2. 1.
```

…is interpreted as, “Set the 5th dimension of instrument 0 to be named ‘Rate’. Set its low limit to 0.0, its high limit to 2.0 and its default value to 1.0”. From then on, every note added to the staff containing this instrument will conform to this dimension name space. Read the discussion above for more information on how such extra dimensions can be used to drive your custom MSP patches.

setNoteDimension
args:

int dimension index
float value

Set the value of the specified dimension of the selected notes. Passing 5 0.22 for example would set the 5th dimension of all selected notes to 0.22. This assumes you have first set up the instrument dimensions how you want them (see setInstrumentDimension above)

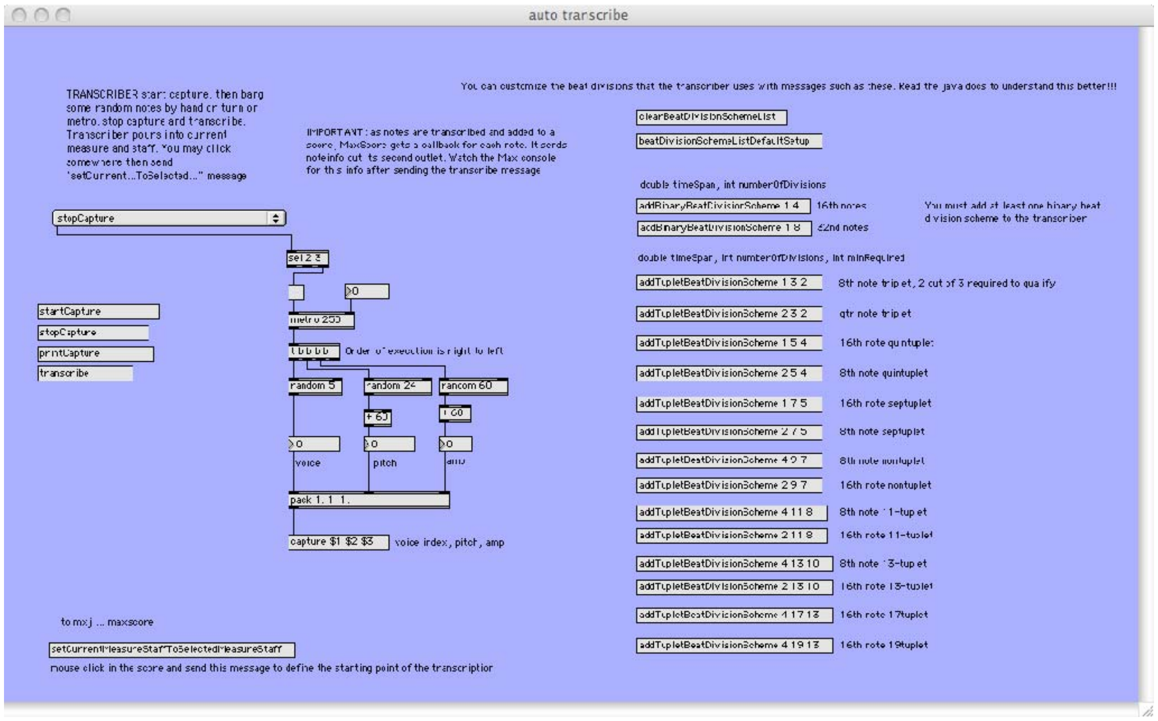
setNoteDimension
args:

String dimension name
float value

Set the value of the specified dimension of the selected notes. Passing Rate 0.22 for example would set the dimension named “Rate” to 0.22 for all selected notes. Same as setNoteDimension above except you can refer to the dimension by name inseatd of by index.

p auto transcribe

The “auto transcribe” window shows how to use MaxScore’s transcriber to capture a real-time stream of musical events and transcribe them to common music notation. Use this if you have a Max patch that generates musical material and you want to notate it. The duration between events does not have to be quantized or massaged to conform to standard durations. MaxScore’s transcriber will look for the best fit to assign standard durations to your input events.



TRANSCRIBER start capture. then bang some random notes by hand or turn on metro. stop capture and transcribe. Transcriber pours into current measure and staff. You may click somewhere then send "setCurrent...ToSelected..." message

IMPORTANT: as notes are transcribed and added to a score, MaxScore gets a callback for each note. It sends noteinfo out its second outlet. Watch the Max console for this info after sending the transcribe message

Order of execution is right to left

capture $1 $2 $3 voice index, pitch, amp

to mxj ... maxscore

setCurrentMeasureStaffToSelectedMeasureStaff

mouse click in the score and send this message to define the starting point of the transcription

You can customize the beat divisions that the transcriber uses with messages such as these. Read the java docs to understand this better!!!

clearBeatDivisionSchemeList

beatDivisionSchemeListDefaultSetup

double timeSpan, int numberOfDivisions

addBinaryBeatDivisionScheme 1 4 16th notes

addBinaryBeatDivisionScheme 1 8 32nd notes

You must add at least one binary beat division scheme to the transcriber

double timeSpan, int numberOfDivisions, int minRequired

addTupletBeatDivisionScheme 1 3 2 8th note triplet, 2 out of 3 required to qualify

addTupletBeatDivisionScheme 2 3 2 qtr note triplet

addTupletBeatDivisionScheme 1 5 4 16th note quintuplet

addTupletBeatDivisionScheme 2 5 4 8th note quintuplet

addTupletBeatDivisionScheme 1 7 5 16th note septuplet

addTupletBeatDivisionScheme 2 7 5 8th note septuplet

addTupletBeatDivisionScheme 4 9 7 8th note nontuplet

addTupletBeatDivisionScheme 2 9 7 16th note nontuplet

addTupletBeatDivisionScheme 4 11 8 8th note 11-tuplet

addTupletBeatDivisionScheme 2 11 8 16th note 11-tuplet

addTupletBeatDivisionScheme 4 13 10 8th note 13-tuplet

addTupletBeatDivisionScheme 2 13 10 16th note 13-tuplet

addTupletBeatDivisionScheme 4 17 13 16th note 17tuplet

addTupletBeatDivisionScheme 4 19 13 16th note 19tuplet

The auto transcribe window above shows a simple patch that generates a sequence of note events that the transcriber can capture. The metro 250 object fires 4 times per second (16th notes), but you may change that during capture. The metro triggers three numbers very time it plays: a voice index which assigns a note event to a staff, a pitch within two octaves of pitch 60 (middle C), and a random MIDI-style velocity between 60 and 120. It packs these three numbers and sends them to MaxScore’s “capture” method. When finished, the user sends the transcribe method to convert these captured events into common music notation.

startCapture
args: none
MaxScore begins capturing events that arrive in its “capture” inlet.

stopCapture
args: none
MaxScore will stop capturing events that arrive in its “capture” inlet.

printCapture
args: none
Prints captured events to the Max window.

transcribe
args: none
MaxScore will process all captured events and send common music notation into the current active score. Transcription will begin in the currently selected measure and staff. You may set the current measure and staff by clicking there with the mouse and then sending the setCurrentMeasureStaffToSelectedMeasureStaff message. Alternatively, you may set this programmatically by sending setCurrentMeasure and setCurrentStaff messages.

Acknowledgements

Nick Didkovsky would like to thank Georg Hajdu for driving this project, and Langdon Crawford for his enthusiasm, expertise, and support.