Anton Bruckner Privatuniversität
Institut für Dirigieren, Komposition, Musiktheorie
PMA Master Instrumental(Gesangs)pädagogik Arrangement und Komposition
(UniAkkG/S09)

Masterarbeit

Die Partitur auf Knopfdruck?

Computergestützte algorithmisch notierte Komposition
mit
MAX und LilyPond

Michael Enzenhofer

Matrikel Nr.: 12BU246

März 27, 2015

1. Gutachter: Ao.Univ.Prof. Mag. Andreas Weixler
2. Gutachter: Ao.Univ.Prof. Sven Birch

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Masterarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Abstract

In vorliegender Arbeit werden ungeblendet von schnellen Lösungen der Zeit einerseits ein paar Schritte zurück gewagt, um das Thema der Notation wieder ins zentrale Blickfeld der computergestützten Komposition zu rücken, andererseits werden ein paar Schritte nach vorne unternommen, und neue mögliche Anwendungen aufgezeigt, beziehungsweise anhand von konkreten Vorschlägen in die Wege geleitet.

Es werden wesentliche Programm-Bausteine zur computergestützten algorithmischen Komposition sowie neue Errungenschaften zur Notation in *Echtzeit* innerhalb der MAX-Programmierumgebung vorgestellt.

Erstmals in der Geschichte der algorithmischen Komposition werden die Programme MAX und LilyPond zusammengeführt, indem *Patches* zum Erzeugen von für LilyPond verständlichem Code programmiert und offen dargelegt werden.

Des Weiteren werden nachvollziehbare Lösungen vorgestellt, wie notwendige Softwarebestandteile des Rechners zur Zusammenarbeit gebracht werden und wie damit die „Partitur auf Knopfdruck“ mit größtmöglicher Flexibilität und ohne Einschränkungen von MAX aus möglich wird.

Inhaltsverzeichnis

1 Einleitung

Der Arbeitstitel zu vorliegender Arbeit war:
„*Über die Notation in der computergestützten algorithmischen Komposition*“
Es liegt „auf der Hand“, dass jeder der Begriffe

- Notation
- Computer
- Algorithmus
- Komposition

für sich eine ganze (Staats-)Bibliothek füllen könnte.

ALGORITHMUS	KOMPOSITION	NOTATION	COMPUTER
ALGORITHMUS	KOMPOSITION	COMPUTER	NOTATION
ALGORITHMUS	NOTATION	KOMPOSITION	COMPUTER
ALGORITHMUS	NOTATION	COMPUTER	KOMPOSITION
ALGORITHMUS	COMPUTER	KOMPOSITION	NOTATION
ALGORITHMUS	COMPUTER	NOTATION	KOMPOSITION
KOMPOSITION	ALGORITHMUS	NOTATION	COMPUTER
KOMPOSITION	ALGORITHMUS	COMPUTER	NOTATION
KOMPOSITION	NOTATION	ALGORITHMUS	COMPUTER
KOMPOSITION	NOTATION	COMPUTER	ALGORITHMUS
KOMPOSITION	COMPUTER	ALGORITHMUS	NOTATION
KOMPOSITION	COMPUTER	NOTATION	ALGORITHMUS
NOTATION	ALGORITHMUS	KOMPOSITION	COMPUTER
NOTATION	ALGORITHMUS	COMPUTER	KOMPOSITION
NOTATION	KOMPOSITION	ALGORITHMUS	COMPUTER
NOTATION	KOMPOSITION	COMPUTER	ALGORITHMUS
NOTATION	COMPUTER	ALGORITHMUS	KOMPOSITION
NOTATION	COMPUTER	KOMPOSITION	ALGORITHMUS
COMPUTER	ALGORITHMUS	KOMPOSITION	NOTATION
COMPUTER	ALGORITHMUS	NOTATION	KOMPOSITION
COMPUTER	KOMPOSITION	ALGORITHMUS	NOTATION
COMPUTER	KOMPOSITION	NOTATION	ALGORITHMUS
COMPUTER	NOTATION	ALGORITHMUS	KOMPOSITION
COMPUTER	NOTATION	KOMPOSITION	ALGORITHMUS

Ich wüsste auch nicht, welche Reihung stimmiger wäre.
Jedes der vier Elemente hat einen nicht wegzudenkenden Platz im Kontext.

Von vornherein war mir klar, dass der Fokus auf *Notation* liegen wird.

Um musikalische Inhalte zu *notieren* muss nicht gleich *komponiert* werden, und um zu *komponieren* muss nicht unbedingt alles *notiert* werden.

Dennoch kann eine *Notation* schon *Komposition* sein, obwohl sie noch nicht erklingt. Es kann aber eine *Komposition* erklingen, obwohl sie nicht *notiert* ist.

Hier läuft man ganz schön Gefahr, sich zu verirren...

Bei meinen Ausführungen gehe ich davon aus, dass beim *Komponieren* auch *notiert* wird. Und der *Computer* vollbringt nichts beziehungsweise kaum etwas,

was nicht auch von Hand erledigt werden könnte.

Da in meinen Ausführungen *Notation* nicht nur mit dem *Computer* erledigt wird, sondern vorrangig auch mit dem *Computer komponiert* wird – indem er *notiert*, habe ich mich für den Untertitel:
„*Computergestützte algorithmisch notierte Komposition*“
entschieden.

„*Die Partitur auf Knopfdruck*“ ist das eigentliche Ziel hinter der gewählten Thematik, worüber zu debattieren und zu philosophieren gleichermaßen interessant wäre – deswegen das Fragezeichen nach dem Titel –, ich aber „handfeste“ Vorschläge dazu unterbreiten möchte.

Diese Vorschläge beruhen auf konkreten Software-Bausteinen, die uns als *Computer*-Anwender im Musikbereich zur Verfügung stehen.

MAX sowie LilyPond leisten uns dabei beste Dienste.
Besonders, indem ihre Kräfte vereint werden.

2 Begriffsbestimmungen und Unterscheidungen

Oder: Worüber wir (nicht) reden.

Algorithmus

Algorithmus[1] bedeutet nach Duden ein:

> „Verfahren zur schrittweisen Umformung von Zeichenreihen;
> Rechenvorgang nach einem bestimmten [sich wiederholenden] Schema".

Oder nach Wiktionary[2]

> „eine exakt beschriebene Vorgehensweise zum Lösen eines Problems".

Über die Herkunft und Definition von *Algorithmus* wird viel spekuliert,[3] jedenfalls ist der erste für einen Computer gedachte *Algorithmus* 1843 von Ada Lovelace (1815–1852) zur Berechnung von Bernoullizahlen festgehalten.[4]
Ihr *Algorithmus* ging als erstes Computerprogramm der Welt in die Geschichte ein.[5]

[1] mittellateinisch algorismus = Art der indischen Rechenkunst.
[2] siehe: http://de.wiktionary.org/wiki/Algorithmus (August 2014).
[3] Vergleiche: Gerhard Nierhaus: *Algorithmic Composition – Paradigms of Automated Music Generation*. SpringerWienNewYork (Wien 2009) S. 1–3.
[4] Vergleiche: http://de.wikipedia.org/wiki/Algorithmus (August 2014).
[5] http://www.welt.de/wissenschaft/article111915874/Die-Frau-die-das-erste-Computerprogramm-schrieb.html (August 2014).

Algorithmen wurden im Mathematik-, Computer- und Informatikbereich eingesetzt, lange bevor sie um 1950 in der musikalischen Begrifflichkeit auftauchten.[1]

Es wird zwischen:
deterministischem und *randomisiertem Algorithmus* unterschieden:

Deterministischer Algorithmus

> „Ein Algorithmus ist deterministisch, wenn zu jedem Zeitpunkt der Algorithmusausführung der nächste Handlungsschritt eindeutig definiert ist.“[2]

Randomisierter Algorithmus

> „Ein randomisierter Algorithmus (auch stochastischer oder probabilistischer Algorithmus) versucht, durch die Wahl von zufälligen Zwischenergebnissen zu einem (im Mittel) guten bzw. näherungsweise korrekten Ergebnis zu gelangen. Er bildet somit das Gegenstück zum deterministischen Algorithmus.“[3]

Vereinfacht ausgedrückt handelt es sich bei *Algorithmen* um Rechenvorschriften – also um mathematische Prozeduren – und jede Aufgabenstellung, die mit einem *Algorithmus* lösbar ist, ist auch mit*hilfe* eines Computers lösbar.

Letztlich ist jedes Computerprogramm *Algorithmus* oder setzt sich aus vielen *Algorithmen* zusammen.

Komposition

Selbstverständlich reden wir über die musikalische *Komposition*[4]. Es soll jedoch dabei bewusst sein, dass der *Kompositions*-Begriff beispielsweise auch in der

[1] Vergleiche: http://www.zkm.de/algorithmische-revolution/ (August 2014).
[2] http://de.wikipedia.org/wiki/Algorithmus (August 2014).
[3] http://de.wikipedia.org/wiki/Algorithmus (August 2014).
[4] lateinisch compositio = Zusammenstellung oder componere = zusammenfügen.

Grammatik, der bildenden Kunst oder in der Mathematik besteht.

Wenn wir von *Algorithmen* sprechen und von *Komponisten*, die *Algorithmen* entwerfen oder programmieren, könnten wir auch die Idee der Meta-*Komponisten* und Meta-*Kompositionen*[1] aufgreifen. Es sollte klar sein, dass ein *Algorithmus* eines *Komponisten* andere *Komponisten* befähigt, mit diesen *Algorithmen* wiederum ihre *Kompositionen* zu gestalten.

Vielleicht ist deshalb so selten von den einer *Komposition* zugrundeliegenden *Algorithmen* zu erfahren.

Notation

> *„Musik lebt in der Zeit. Sie existiert physikalisch als periodische Schwingung in der Luft. Diese «ätherische» Kunst aus dem «Strom der Zeit» herauszugreifen und schriftlich zu fixieren, erscheint fast wie die Quadratur des Kreises.“*[2]

Obwohl heute unsere traditionelle *Notation* – oder die westeuropäische, abendländische *Notations*-Tradition – als nicht mehr zeitgemäß eingestuft werden könnte,[3] möchte ich meine Abhandlungen darauf aufbauen. Ich möchte auch unterstreichen, welch großartige Klangereignisse eben damit in der Vergangenheit geschaffen wurden und sicher auch noch in Zukunft geschaffen werden.

Die Möglichkeiten zu *notieren* sind begrenzt und es scheint im vorhandenen Zeichenvorrat vieles zu fehlen, wenn es um feine Nuancen der Tonhöhen-, Rhythmus- oder Klanggestaltung geht.

[1] Vergleiche: Douglas R. Hofstadter: *Gödel Escher Bach – ein Endloses Geflochtenes Band.* dtv/Klett-Cotta (München 1991) S. 646.
[2] Hermann Rauhe u. Reinhard Flender: *Schlüssel zur Musik.* Knaur (Ulm 1990) S. 68.
[3] Vergleiche: Christoph Herndler u. Florian Neuner (Hrsg.): *Der unfassbare Klang – Notationskonzepte heute.* Klever (Wien 2014) S. 8.

Interpreten von Musik haben gelernt, sozusagen auch „zwischen den Zeilen“ zu lesen und es ist mehr als erstaunlich, dass diese Zeichen-Schrift annähernd weltweit verstanden wird.

Es steht aber auch jedem Komponisten frei, traditionelle Notationsformen mit neuen zu kombinieren.

Meine Abhandlungen konzentrieren sich zwar auf die traditionelle Notation – sie sind aber auf ältere sowie auf neuere Notationspraktiken analog übertragbar.

Algorithmische Komposition

> *„Mir ist das Zetergeschrei bekannt, das sich an einigen Stellen erhebt: «Das ist purer Konstruktivismus, wo bleibt die Erfindung?» Von jeher war es in der Satzkunst so, daß man das, was andere beherrschten, was einem selbst aber noch nicht geläufig war, als erfindungstötende Konstruktion ansah. Die Regeln der Harmonielehre und des Kontrapunkts gelten nur deshalb nicht als Konstruktionsschemata, weil man an sie gewöhnt ist. Wo fängt auf dem Wege zwischen diesen Handwerkslehren und einer reichhaltigeren Arbeitsanweisung der Konstruktivismus an? Für den, der sich nie Gedanken über die Theorie des Tonsetzens gemacht hat, vermutlich sofort bei den Akkorden, die er sich wie eine verbotene Frucht vom Baume seiner mangelhaften Harmonieerkenntnis pflückt, die er als die Kinder der Sünde gegen die Harmonielehre besonders liebt und deshalb nicht in ihren abgründigen Eigenschaften kennen lernen will. Für den starken Musiker, der seiner Eingebung und seiner Kenntnisse sicher ist, gibt es keinen Konstruktivismus. Jeder erworbene Handgriff mehr zur Bewältigung des Materials ist ihm ein Schritt näher zum innersten Heiligtum der Musik.“*[1]
>
> Paul Hindemith

[1] Paul Hindemith: *Unterweisung im Tonsatz – Theoretischer Teil.* B. Schott´s Söhne (Mainz 1940) S. 194–195.

Wir können uns die Grenzen zwischen „traditioneller“ *Komposition* und *algorithmischer Komposition* fließend vorstellen. Jede Verwendung von Regeln beim *Komponieren* könnte man streng genommen bereits als *algorithmisch* bezeichnen.[1] *Algorithmisch* wird *komponiert*, wenn musikalische Zusammenhänge mathematisch formulierbar sind. Der *Komponist* findet Formeln oder Muster, nach denen er vorgeht. Der Prozess des *Komponierens* geht nicht Ton für Ton voran, sondern *Kompositionen* werden in größeren Zusammenhängen abgearbeitet. Oft werden Materialien wie Würfel oder Tonkarten verwendet, wenn Parameter, die den Zufall zu *kompositorischen* Entscheidungen zulassen, herangezogen werden.

Algorithmisch Komponieren könnte in weiterem Sinne auch mit Konstruieren verglichen werden. Die Kenntnis von zumindest einfachen mathematischen oder geometrischen Zusammenhängen ist dabei von Vorteil, wenn nicht unerlässlich. Jedenfalls unterscheidet sich die Vorgangsweise von *Komponisten*, die aus dem „Bauch heraus“ *komponieren* oder sich eine *Komposition* „erimprovisieren“ von jenen, die mit *algorithmischen* Überlegungen ans Werk gehen. Das heißt aber nicht, dass ein *Algorithmus* nicht auch aus dem „Bauch heraus“ entstehen könnte oder ein Meister sein Regelwerk nicht auch im „Bauch“ haben könnte.

Immer, wenn Begriffe wie:
Regeln, Muster, Parameter, Gesetzmäßigkeiten, Schablonen, Modi und so weiter verwendet werden, liegt der Verdacht nahe, dass im eigentlichen Sinne *algorithmisch komponiert* wird.[2]

Im Zusammenhang mit *algorithmischer Komposition* ist der Einsatz eines Computers zwar zeitgemäß, aber zur Definition nicht notwendig.

[1] Vergleiche: http://de.wikipedia.org/wiki/Algorithmische_Komposition (August 2014).
[2] Vergleiche: Claus-Steffen Mahnkopf: *Technik und moderne Musik*. In: Christoph Lütge und Torsten L. Meyer (Hrsg.): *Musik – Technik – Philosophie*. Karl Alber (Freiburg/München 2005) S. 151.

Wenn von *algorithmischer Komposition* die Rede ist, schwingt der Begriff **Aleatorik**[1] mit. Damit ist die Komponente des Zufalls zu verstehen.
Heute dürften wir von *randomisiertem Algorithmus* sprechen.

Ich möchte betonen, dass der Zufall in der *algorithmischen Komposition* nur soweit eine Rolle spielt, wie er auch in der Mathematik – beispielsweise in der Wahrscheinlichkeitsrechnung – eine Rolle spielt.

So wie wir den Begriff *Komposition* nicht nur in der Musik finden, so ist vom Prinzip her *algorithmische Komposition* auch in anderen, wenn nicht in allen Kunstsparten zu finden.
Oft wird auch von *generativer Kunst*, von *iterativen Prozessen* oder von *fraktaler Geometrie* gesprochen.

Das Gestalten mit *Algorithmen* scheint mir eindeutig von epochaler und spartenübergreifender Bedeutung.

Die computergestützte Komposition

Hier spricht man auch von ***CAC*** *(Computer Aided Composition)*.[2]
Der Vollständigkeit halber muss erwähnt werden, dass CAC auch als Abkürzung für *computer assisted Composition* verstanden wird, welche als gleichbedeutend angesehen werden kann.

Bei der computergestützten Komposition muss es sich nicht zwangsläufig um eine algorithmisch komponierte Komposition handeln. Es sind auch – besonders im Popularmusikbereich – sogenannte Sequenzerprogramme[3] in Verwendung. Diese Werkzeuge können für den Komponisten sehr nützlich werden, haben

[1] lateinisch aleatorius = „zum Spieler gehörig“, alea = „Würfel, Risiko, Zufall“.
[2] Vergleiche: David Cope: *The Algorithmic Composer*. A-R Editions (Madison, Wisconsin 2000) S. 1–2.
[3] Im Wesentlichen meine ich hier stellvertretend Softwaretools wie LOGIC oder CUBASE.

aber im Normalfall wenig mit algorithmischer Komposition zu tun.
Deswegen muss ich mich hier – aus Gründen des Themas und des Umfangs – distanzieren. Viele Soft- und Hardware-Werkzeuge können der Komposition dienlich sein, sind aber im engeren Sinn nicht unbedingt Werkzeuge zur algorithmischen Komposition.

Computergestützte algorithmische Komposition

CAAC *(Computer Aided Algorithmic Composition)* – noch treffender – und meiner Meinung nach auch richtiger – wäre:
Computer Aided Algorithmic Music Composition[1] und damit: ***CAAMC***.

> *„Now we know what an algorithm is, but what is the connection to computers? The key point is that computers need to be programmed with very precise instructions. Therefore, before we can get a computer to solve a particular problem for us, we need to develop an algorithm for that problem.“*[2]
>
> John MacCormick

Ein Gedanke hinter computergestützten algorithmischen Kompositionswerkzeugen und -techniken ist, mit einem „Knopfdruck“ einen Arbeitsauftrag an den Computer zu schicken, um die kompositorische Arbeit zu erledigen. Je nach Absicht und Fähigkeit des Komponisten werden dabei kleinere oder größere Abschnitte einer Komposition mit*hilfe* des Computers umgesetzt.

Für Kritiker der ernst zu nehmenden computergestützten algorithmischen Komposition sei hier angemerkt, dass der Computer nichts vollbringt, was der Komponist nicht auch von Hand erledigen könnte.[3]

Der Umgang des Computers mit mathematischen Problemstellungen, mit Zufällen und Wahrscheinlichkeiten, sowie dessen schnelle Entscheidungsfähigkeit kann dem Komponisten zum Umsetzen seiner Vorstellungen sehr dienlich sein.

[1] Vergleiche: http://www.algorithmic.net(August 2014).
[2] John MacCormick: *9 Algorithms that changed the future*. Princeton University Press (Princeton 2012) S. 4.
[3] Vergleiche: http://blog.zdf.de/hyperland/2012/10/kreative-maschinen-wie-algorithmen-komponieren/ (August 2014).

ICAAC *(Interactive Computer Aided Algorithmic Composition)* wäre mein Bezeichnungsvorschlag für die Definition, wenn es sich um die Möglichkeit der Einflussnahme ins kompositorische Geschehen handelt. Hier spricht man von Einflussnahme in *Echtzeit*. Der Computer rechnet zwar mitunter eine Unmenge von Algorithmen ab, es dauert aber heutzutage kaum länger als ein paar Millisekunden bis die Reaktion erfolgt. Dies kann sich so äußern, dass musikalische Aktionen scheinbar gleichzeitig passieren (beispielsweise das gleichzeitige Erklingen von parallelen Stimmen zur eigentlich gespielten Stimme beim Spiel auf dem digitalen Klavier), weil die kurze Reaktionszeit (auch *Latenzzeit*) dem Hörer kaum bis überhaupt nicht auffällt.

Wenn bei *CAAC* oder *ICAAC* auch in *Echtzeit* notiert wird, trifft diese Rubrik unseren Themenbereich.
Von Notation ist ansonsten hierbei keine Rede.

Computergestützte Notation

Oder auch: ***CAN*** *(Computer Aided Notation)*
Je nachdem wie ein Komponist arbeitet, wird eine Komposition gewöhnlich gar nicht, handschriftlich oder mit einem Notenschreibprogramm zu Papier gebracht.

Mir wäre in diesem Zusammenhang wichtig, dass die vorliegende Arbeit, in der es über die *Notation* in der *computergestützten algorithmischen Komposition* geht, nicht die übliche Hilfe des Computers zum Notieren behandelt.

Selbstverständlich existierten nützliche Software-Produkte, die in der Zwischenzeit den Computer – auch für den Komponisten – zum Notieren unverzichtbar machen. Musiker sind es gewöhnt, von einem perfekten Notenbild zu interpretieren. Äußerst selten sind Komponisten grafisch auch entsprechend begabt.

Ausnahmen wie Alfred Peschek[1] bestätigen die Regel.
Auch wenn es „nur“ – oder gerade, wenn es um Notation geht, ist der Computer mit sorgfältig ausgewählter Software ohne Zweifel ein nicht mehr wegzudenkendes Werkzeug für Komponisten.
(Aber nicht immer ein zeitsparendes ;-))

Notation in der computergestützten algorithmischen Komposition

Meist ist der Komponist, wenn er mit*hilfe* des Computers komponiert, mit Zahlen und Zahlenlisten konfrontiert. Die Motivation zur vorliegenden Arbeit wurde durch das Kennenlernen neuer Methoden ausgelöst. Die neuen Errungenschaften in der algorithmischen Komposition liegen in der Möglichkeit der unmittelbaren Umwandlung von generierten Zahlen bzw. Zahlenlisten in ein Notenbild und dessen Anzeige am Bildschirm. Für das Vorstellungsvermögen – zumindest für normalsterbliche Komponisten – eine große Errungenschaft.

Computergestützte algorithmisch notierte Komposition

Genau genommen müsste die Überschrift lauten:
Computergestützte algorithmische Komposition – computergestützt algorithmisch notiert.

Ich würde korrekter Weise bei der Definition – immer wenn algorithmische Prozesse im „Spiel“ sind – vereinfacht zumindest auf das zweite **A** Wert legen:
CAAN für (*Computer Aided Algorithmic Notation).*
CAANC würde die Thematik noch besser treffen.[2]

Spätestens hier wird das Dilemma deutlich. Nichtsdestotrotz wäre unter dieser

[1] Komponist in Linz (14. 5. 1929 – 4. 2. 2015). Mein Vorbild – auch was ansprechende Notation angeht – er schrieb sämtliche Partituren mit der Hand.
[2] Es gibt aufgrund der relativ jungen Entwicklung noch kaum einheitliche Definitionen für die doch sehr unterschiedlichen Bereiche.

Rubrik meiner Meinung nach die Königsdisziplin der algorithmischen Komposition angesiedelt:

Vielleicht die fertig notierte Komposition mit perfektem Layout
„auf Knopfdruck“
und als Draufgabe gleich fertig aus dem Drucker ausgedruckt.

Üblicherweise sind bei zu notierender algorithmischer Komposition vereinfacht betrachtet mehrere Arbeitsschritte erforderlich:

Erstens Programmieren des Algorithmus.

Zweitens Umwandeln des Produzierten in eine von einem Notationsprogramm verständliche Sprache.

Drittens Importieren des Umgewandelten in ein Notensatzprogramm.

Viertens Erstellen des Layouts.

Fünftens Ausdruck der Partitur.

Aufgrund der Umständlichkeit der Prozedur wird oft auf Notation von algorithmisch generierter Musik verzichtet. Algorithmisch generierte Musik begegnet uns deshalb nicht selten im Zusammenhang mit Klanginstallationen.

Ziel der Forschung und Tätigkeiten der algorithmisch komponierenden Komponistengemeinde ist, umständliche Workarounds so gut als möglich abzukürzen und dennoch die gewünschten Resultate zu erzielen.

Mein Ziel in der vorliegenden Arbeit wäre, letztlich einen möglichen Weg „zur fertig notierten Partitur auf Knopfdruck“ – zumindest als „Ausblick“ – aufzubereiten.

Auf dem Weg zum Ziel werde ich dem Unterfangen zulängliche Werkzeuge vorstellen und auch entsprechend aufbereiten.

3 MAX

3.1 Überblick MAX

MAX ist eine graphische Programmierumgebung, die die Programmierung von Musikprogrammen besonders unterstützt. Es handelt sich um eine sogenannte „höhere Programmiersprache“ welche bereits viele Schnittstellen zur Interaktion – insbesonders musikspezifische Schnittstellen – bereitstellt. Das heißt, dass nicht jede Programmier-Routine von Grund auf (sozusagen auf „LOW-Level“) programmiert werden muss, sondern in MAX bereits als sogenanntes OBJECT fertig zur Verfügung steht.

MAX bietet dem Musiker oder Komponisten eine Programmier-Plattform, die ihm Musik-Programmierung erlaubt, ohne dabei Gefahr zu laufen, den wesentlichen Bezug zur Musik aus den Augen zu verlieren und sich im darunterliegenden CODE zu verirren. Dennoch ist der Grad der Komplexität unbegrenzt, da sich programmierte Module (sogenannte „*Patches*“) miteinander verknüpfen lassen. Weltweit werden von der mit MAX auf „HIGH-Level“ programmierenden Musikgemeinde MAX-*Patches* entwickelt und ausgetauscht.

MAX ist mittlerweile nicht nur im Musik- sondern im gesamten Multimediabereich vertreten. Auf der Homepage von MAX[1] wird geworben mit:

> „A full kit of creative tools for sound, graphics, music and interactivity in a visual environment.“

[1] https://cycling74.com (Dezember 2014).

3.2 Geschichte von MAX

Die erste Version von MAX wurde 1988[1] von MILLER PUCKETTE[2] am IRCAM[3] in Paris entwickelt. Anfang 1990 wurde eine kommerzielle Version von OPCODE SYSTEMS[4] veröffentlicht (entwickelt und erweitert von DAVID ZICARELLI). Seit 1999 wird MAX über ZICARELLIs Firma CYCLING'74 vertrieben.

MAX ist nach MAX MATHEWS[5] benannt und kann als Nachfolger von MUSIC-N[6] angesehen werden.[7] MAX wurde ursprünglich zur Verarbeitung von Midi-Signalen in Echtzeit eingesetzt. 1997 wurde MAX mit MSP *(Max Signal Processing)*, der Möglichkeit, Audio in Echtzeit zu verarbeiten, ergänzt. Die vollständige Bezeichnung lautete danach MAX/MSP. Seit dem Jahr 2002 ist mit dem zusätzlichen Programmpaket JITTER die Bearbeitung beziehungsweise Manipulation von Videosignalen in Echtzeit möglich.

Vor der Version 6 des Programms waren die Erweiterungen MSP und JITTER separat erhältlich. Seit dem Erscheinen der Version 6 im Jahr 2011 sind alle Erweiterungen zu einem Programm-Paket und wieder zu einem Namen: MAX vereint.

Zu Beginn war MAX „nur" für Apple-Macintosh-Computer entwickelt worden. Erst Jahre später war das Programm auch für Computer mit Windows-Betriebssystemen erhältlich.

[1] http://de.wikipedia.org/wiki/IRCAM (Dezember 2014).
[2] MILLER PUCKETTE (* 1959) ist der Entwickler der beiden Programme MAX und PURE DATA. Derzeit ist er *Associate Director* für die Abteilung *Music* am *Center for Research in Computing and the Arts (CRCA)* der *University of California*, San Diego.
[3] IRCAM ist die Abkürzung für *Institut de Recherche et Coordination Acoustique/Musique*. Es befindet sich im Centre Pompidou in Paris.
[4] Opcode Systems, Inc. wurde 1985 von DAVE OPPENHEIM gegründet.
[5] MAX VERNON MATHEWS (13. 11. 1926 - 21. 04. 2011) war ein US-amerikanischer Elektroingenieur und Pionier der Computermusik.
[6] Das Programm hat MAX MATHEWS 1957 bei BELL LABS geschrieben. Es war das erste Computerprogramm, das digitale Wellenformen durch direkte Synthese erzeugen konnte.
[7] Vergleiche: http://de.wikipedia.org/wiki/Max/MSP (Dezember 2014).

3.3 Grundlegende Objects zur CAAC in MAX

Mittlerweile sind bei der „Grundausstattung“ von MAX mehr als 600 OBJECTS inkludiert. Für unser Vorhaben in der *computergestützten algorithmischen Komposition* ist MAX aber seit seiner ursprünglichen Variante schon ausreichend entwickelt. Es waren die meisten der dafür wesentlichen Bausteine schon in Form von OBJECTS implementiert.

In der Folge werden wesentliche OBJECTS in ihrer Funktion, und wie sie sich optisch in der Version 6 zeigen, kurz vorgestellt:

Objects für einfache Rechenoperationen

+	*plus*	addiert zwei Zahlen und gibt das Ergebnis aus.
-	*minus*	subtrahiert zwei Zahlen und gibt das Ergebnis aus.
*	*times*	multipliziert zwei Zahlen und gibt das Ergebnis aus.
/	*div*	dividiert zwei Zahlen und gibt das Ergebnis aus.
%	*modulo*	dividiert zwei Zahlen und gibt den Rest aus.
expr ($i1+$i2) * $i3		berechnet einen mathematischen Ausdruck.

Vergleichende Operatoren

>	*greaterthan*	größer als
<	*lessthan*	kleiner als
>=	*greaterthaneq*	größer oder gleich
<=	*lessthaneq*	kleiner oder gleich
==	*equals*	gleich
!=	*notequals*	nicht gleich
if $f1 > 0. then 1. else -1.		bedingte Anweisung

Objects für Midi Ein- und Ausgabe

notein	noteout	empfängt oder sendet MIDI NoteOn-Nachrichten.
ctlin	ctlout	empfängt oder sendet MIDI Controller-Werte.
bendin	bendout	empfängt oder sendet MIDI Pitch-Bend-Werte.
xbendin	xbendout	empfängt oder sendet 14-Bit MIDI Pitch-Bend-Werte.
touchin	touchout	empfängt oder sendet MIDI Aftertouch-Werte.
pgmin	pgmout	empfängt oder sendet MIDI Programm-Nummern.
midiin	midiout	empfängt oder sendet MIDI Roh-Daten.
sysexin	sxformat	empfängt oder bereitet MIDI system exclusive Nachrichten auf.
makenote		generiert ein Note-On-/Note-Off-Paar.

Zeitbasierende Objects

metro	gibt in regelmäßigen Intervallen Bang-Messages aus.
tempo	gibt Zahlen in einem metronomischen Tempo aus (in Beats per Minute [BPM]).
clocker	gibt in regelmäßigen Abständen die verstrichene Zeit aus.
delay	verzögert einen Bang.
pipe	verzögert Zahlen, Listen oder Symbole.

Wichtige allgemeine Objects

select	gibt bei Übereinstimmung Bangs aus.
gate	routet das Eingangssignal zu einem bestimmten Ausgang.
switch	routet einen definierten Eingang zum Ausgang.
counter	gibt die Anzahl der Bang-Nachrichten innerhalb eines festgelegten Bereichs aus.
uzi	sendet so schnell als möglich eine festgelegte Anzahl von Bangs.

Objects zum Speichern von Daten

speichert, addiert und multipliziert Zahlen.

speichert, organisiert oder gibt
Sammlungen von verschiedensten Daten wieder aus.

Ablaufsteuerung zum Aufnehmen und Abspielen von Midi-Daten.

zeichnet Nachrichten auf und
ermöglicht eine ablaufgesteuerte Wiedergabe.

Objects für zufallsgesteuerte Prozesse

generiert eine Zufallszahl.

generiert Zufallszahlen ohne Wiederholungen.

gibt Zufallszahlen innerhalb definierter Schrittweiten aus.

wählt die Zufallswerte EINS oder NULL.

UserInterface-Objects (UI-Objects)

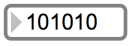
Diese OBJECTS unterscheiden sich von den vorangehenden durch die Möglichkeit der Interaktion durch den Benutzer eines MAX-Patches.


button sendet und zeigt eine Bang-Nachricht an.

Der BANG wird verwendet um Nachrichten oder Prozesse zu triggern.
Es handelt sich dabei in MAX um *die* essentielle Botschaft.
Diese wird auch mit *DO IT!* übersetzt.


toggle Schalter EIN/AUS.

number zeigt und gibt ganze Zahlen ein oder aus.



number zur Anzeige von binären Zahlen formatiert.

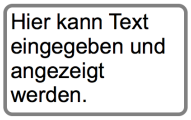
number zur Anzeige von Midi-Noten-Namen formatiert.

flonum zeigt und gibt Fließkomma-Zahlen ein oder aus.

umenu

erzeugt ein Pop-up-Menü.

textedit

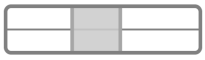


ermöglicht Nachrichten direkt zu schreiben
oder anzeigen zu lassen.

slider

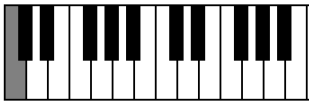
gibt Zahlen in Form eines Schiebereglers
innerhalb eines definierten Bereichs aus.

rslider



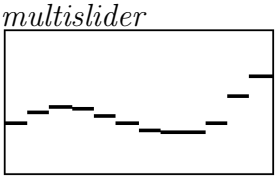
definiert einen Werte-Bereich
vom niedrigsten zum höchsten.

kslider



gibt Tonhöhe und Anschlagstärke als Zahl aus.

multislider

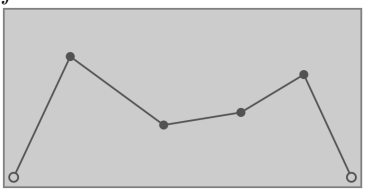


zeigt Daten in einer Anzahl von Schiebereglern oder
bildet ein laufendes Anzeigeband.

dial

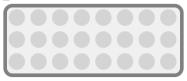
gibt Zahlen in Abhängigkeit des Umdrehungswinkels aus.

function



graphische Bearbeitungsmöglichkeit einer Funktionskurve.

preset



Speichern und Aufrufen von Programm-Zuständen.

Nachrichten in MAX

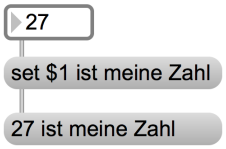
Nachrichten beziehungsweise *Messages* werden in Form von *Message-Boxen* übermittelt.

message

zeigt und sendet eine Nachricht.

In der Dokumentation von MAX steht der Zusatz:
Send a simple message, or construct a message of any degree of complexity[1]

Zumal auch sogenannte *changeable-arguments* in die *Message-Box* eingefügt werden können, ist der Umgang mit ihnen sehr vielseitig.

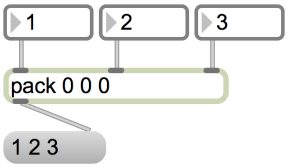


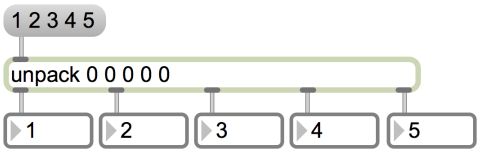
[1]Dokumentation des OBJECTS *message* innerhalb der Software MAX (Version 6).

Listen in MAX

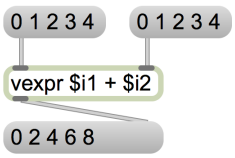
Der Behandlung von *Listen* in MAX kommt eine besondere Bedeutung zu. *Listen* sind *mächtige*[1] Datenstrukturen in MAX. Sie können beliebige Anordnungen von Zahlenwerten repräsentieren. Eine *Liste* kann als *Message* an andere OBJECTS gesendet – oder auch beliebig manipuliert werden. Der Umgang mit *Listen* ist für unser Vorhaben bezüglich der *Notation in der CAAC* unumgänglich.
Folgend werden einige wesentliche Bausteine angeführt.

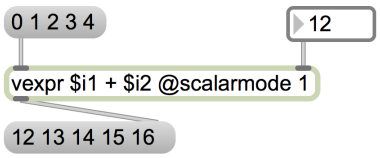
Objects für Listenoperationen



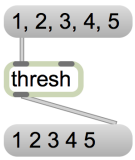


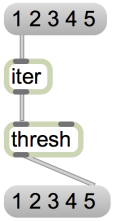
pack und *unpack* zum Bilden von *Listen* oder zum Aufbrechen von *Listen* in einzelne Nachrichten.





vexpr berechnet mathematische Ausdrücke für *Listen*.

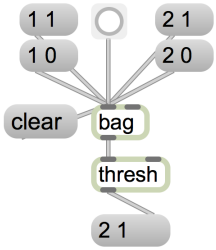


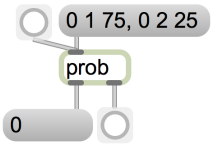


thresh sammelt Zahlen und *Listen* und vereint sie zu *Listen*, wenn sie innerhalb einer voreinstellbaren Zeit eintreffen.
iter bricht eine *Liste* in individuelle Nachrichten auf.

Listen-Einträge, welche mit Beistrichen getrennt sind, werden nacheinander gesendet (!).

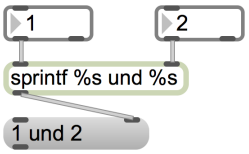
[1] Der Begriff *mächtig* ist in Programmiersprachen gängig, wenn es sich um umfangreiche und komplexe Datenstrukturen handelt.

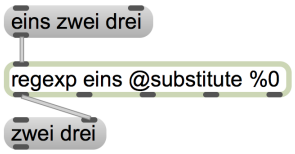


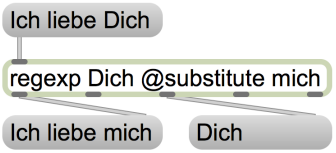


bag speichert und verwaltet eine Ansammlung von Zahlen.

prob gibt eine *gewichtete* Zufallsentscheidung aus.
Die Berechnung basiert auf den unmittelbar vorhergehenden Ausgang und der angegebenen Wahrscheinlichkeit, mit welcher eine Zahl der anderen folgt.

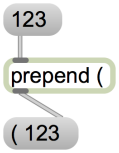


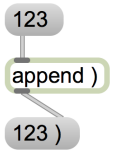


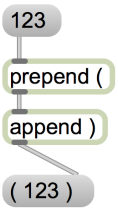


sprintf formatiert Nachrichten von Wörtern und Zahlen.

regexp *regular expressions* – ein in vielen Computersprachen verwendetes Zeichenformat kann aus Text beziehungsweise MAX-Nachrichten bestimmte Eigenschaften herausfiltern oder abgleichen.





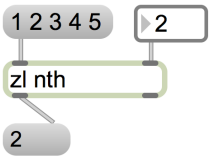


prepend setzt Zeichen vor eine Nachricht.

append hängt Zeichen an eine Nachricht.

Das wesentliche OBJECT zum Manipulieren von *Listen* in MAX ist *zl*.

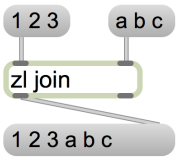
zl ist dem Programmpaket MAX in 33 verschiedenen Variationen beigefügt. Einige sind hier angeführt:

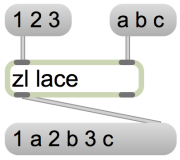


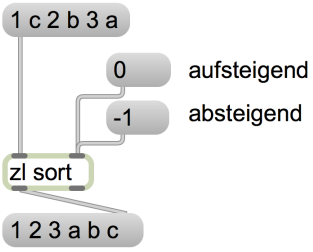
zl sum gibt die Summe der *Listen*-Einträge aus.

zl len gibt die Anzahl der *Listen*-Einträge aus.

zl nth gibt das n-te Element der *Liste* aus.



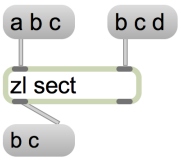


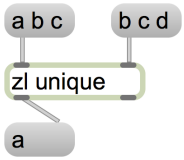


zl join kombiniert zwei *Listen*.

zl lace verschachtelt zwei *Listen*.

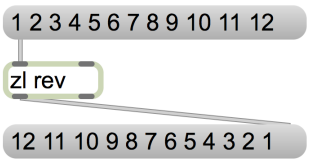
zl sort sortiert zwei *Listen*.

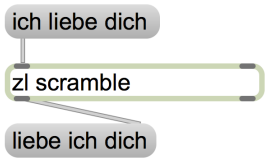




zl sect gibt das Gemeinsame von zwei *Listen* aus.

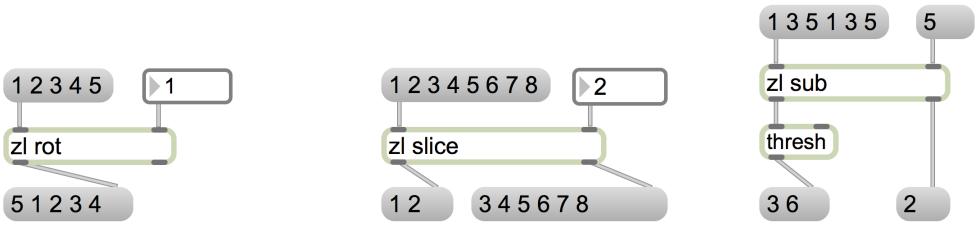
zl unique gibt die Teile einer *Liste* aus, die in der *Liste* am rechten Eingang des OBJECTS nicht vorkommen.





zl rev kehrt eine *Liste* um.

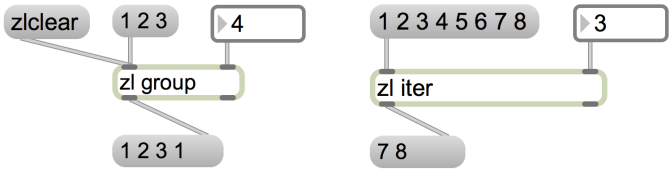
zl scramble gibt eine zufällige Anordnung der Elemente der *Liste* aus.



zl rot rotiert eine *Liste* um n Plätze.

zl slice gibt die ersten n Elemente am linken Ausgang des OBJECTS aus, die restlichen Elemente am rechten.

zl sub gibt die Position der Elemente der rechten *Liste*, die in der linken *Liste* vorkommen, wieder. Der rechte Ausgang gibt die Anzahl der gemeinsamen Vorkommnisse aus.

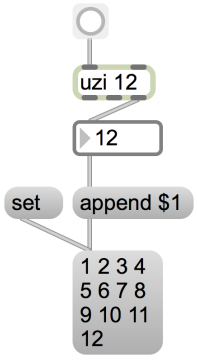


zl group gibt eine *Liste* erst aus, wenn n Elemente empfangen wurden.

zl iter gibt nacheinander *Listen* mit n Elementen aus.

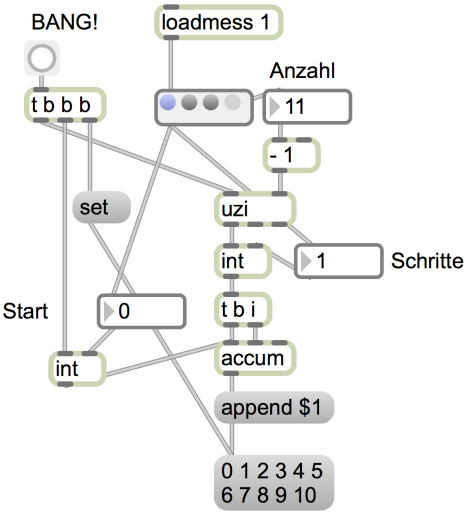
Weil Berechnungen in MAX annähernd in Prozessorgeschwindigkeit passieren, lässt sich meist optisch nicht nachvollziehen, was wann genau passiert. Im Beispiel von *zl iter* können zur Veranschaulichung ein *textedit* UI-OBJECT verwenden.

Es gibt für jede Problemstellung mehrere Lösungen – nicht anders in MAX. Eine Lösung zum Erstellen einer Liste bietet die *Message append*:

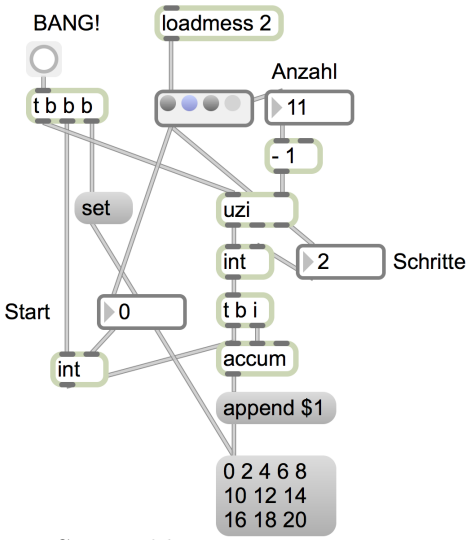


Diesen kleinen MAX-*Patch* – oder auch hier *Algorithmus* – zum Erstellen von Zahlenlisten könnten wir für noch allgemeinere Lösungen erweitern.

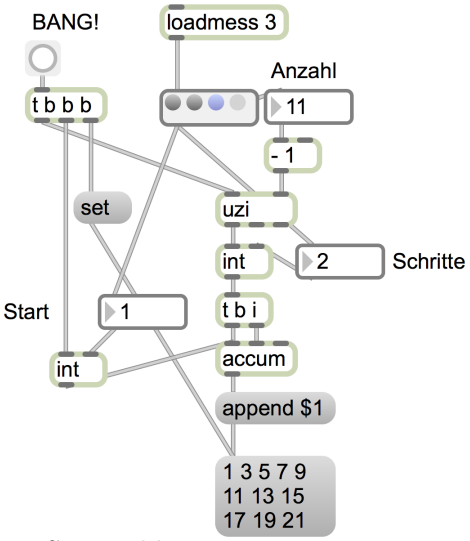
Die *Startzahl*, die *Anzahl* der Listeneinträge und die *Schrittweite* sollten einstellbar sein. Das MAX-OBJECT *accum* leistet uns dabei gute Dienste:



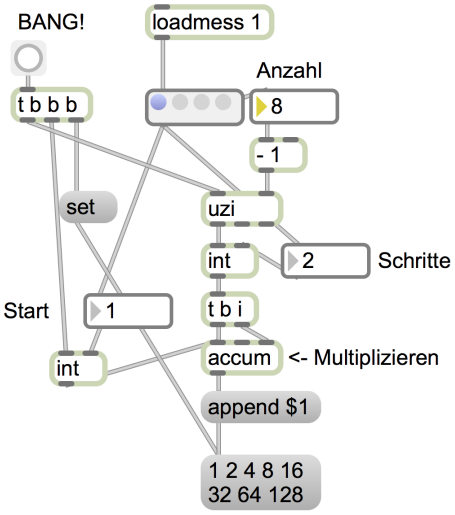
Startzahl: 0
Anzahl: 11
Schrittweite: 1



Startzahl: 0
Anzahl: 11
Schrittweite: 2



Startzahl: 1
Anzahl: 11
Schrittweite: 2



Startzahl: 1
Anzahl: 8
Schrittweite: 2

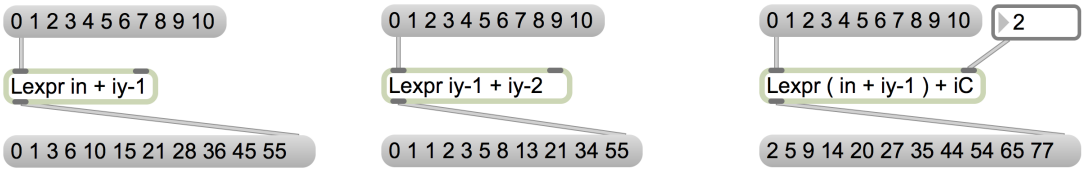
3.4 Externe Objects im Umgang mit Listen

Nachdem das Operieren mit *Listen* in MAX so wesentlich ist, kann es nicht genug Möglichkeiten geben, sie manipulierbar zu machen. Die Werkzeuge in Form von OBJECTS sollten für sämtliche Eventualitäten vorhanden sein. Für viele Vorhaben ist MAX von Grund auf gerüstet. Für den kreativen Geist gibt es allerdings keine Grenzen und so werden weltweit immer weitere OBJECTS, auch für das Operieren, Manipulieren oder für den Aufbau von *Listen* entwickelt.

Zusätzliche beziehungsweise ergänzende OBECTS werden dem MAX-Programmierer entweder als sogenannte *Externals*[1] oder *Abstractions*[2] zur Verfügung gestellt.

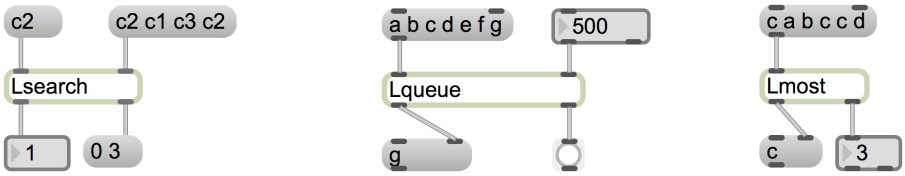
[1] *Externals* sind kompakte OBJECTS in der Programmiersprache *C* entwickelt.
[2] *Abstractions* sind aus den vorhandenen OBJECTS von MAX zusammengestellt. Bei einem Doppelklick auf ein *Abstraction*-OBJECT wird die dahinterliegende MAX-Programmierung offenbart.

Einen großen Fundus von über 120 *Listen*-OBJECTS hat beispielsweise Peter Elsea[1] mit seinen LOBJECTS schon seit 1994 als Download[2] zur Verfügung gestellt und dabei ständig weiterentwickelt.
Einige der LOBJECTS wurden im Laufe der Jahre durch das *zl*-OBJECT ersetzt, viele davon verloren nicht an Nützlichkeit:
Beispielsweise:



Lexpr ist ähnlich dem *expr*-OBJECT mit besonderen Variablen:

in, fn	=	momentanes Element der Eingangs-*Liste*.
in-1, fn-1	=	vorausgegangenes Element der Eingangs-*Liste*.
iy-1, fy-1	=	vorausgegangenes Ergebnis.
iC, fC	=	Konstante am rechten Eingang.

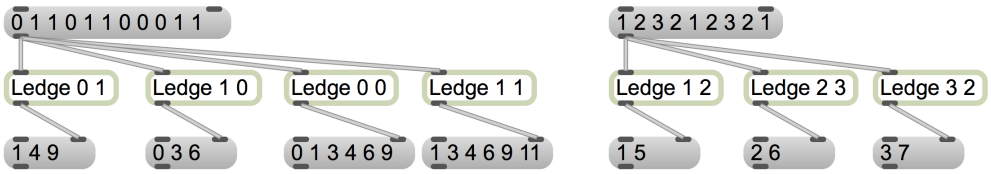


Lsearch gibt aus, ob und an welchen Stellen ein Element in einer *Liste* vorkommt.

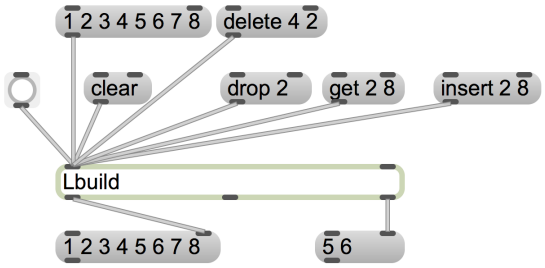
Lqueue Zahlen in einer *Liste* werden nacheinander nach *n* Millisekunden ausgegeben.

Lmost gibt das am häufigsten vertretene Element in einer *Liste* aus. Wie oft es vorkommt, wird am rechten Ausgang ausgegeben.

[1] Seit 1980 Direktor von *Electronic Music Studios* an der *University of California*, Santa Cruz.
[2] ftp://arts.ucsc.edu/pub/ems/Lobjects (Dezember 2014).

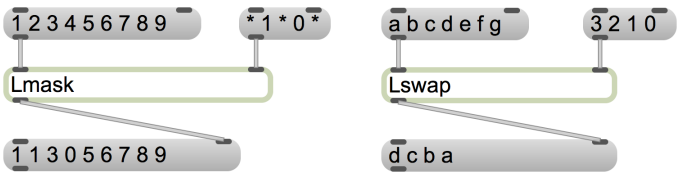


Ledge findet Wechsel oder Übergänge in einer *Liste*.



Lbuild baut *Listen*.

delete n	entfernt das Element an der Stelle n.
delete n m	das zweite Argument legt fest, wie viele Elemente der *Liste* entfernt werden.
drop	entfernt das letzte Element. Die entfernten Elemente erscheinen am rechten Ausgang.
get n	gibt das Element an der Stelle n aus.
get n m	das zweite Argument legt fest, wie viele Elemente der *Liste* ausgegeben werden.
insert	fügt ein Elemente ein. Das erste Argument gibt die Stelle an, an der eingefügt wird.



Lmask deckt Teile einer *Liste* ab.

Lswap zum Neuordnen einer *Liste*.

Lisp

Nicht zuletzt aus historischen Gründen möchte ich erwähnen, dass in der Programmiersprache LISP[1] die *Listen* zu Hause sind. LISP steht für **LIS**t **P**rocessing . Viele der heute bekannte *Listen*-OBJECTS haben ihren Ursprung – und tragen meist auch fairerweise ihre Namen – aus LISP.

> *„Lisp's core occupies some kind of local optimum*
> *in the space of programming languages.“*
>
> John McCarthy[2]

LISP kann – anders als MAX – auch mit verschachtelten *Listen* beziehungsweise mit *Listen* von *Listen* umgehen. Hier einige typische *Anweisungen* mit ihrer *Ausgabe* in der *Syntax* von LISP:[3]

```
(first '(a b c d))  => (A)
(second '(a b c d)) => (B)
(rest '(a b c d))  => (B C D)

(last '(a b c d))  => (D)
(last '(a b c d) 2)  => (C D)

(butlast '(a b c d))  => (A B C)
(butlast '(a b c d) 2)  => (A B)

(nth 0 '(eins zwei drei))  => eins
(nth 1 '(eins zwei drei))  => zwei
```

[1] LISP ist nach FORTRAN die zweitälteste Programmiersprache.
COMMON LISP (CL) und SCHEME sind zwei bekannte *Dialekte*.

[2] JOHN MCCARTHY (1927 – 2011) war US-amerikanischer Logiker, Informatiker und Autor. Er hat 1958 die Programmiersprache LISP an der *Massachusetts Institute of Technology (MIT)* erfunden.

[3] Vergleiche: http://letoverlambda.com (Jänner 2015).
http://jtra.cz/stuff/lisp/sclr/index.html (Jänner 2015).
http://www.lispworks.com/documentation/HyperSpec/Front/Contents.htm (Jänner 2015).

```
(nth 3 '(eins zwei drei))  => (NIL)

(reverse '(a b c d))  => (D C B A)
(list-length '(a b c d))  => 4

(append '(a b c) '(d e f))  => (A B C D E F )
(merge 'list '(1 3 4 6 7) '(2 5 8) #'<) => (1 2 3 4 5 6 7 8)
(union '(1 2 3) '(2 3 4))  => (4 1 2 3)
(set-difference '(a b c d e) '(a c)) => (E D B)
(intersection '(1 2 3 4 5) '(2 3 4)) => (4 3 2)

(remove 4 '(1 2 3 4 5))  => (1 2 3 5)
(remove-duplicates '(a a b c c a))  => (B C A)
(remove-duplicates '(a a b c c a) :from-end t)  => (A B C)
(remove-if #'evenp '(1 2 3 4 5 6 7)) => (1 3 5 7)
(remove-if #'oddp '(1 2 3 4 5 6 7)) => (2 4 6)
(remove-if (complement #'oddp) '(1 2 3 4 5 6 7)) => (1 3 5 7)
(substitute 9 4 '(1 2 3 4 5)) => (1 2 3 9 5)

(lcm 1 2 3 4 5 6) => 60
(gcd 63 -42 35) => 7

(make-list 5 :initial-element 'a) => (A A A A A)

(sort '(5 1 2 4 3 8 9 6 7 11) #'<) => (1 2 3 4 5 6 7 8 9 11)

(mapcar #'abs '(3 -4 2 -5 -6)) => (3 4 2 5 6)
(mapcar #'car '((1 a) (2 b) (3 c))) =>  (1 2 3)
(mapcar '- '(2 -4 6)) => (-2 4 -6)
(mapcar #'round '(1.3 2.7 3.4 4.5)) => (1 3 3 4)
(mapcar #'* '(3 4 5) '(4 5 6)) => (12 20 30)

(mapcan (lambda (x) (and (numberp x) (list x))) '(a 1 b c 3 4 d 5)) => (1 3 4 5)
(mapcan (lambda (x) (if (> x 0) (list x) nil)) '(-4 6 -23 1 0 12 )) => (6 1 12)
```

```
(mapcon #'list '(1 2 3 4)) =>  ((1 2 3 4) (2 3 4) (3 4) (4))

(subst 10 1 '(1 2 (3 2 1) ((1 1) (2 2)))) => (10 2 (3 2 10) ((10 10) (2 2)))

(member 1 '(0 1 0 0 0 1 0)) => (1 0 0 0 1 0)

(search "b" "anton bruckner") => 6

(subseq "anton bruckner" 6) => "bruckner"
(subseq "anton bruckner" 6 7) => "b"

(equal '(1 2 3 4 5) '(1 2 3 4 5)) => T
(subsetp '(3 2 1) '(1 2 3 4)) => T
(subsetp '(1 2 3 4) '(3 2 1)) => NIL
```

Bei Fehlen von vordefinierten LISP-Befehlen können neue definiert werden:
Beispielsweise zum *Permutieren* von Listen[1]

```
(defun all-permutations (lst &optional (remain lst))
(cond ((null remain) nil)
      ((null (rest lst)) (list lst))
      (t (append
      (mapcar (lambda (l) (cons (first lst) l)) (all-permutations (rest lst)))
(all-permutations (append (rest lst) (list (first lst))) (rest remain))))))

(all-permutations '( 1 2 3))  =>  ((1 2 3) (1 3 2) (2 3 1) (2 1 3) (3 1 2) (3 2 1))
```

oder zum Replizieren von *Listen*-Einträgen[2]

```
(defun repli (lista int &optional (ini int))
```

[1] http://stackoverflow.com/questions/2087693/how-can-i-get-all-possible-permutations-of-a-list-with-common-lisp (Jänner 2015).
[2] http://www.ic.unicamp.br/~meidanis/courses/mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (Jänner 2015).

```
(cond ((eql lista nil) nil)
        ((<= int 0) (dupli (cdr lista) ini ini))
        (t (cons (car lista) (dupli lista (1- int) ini )))))

(repli '(a b c) 3)  =>  (A A A B B B C C C)
```

oder um eine zufällige Anzahl von Elementen aus einer *Liste* zu erhalten[1]

```
(defun rnd-select (org-list num &optional (selected 0))
(if (eql num selected)
    nil
    (let ((rand-pos (+ (random (length org-list)) 1)))
       (cons (element-at org-list rand-pos) (rnd-select (remove-at org-list rand-pos)

(defun element-at (org-list pos &optional (ini 1))
    (if (eql ini pos)
        (car org-list)
        (element-at (cdr org-list) pos (+ ini 1))))

(rnd-select '(a b c d e f g h) 3) => (B D E)
```

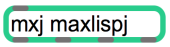
oder um eine Anzahl von Primzahlen zu erhalten.[2]

```
(defun primes (i)
(if (and (integerp i) (> i 0))
  (append (primes (+ i -1))
       (if (primep i)
       (list i)
         nil) )))

(defun primep (n)
```

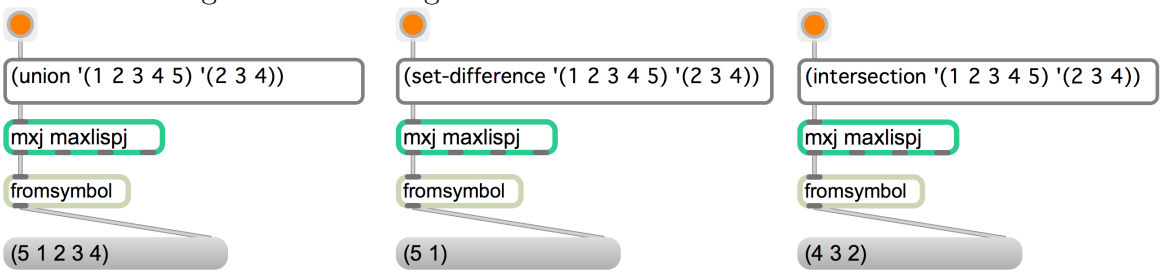
[1] http://www.ic.unicamp.br/~meidanis/courses/mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (Jänner 2015).
[2] http://www.ugcs.caltech.edu/~rona/tlisp/tlsamples.html (Jänner 2015).

```
(primep_extra n 2))

(primes 50) => (1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47)
```

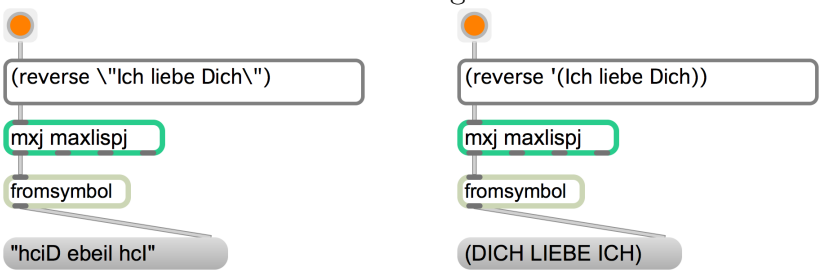
Es scheint, als ob für LISP mit *Listen* nichts unmöglich ist.
Glücklicherweise hat BRAD GARTON[1] ein OBJECT für MAX entwickelt und der MAX-Welt zur Verfügung gestellt.[2]



maxlispj versteht LISP-Anweisungen und gibt das Ergebnis aus. Mit dieser Ergänzung wird die *Macht* der LISP-Programmierung mit der *Macht* von MAX verbunden.
Die Verbindung könnte sich folgendermaßen auswirken:

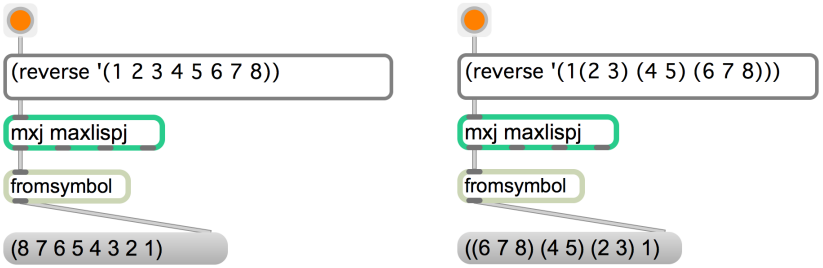


LISP versteht sich auch hervorragend mit Text:

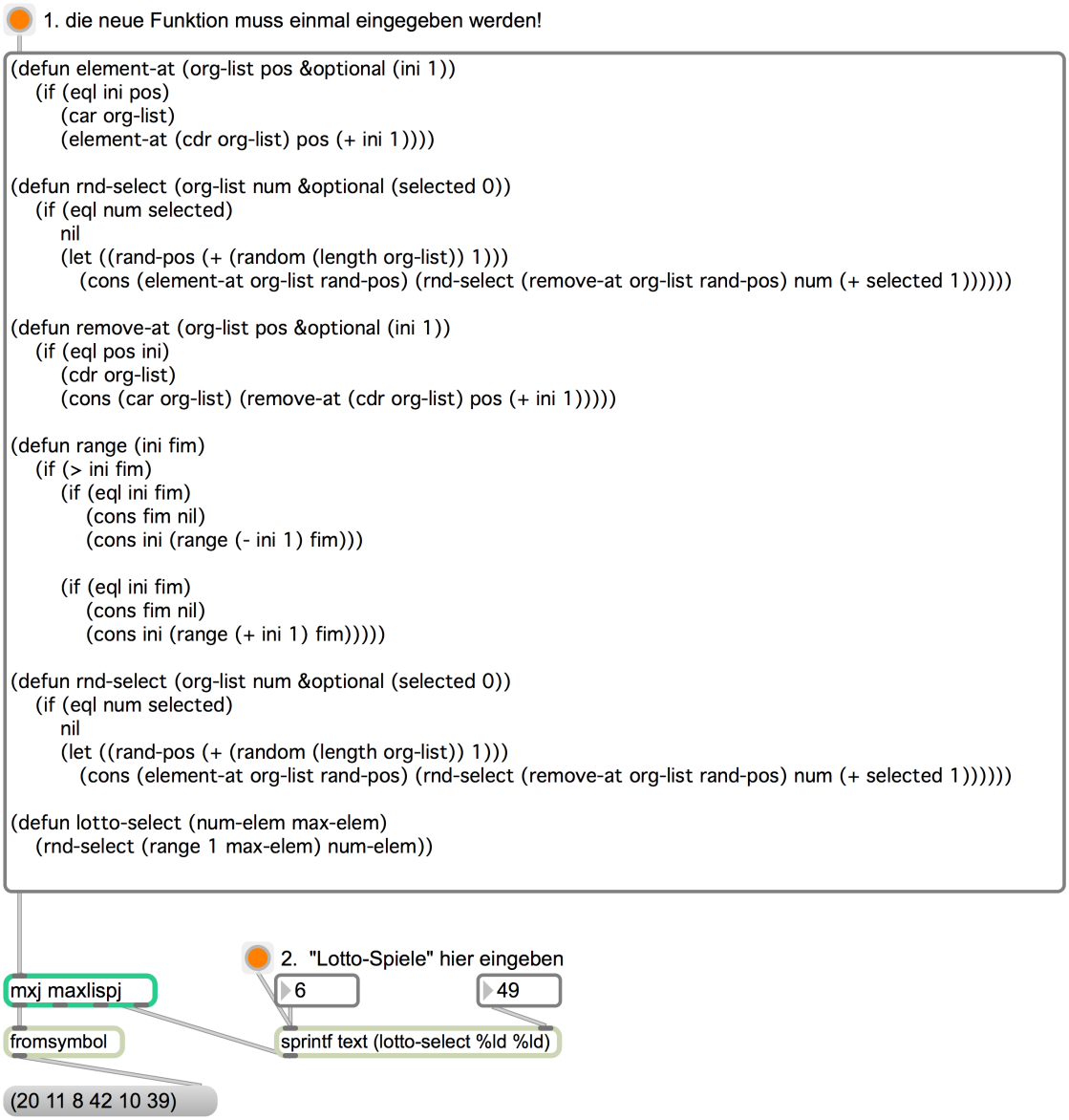


[1] BRAD GARTON ist Professor an der *Columbia University* und Direktor des *Computer Music Center* (früher *Columbia-Princeton Electronic Music Center*).
[2] http://music.columbia.edu/~brad/maxlispj/ (Jänner 2015).

mit *Listen* in *Listen*:

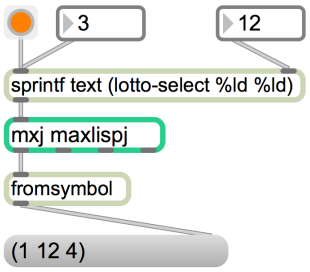


und auch mit der Integration zusätzlicher LISP-Funktionen[1]:

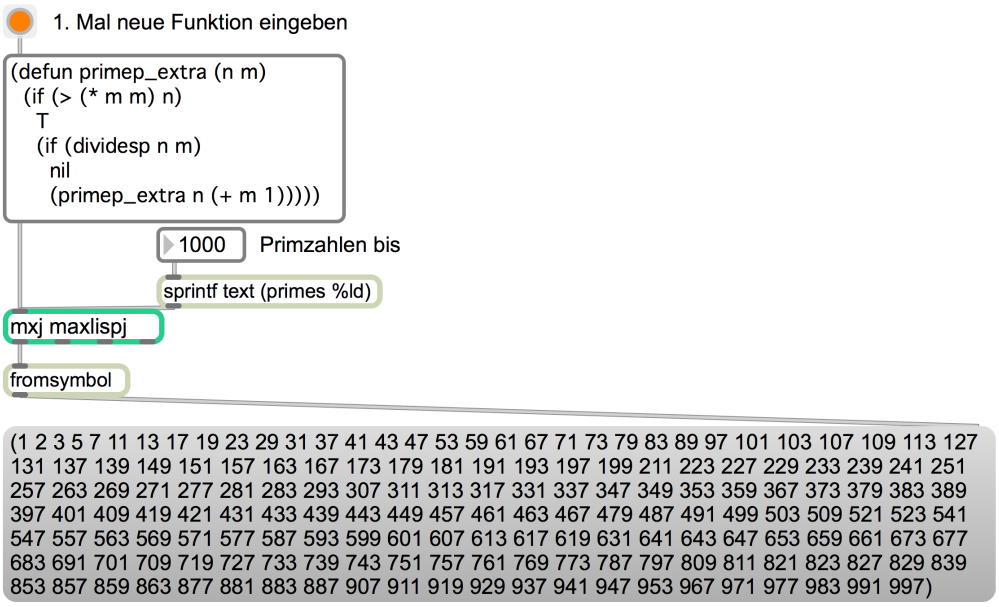


[1] http://www.ic.unicamp.br/~meidanis/courses/mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (Jänner 2015).

Nachdem die neue Funktion einmal eingegeben wurde, geht es kürzer weiter:



Es gibt mehrere Möglichkeiten, hier mit LISP – eine *Liste* mit Primzahlen[1] zu füllen:



3.5 Verwandte von MAX

Im Vergleich zu MAX existieren sehr ähnlich zu handhabende Softwareprodukte. Der Vollständigkeit halber eine kurze Auflistung:

[1] http://www.ugcs.caltech.edu/~rona/tlisp/tlsamples.html (Jänner 2015).

- *j*MAX[1]
  ist eine Variante der MAX-Familie. *j* steht für JAVA. JAVA ermöglichte das graphische Interface auf allen *Plattformen* – auch für LINUX – umzusetzen.

- PD[2]
  wurde entwickelt, um dem Beispiel von MAX zu folgen, aber die Daten-Prozessierung von Midi und Audio auf Echt-Zeit-Video und Web-Interaktion zu erweitern.

- VVVV[3]
  oder auch 4V – ein *multipurpose toolkit* – ist eine sehr umfangreiche und leistungsstarke moderne *hybrid visual/textual*-Programmierumgebung.

- BIDULE[4]
  Dem langjährigen Beobachter der verschiedenen Strömungen der *CAAC* bleiben Annäherungen und Angleichungen der verschiedenen Produkte nicht verborgen – wer hier von wem lernt, kann nur durch Beobachtung vermutet werden.

- CPS[5]
  Dieses Produkt wird derzeit nicht weiterentwickelt. Es funktioniert hervorragend auf OSX für PPC-Computer und Windows XP. Es ist und war eine ideale Ergänzung zur Integration von Midi und Audio innerhalb der Multimedia-Umgebung MACROMEDIA DIRECTOR (heute: ADOBE DIRECTOR) bis zur Version *Mx*.

- REAKTOR[6]
  ist ein Produkt der Berliner Software-Firma NATIVE INSTRUMENTS mit sehr leistungsstarken und komplexen Möglichkeiten – ganz besonders im Audio-Bereich.

[1] http://jmax.sourceforge.net (Jänner 2015).
[2] http://puredata.info (Jänner 2015).
[3] http://vvvv.org (Jänner 2015).
[4] http://www.plogue.com (Jänner 2015).
[5] http://cps.bonneville.nl (Jänner 2014).
[6] http://www.native-instruments.com/de/products/komplete/synths/reaktor-5/ (Jänner 2015).

4 bach: automated composer's helper

4.1 bach

bach: automated composer's helper[1] in der Folge kurz: *bach* ist eine Sammlung von *Patches* und *Externals* für MAX. Obwohl *bach* „nur" als eines von unzähligen *Libraries* für MAX in den Weiten des Internets zu finden ist, stellt es ganz besondere Möglichkeiten zur Verfügung, denen ich hier ein eigenes Kapitel widmen möchte. Die Entdeckung von *bach* hat mich bewogen, über die Thematik der *Notation in der computergestützten algorithmischen Komposition* zu schreiben.

bach ist „the missing Link" in MAX und bringt die ansonsten abstrakte Darstellung von musikalischen Informationen in Zahlen oder *Listen* in ein Notenbild. Für dieses Vorhaben sind viele neue Objects notwendig und in bewundernswerter Weise von Andrea Agostini[2] und Daniele Ghisi[3] auch dafür entwickelt worden.

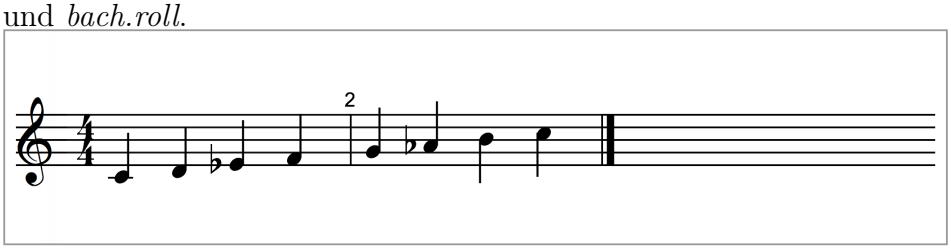
Viele der Errungenschaften aus *bach* sind dennoch nicht unbekannt. Seit Jahren dient OpenMusic (OM)[4] von Ircam als Vorbild dafür, wie Zahlenlisten

[1] http://www.bachproject.net (Jänner 2015).
[2] Andrea Agostini ist Komponist und lebt in Italien und Frankreich.
[3] Daniele Ghisi ist Mathematiker und Komponist aus Italien.
http://www.danieleghisi.com/biography/ (Jänner 2015).
Er erhielt 2012 für *bach* die *AFIM-Jeune Chercheur-* und *A. Piccialli*-Preise.
[4] http://repmus.ircam.fr/openmusic/home (Jänner 2015).

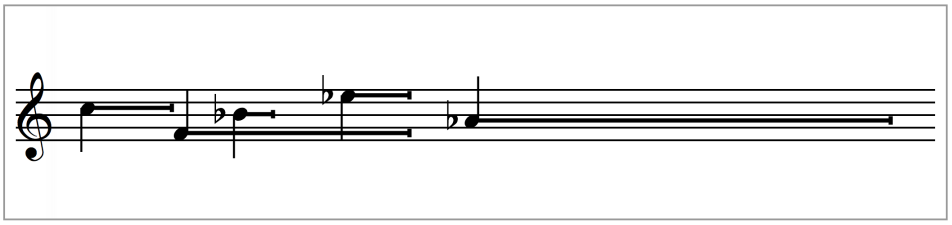
ansprechend als Notation repräsentiert werden.
Das entscheidend Neue mit *bach* sind die Möglichkeiten in *Echtzeit* innerhalb der MAX Programmierumgebung.
Es soll nicht verschwiegen werden, dass eine verwandte *Library* für MAX nämlich MAXSCORE[1] zeitlich schon vor *bach* im „Rennen" war.
MAXSCORE wurde von NICK DIDKOVSKY[2] in der *Java Music Specification Language (JMSL)*[3] gemeinsam mit GEORG HAJDU[4] entwickelt.

Zumindest für diese Arbeit fällt die Entscheidung für *bach* aus. *bach* ist Meister im Umgang mit *Listen* und liefert eine mehr als komplett wirkende Sammlung von überaus nützlichen *Externals* und *Abstractions*, welche auch für alle anderen Bereiche im Umgang mit MAX nützlich sind, mit.

Wesentlich für die Darstellung der Notation sind die OBJECTS *bach.score* und *bach.roll*.



bach.score liefert die erwartete Notation am Bildschirm.



bach.roll präsentiert das Darzustellende in einer *proportionalen*[5] Notation.

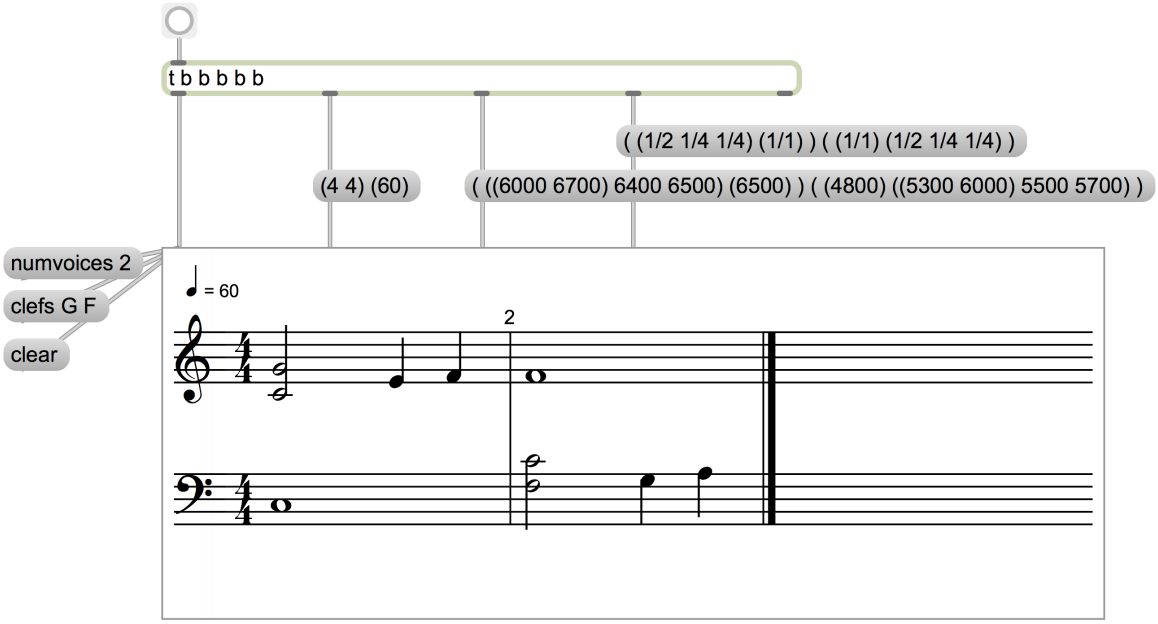
[1] http://www.computermusicnotation.com (Jänner 2015).
[2] NICK DIDKOVSKY ist Gitarrist, Komponist und Programmierer aus Amerika. http://www.didkovsky.com (Jänner 2015).
[3] http://www.algomusic.com (Jänner 2015).
[4] GEORG HAJDU ist Professor für Multimedia-Komposition an der *Hochschule für Musik und Theater* in Hamburg. http://georghajdu.de (Jänner 2015).
[5] Bei der *proportionalen Notation* handelt es sich um eine horizontale Platzverteilung, die jeder Note genau den ihrer Dauer entsprechenden Platz zuweist.

Zwei neue Datentypen mussten eingeführt werden. Zum einen die *Rationale Zahl* mit dem OBJECT *bach.ratnum* und zum anderen die *llll*s.
Rationale Zahlen sind unumgänglich, wenn es sich um *relative* Notenwerte handelt, und sehr nützlich, wenn Verhältnisse oder proportionale Berechnungen zu bewältigen sind.

bach.ratnum

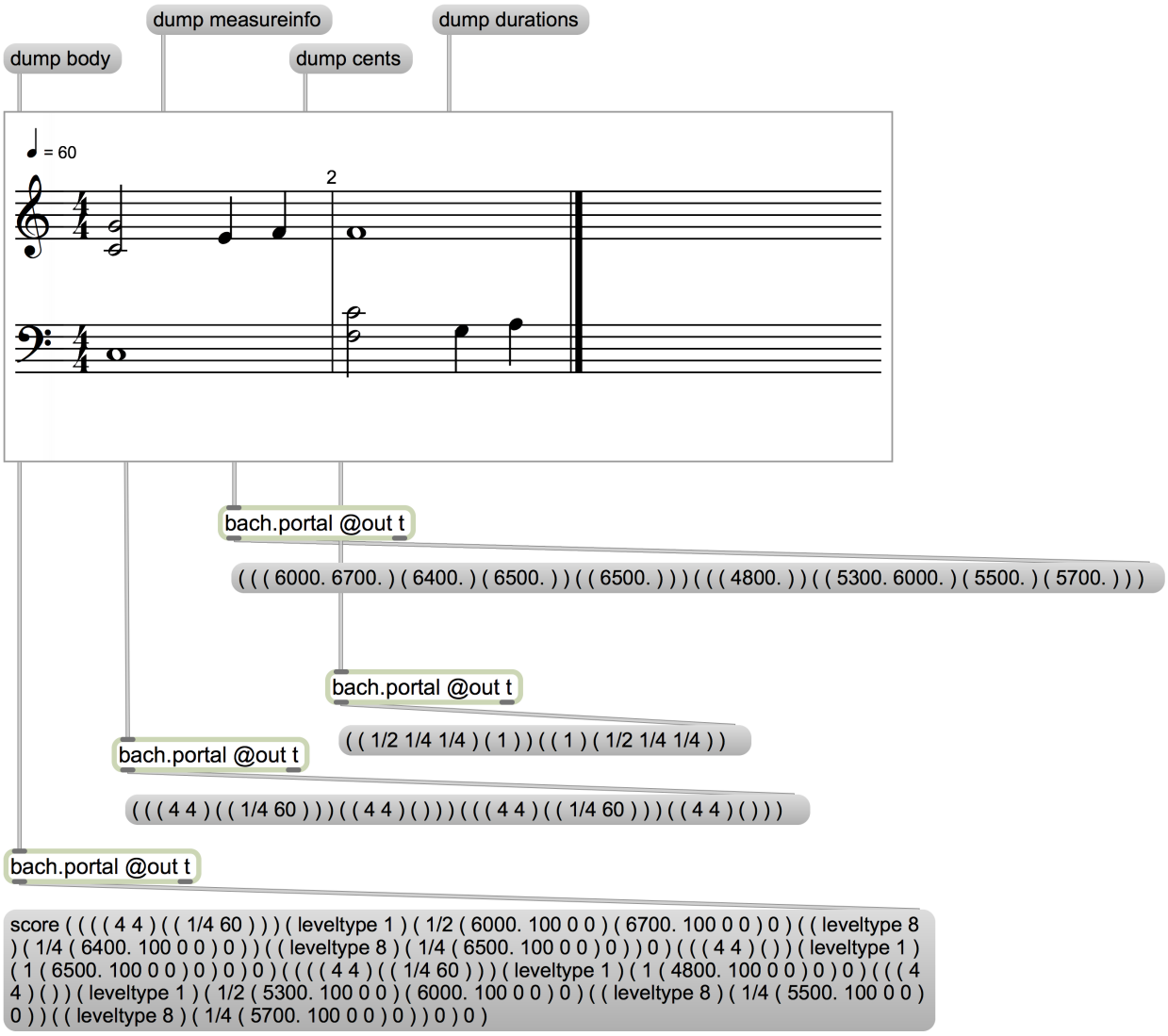
*llll*s sind *Lisp-like linked lists* und für das Kombinieren und Verschachteln von *Listen* – wie wir es schon von LISP kennen – zuständig.

Diese neuen Datenformate in MAX bringen die komplexe *Listen*-Berechnung auf einen Höhepunkt. Ganze Partituren werden damit als verschachtelte *Listen* repräsentiert.

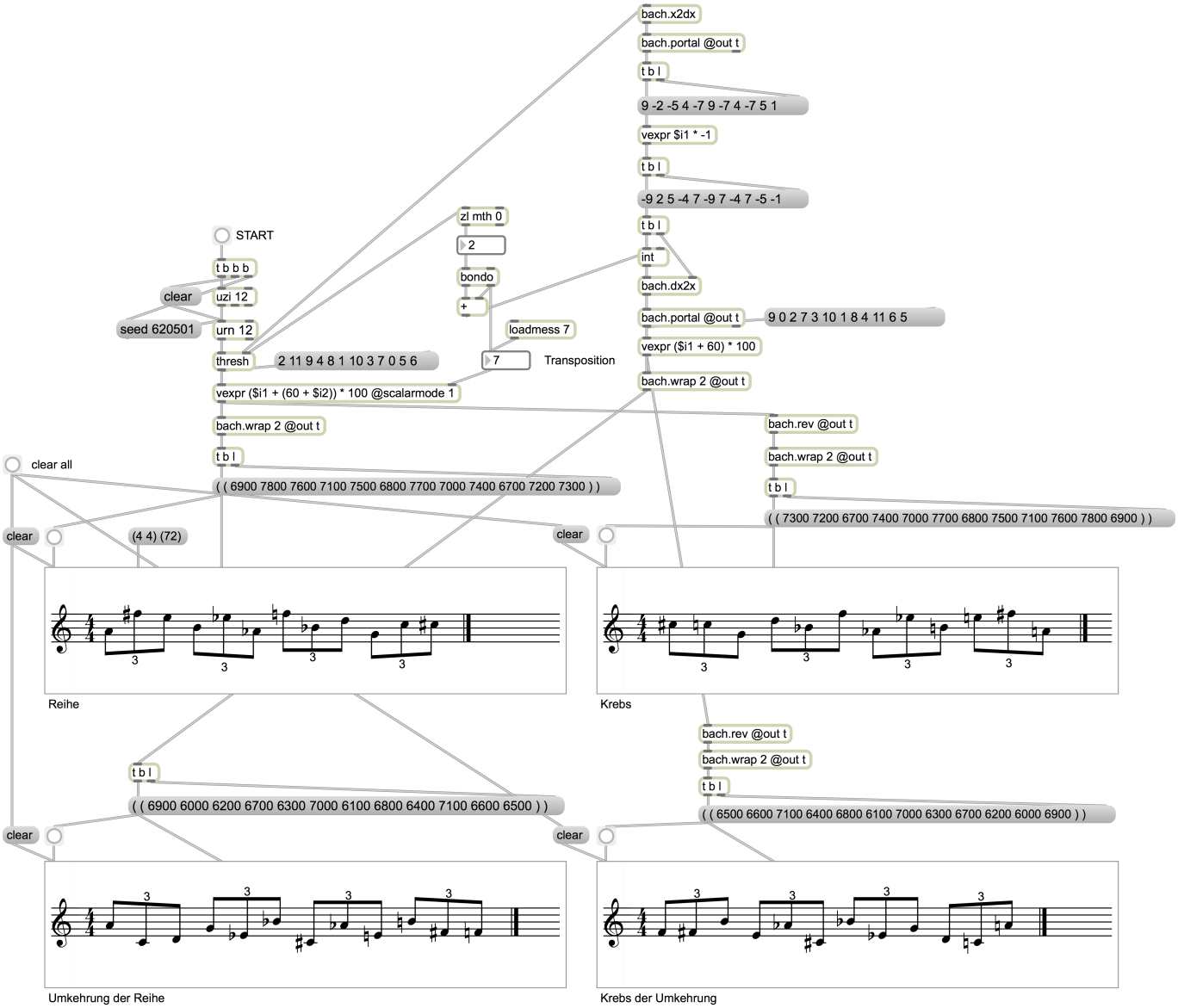


Hier wird auch deutlich, warum die *Listen*-Operationen und das „Jonglieren" mit ihnen im Umgang mit *bach* innerhalb MAX so wesentlich sind.

Inhalte aus *bach.score* können mit *dump-messages* bei den zuständigen *outlets* ausgelesen und gegebenenfalls weiter verarbeitet werden.

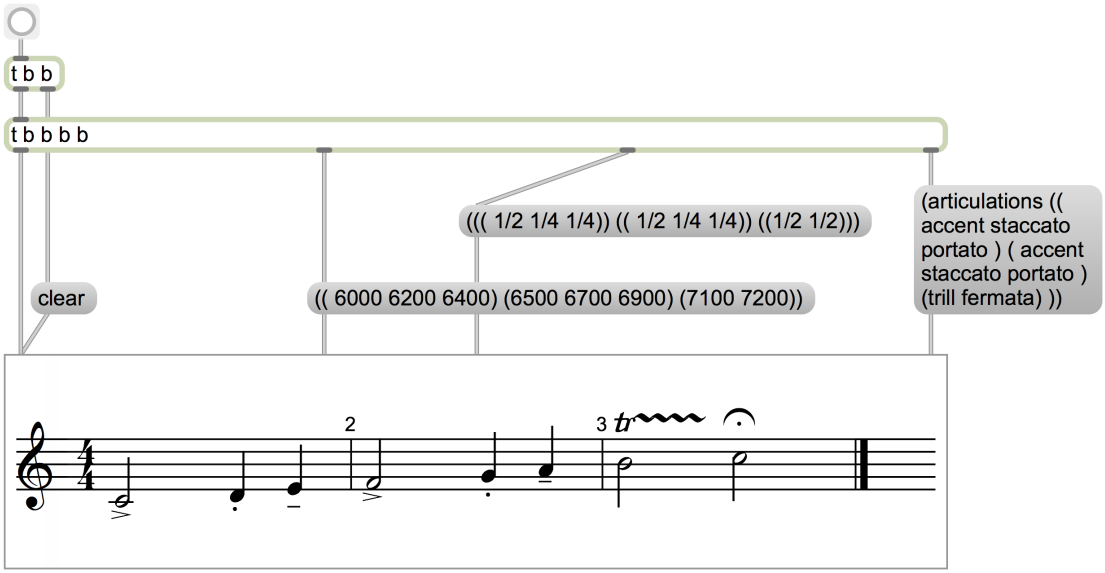


Um Berechnungen durchführbar und überschaubar zu halten, sind spezielle *bach*-Objects mit ihren genau definierten *Messages* großteils notwendig oder zumindest sehr nützlich.



In diesem Beispiel wird eine *Zwölftonreihe* von der *Generierung* bis zu den üblichen *Transformationen* wie *Transposition, Krebs* und *Umkehrung* sowie *Krebs der Umkehrung* gezeigt.

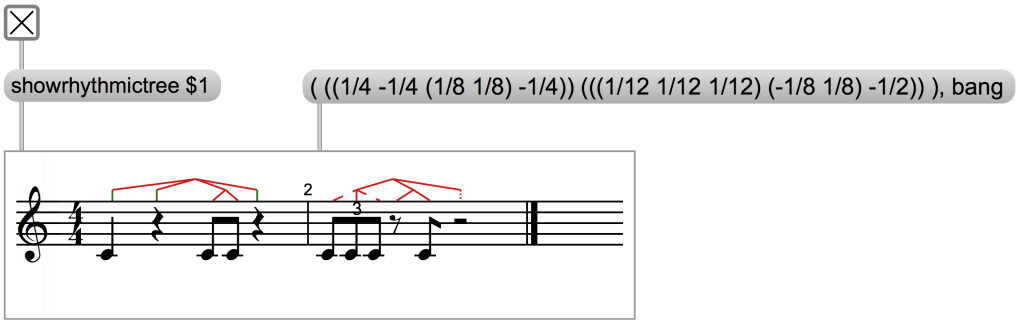
bach versteht sich auch ganz gut mit Artikulationszeichen.



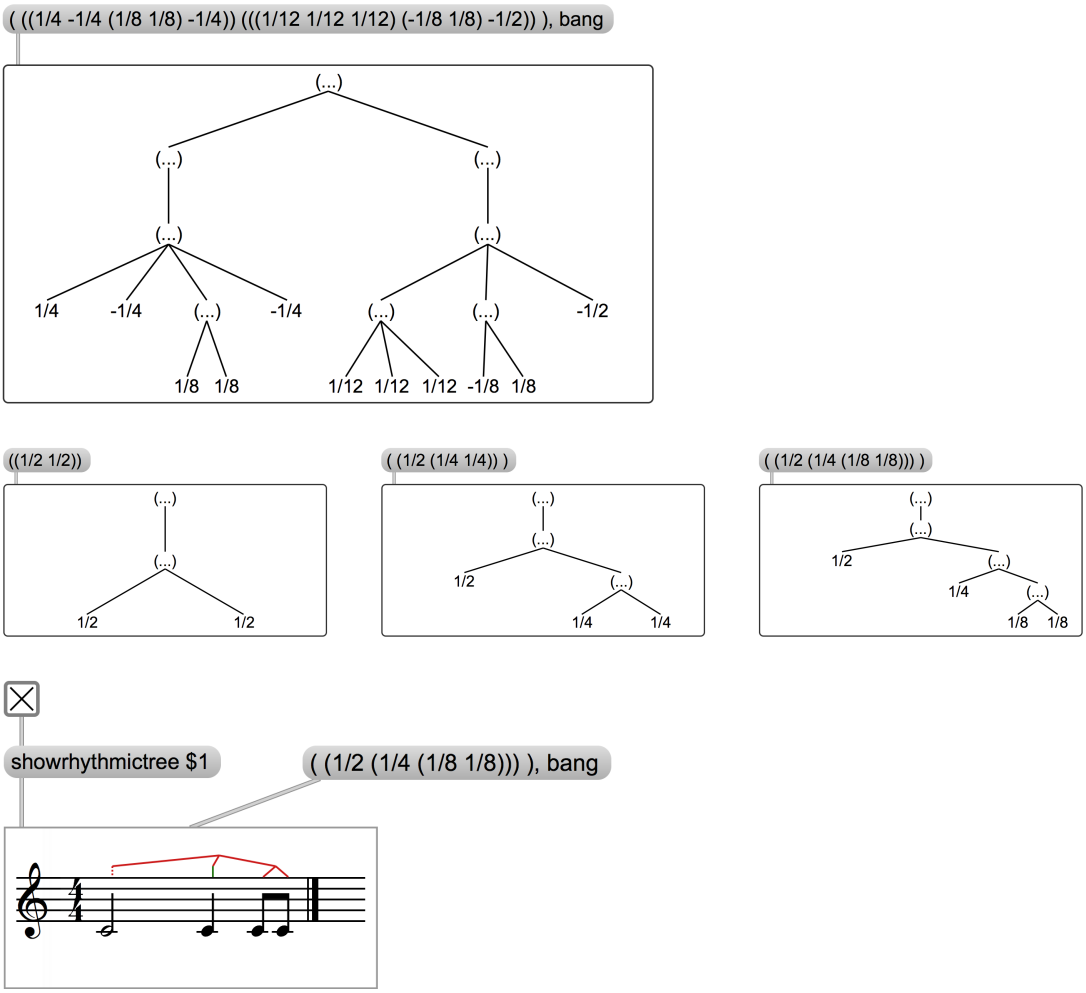
Leider sind derzeit (*bach* Version 0.7.7b) noch keine Legato-Bögen möglich. Zur Verfügung stehen:
staccato, staccatissimo, fermata, portato, accent, accent+staccato, accent+portato, portato+staccato, martellato, martellato+staccato, left hand pizzicato, trill, gruppetto, upward mordent, downward mordent, double mordent, upward bowing, downward bowing, 3-slash tremolo, 2-slash tremolo, 1-slash tremolo.

Wie in den Beispielen zu sehen ist, werden die Tonhöhen in *Midicents*[1] und die Notenwerte in Bruchzahlen angegeben. Für Pausen wird ein "-" vor die Bruchzahl gesetzt.

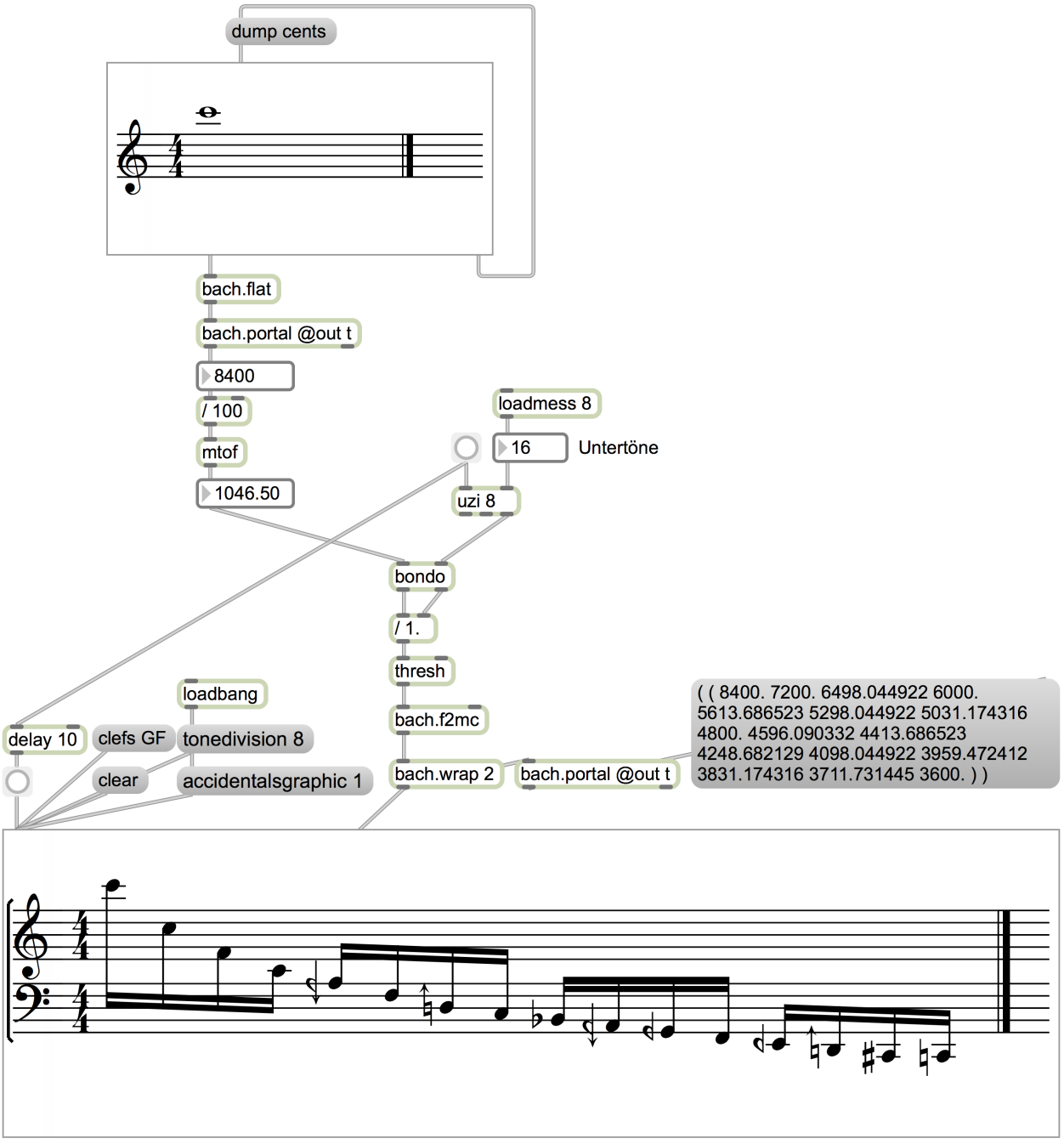
[1]Hier ist die Midi-Notennummer auch in der Cent-Einteilung in 1/100 stel-Schritten zu verstehen.



Bei *bach* wurde an sehr sehr vieles gedacht, und viele Raffinessen zur grafischen Darstellung wurden beigegeben. Beispielsweise das OBJECT *bach.tree* zum Darstellen oder auch Editieren der *llll*s.



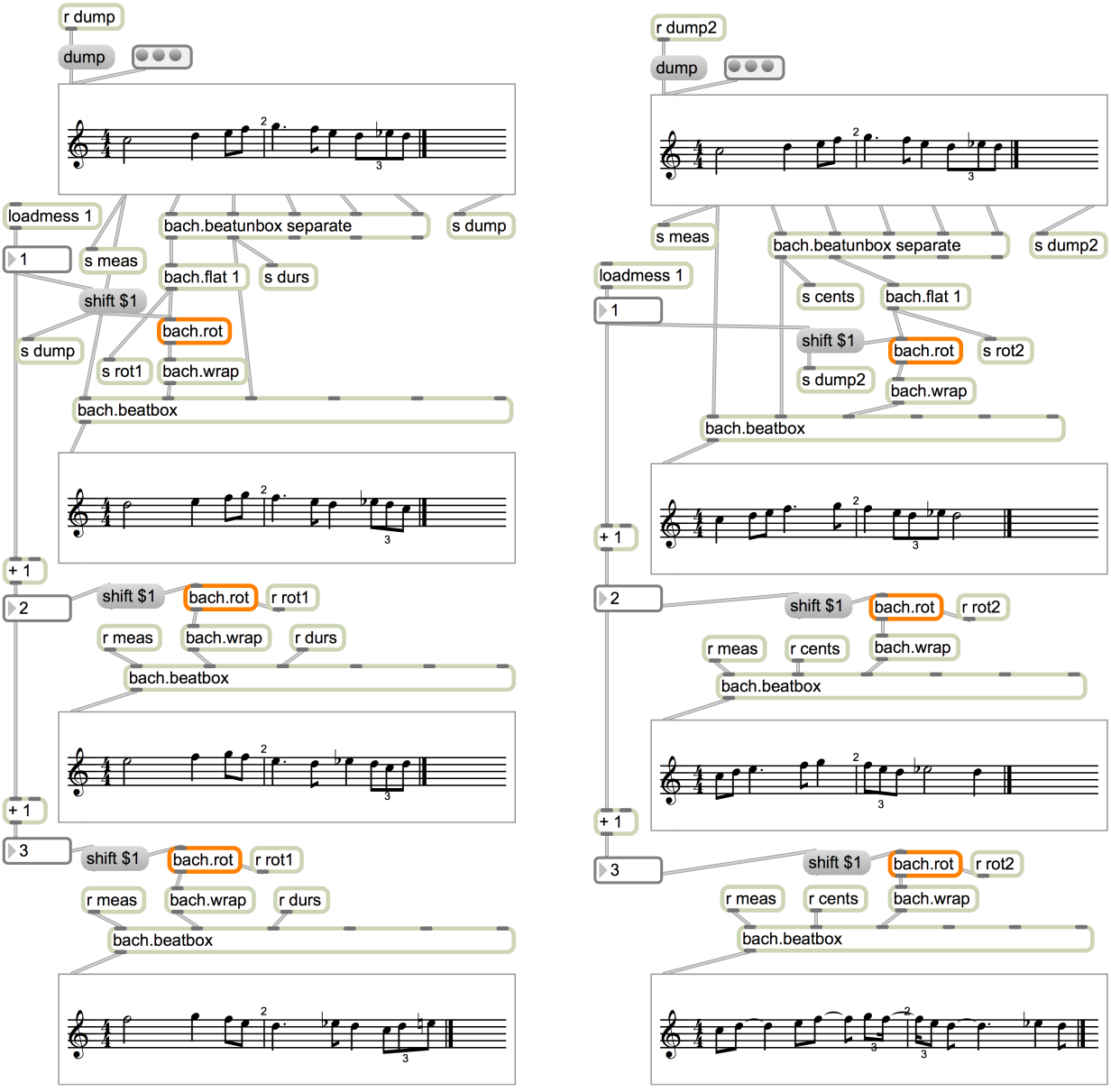
Tonhöhen werden bei Bedarf auch in Mikrointervallen berechnet und in üblicher mikrotonaler Notenschreibweise angezeigt.



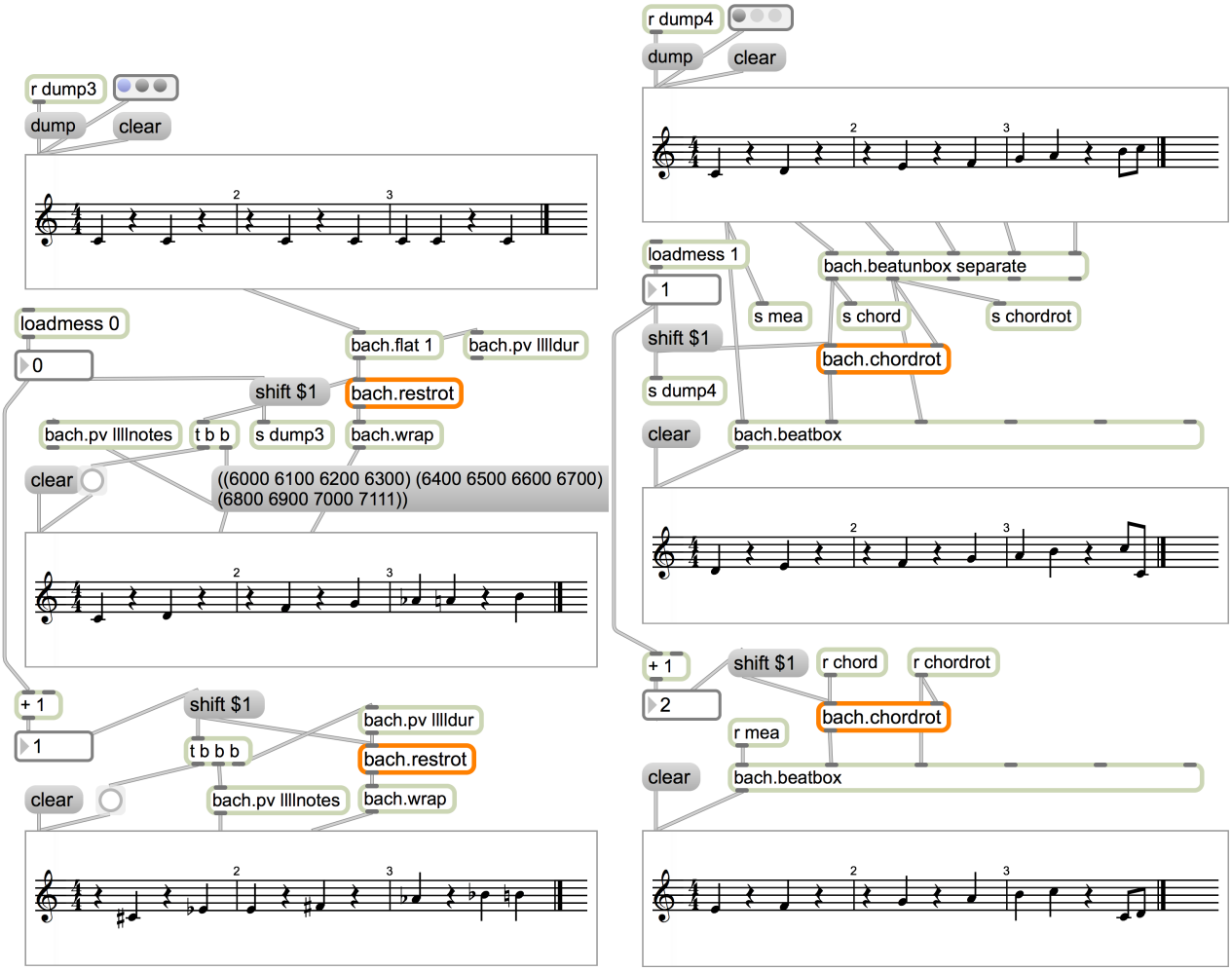
Stellvertretend für eine Fülle spezieller *Listen*-manipulierender *bach*-OBJECTS möchte ich drei vorstellen:

bach.rot bach.restrot bach.chordrot

bach.rot führt eine Rotation von *llll*s mit ihren Sub-*Listen* aus.
bach.restrot rotiert die negativen Vorzeichen – also die Pausen – durch die *Liste*.
bach.chordrot rotiert nicht negative Elemente – also alles, was nicht Pause ist.

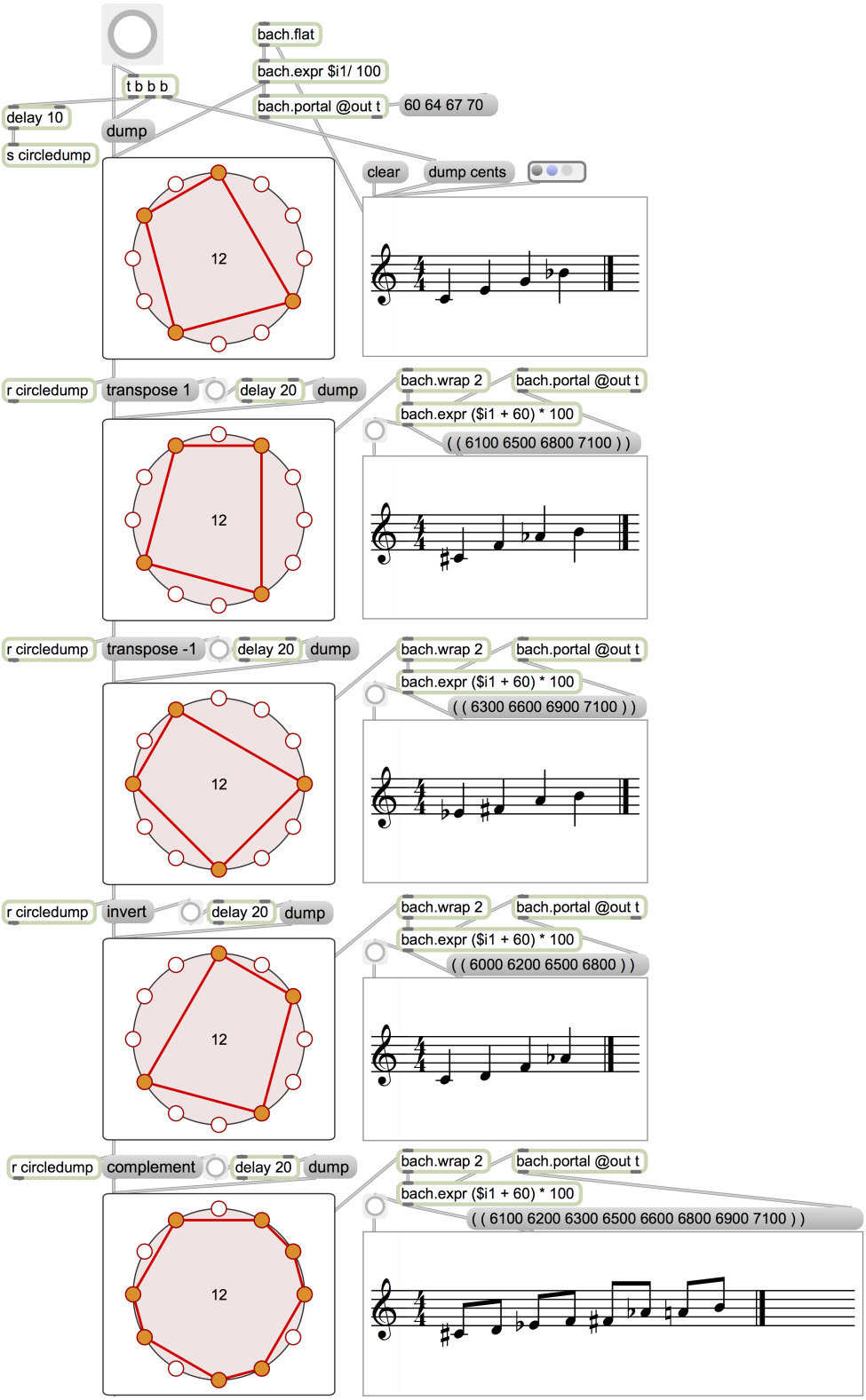


Links im Beispiel werden die Tonhöhen und rechts die Notenwerte rotiert.



In diesem Beispiel werden links die Pausen und rechts die Nicht-Pausen rotiert.

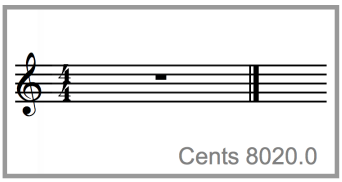
User-Interface-OBJECTS können zur Veranschaulichung dienen, wenn es um Themen der *Pitch-Class-Theory* geht.



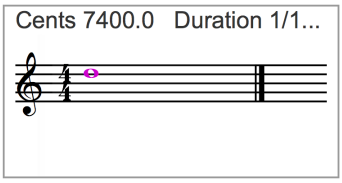
Die Notation in *bach* erfolgt entweder in *Echtzeit* durch Generierung und Umwandlung von *Listen* beziehungsweise *Listen* von *Listen* oder durch direkte Eingabe in die *bach.score*- oder *bach.roll*-OBJECTS.

Bei direkter Eingabe stehen viele Tastaturkommandos oder Mausaktionen – ähnlich jener, die in üblichen Notations-Programmen, beispielsweise bei FINALE[1] oder SIBELIUS[2], gefunden werden – zur Verfügung.

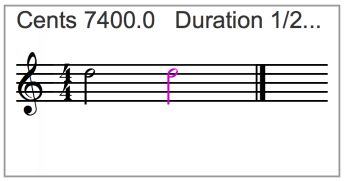
bach.score-Eingabe



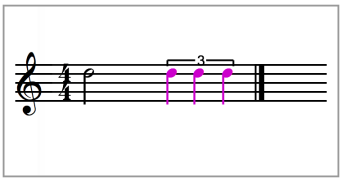
Durch *Cmd+Klick* (bei OSX) oder *Ctrl+Klick* (bei Windows) in das *bach.score*-OBJECT wird eine ganze Pause eingefügt.



Durch *Cmd+Klick* auf die Pause wird diese in eine Note umgewandelt (oder auch umgekehrt).

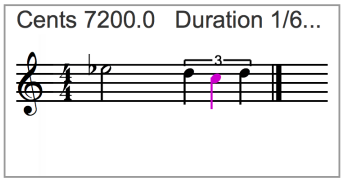


Durch Anwählen oder Selektieren einer Note (es ändert sich dabei die Farbe der Note) und Ausführen eines *Cmd+2*-Befehls werden die Notenwerte aufgeteilt. Das funktioniert bis *Cmd+9*.

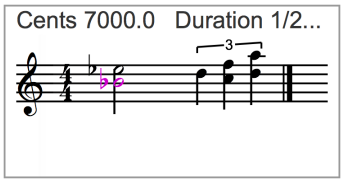


Hier das Resultat aus Anwählen der zweiten Note und Ausführen eines *Cmd+3*-Befehls.

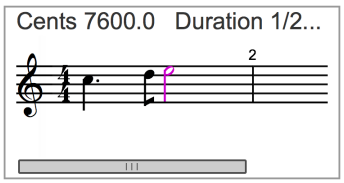
[1] https://www.finalemusic.com (Jänner 2015).
[2] http://www.sibelius.at (Jänner 2015).



Durch Anwählen von Noten und Betätigen der Pfeiltasten werden diese entsprechend auf- oder abwärts transponiert.

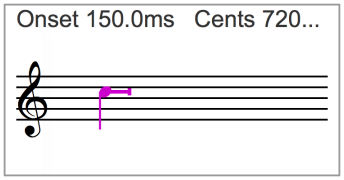


Durch Anwählen von Noten, Betätigen der *Alt*-Taste und Ziehen mit der Maus werden Noten vervielfacht und transponiert.

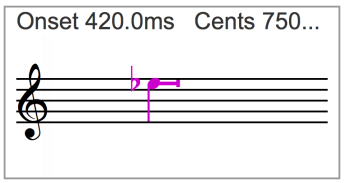


Durch einen *Ctrl+Klick* gelangt man in den *linear editing mode*. Es erscheint ein *pitch cursor*, der mit den Pfeiltasten bewegt werden kann. Notenwerte werden mit den Zahlentasten hinzugefügt. Eine 1/4tel-Note wird mit der Taste 5 eingefügt – so, wie wir es von FINALE kennen. (Eine 1/2-Note mit 6, eine 1/8tel-Note mit 4 und so weiter) Mit dem Punkt "." wird punktiert und mit der t-Taste wird eine Ligatur erzeugt. Mit der ESC-Taste gelangt man wieder in den normalen Editier-Modus zurück.

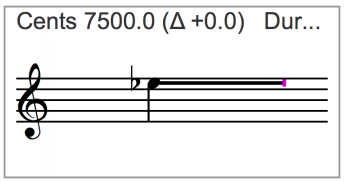
bach.roll-Eingabe



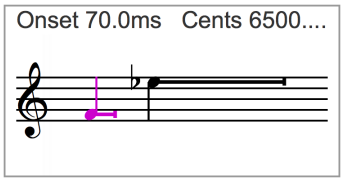
In *bach.roll* wird mit *Cmd+Klick* an derselben Stelle eine Note positioniert.



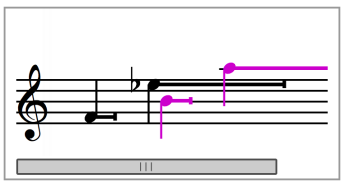
Durch Anwählen und Ziehen der Note mit der Maus wird ihre zeitliche Position oder/und ihre Tonhöhe geändert.



Durch Anwählen des Längenbalkens und durch Ziehen mit der Maus wird die Notenlänge verändert.

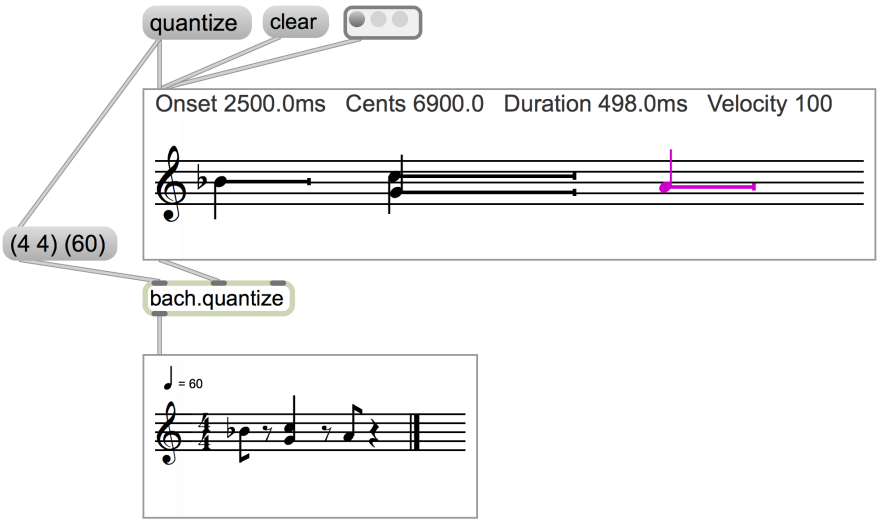


Durch *Cmd+Klick* werden Noten hinzugefügt.

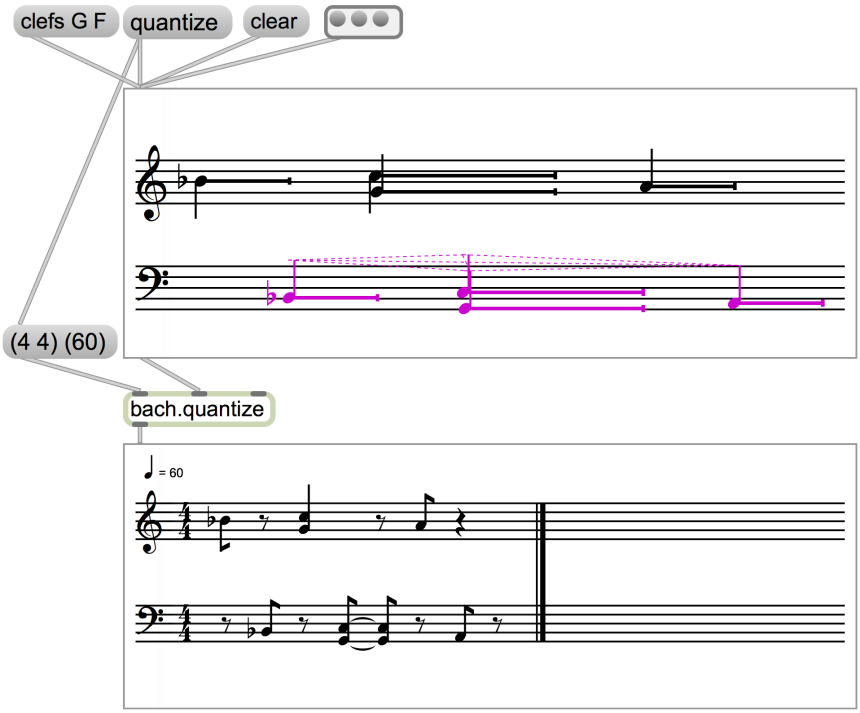


Durch Anwählen von Noten + Ziehen mit der *Alt*-Taste werden Noten oder Notengruppen bewegt und vervielfältigt.

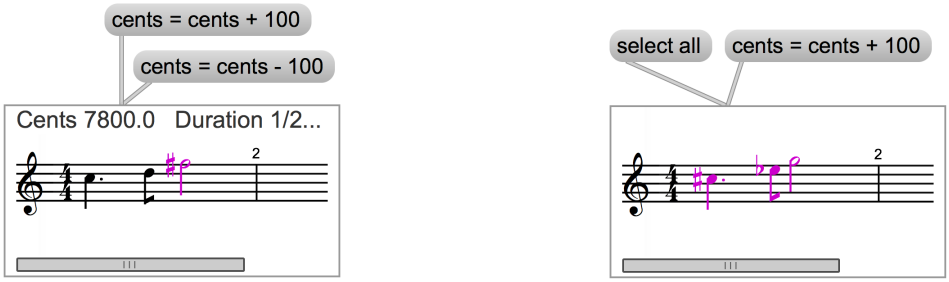
Die proportionale Darstellung in *bach.roll* wird mit dem *bach.quantize*-OBJECT in eine traditionelle Darstellung in *bach.score* umgewandelt.



Mit der Taste G können Noten zu Gruppen zusammengefasst und gemeinsam bewegt oder manipuliert werden.

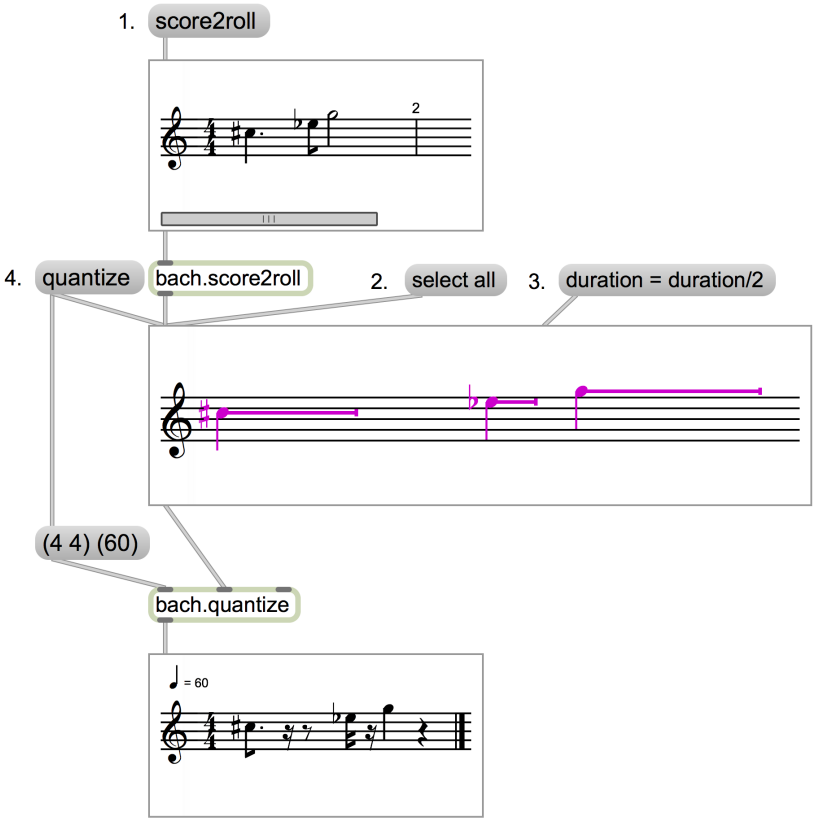


Noten werden mit der Maus oder mit differenzierten *Messages* angewählt und mit Tastatur- und/oder Maus-Kommandos oder wiederum mit speziellen *Messages* manipuliert.

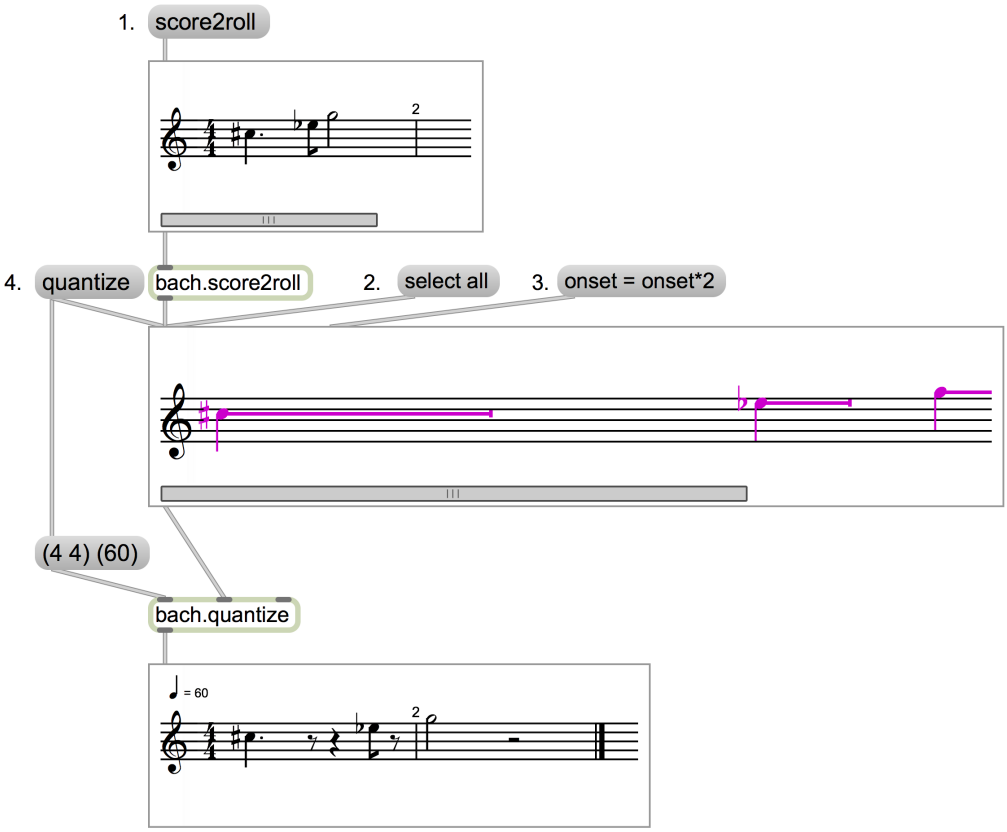


Bei den nächsten beiden Beispielen wird einmal die *duration* der Noten halbiert und einmal die *onset*-Zeit verdoppelt.

Halbieren der Duration

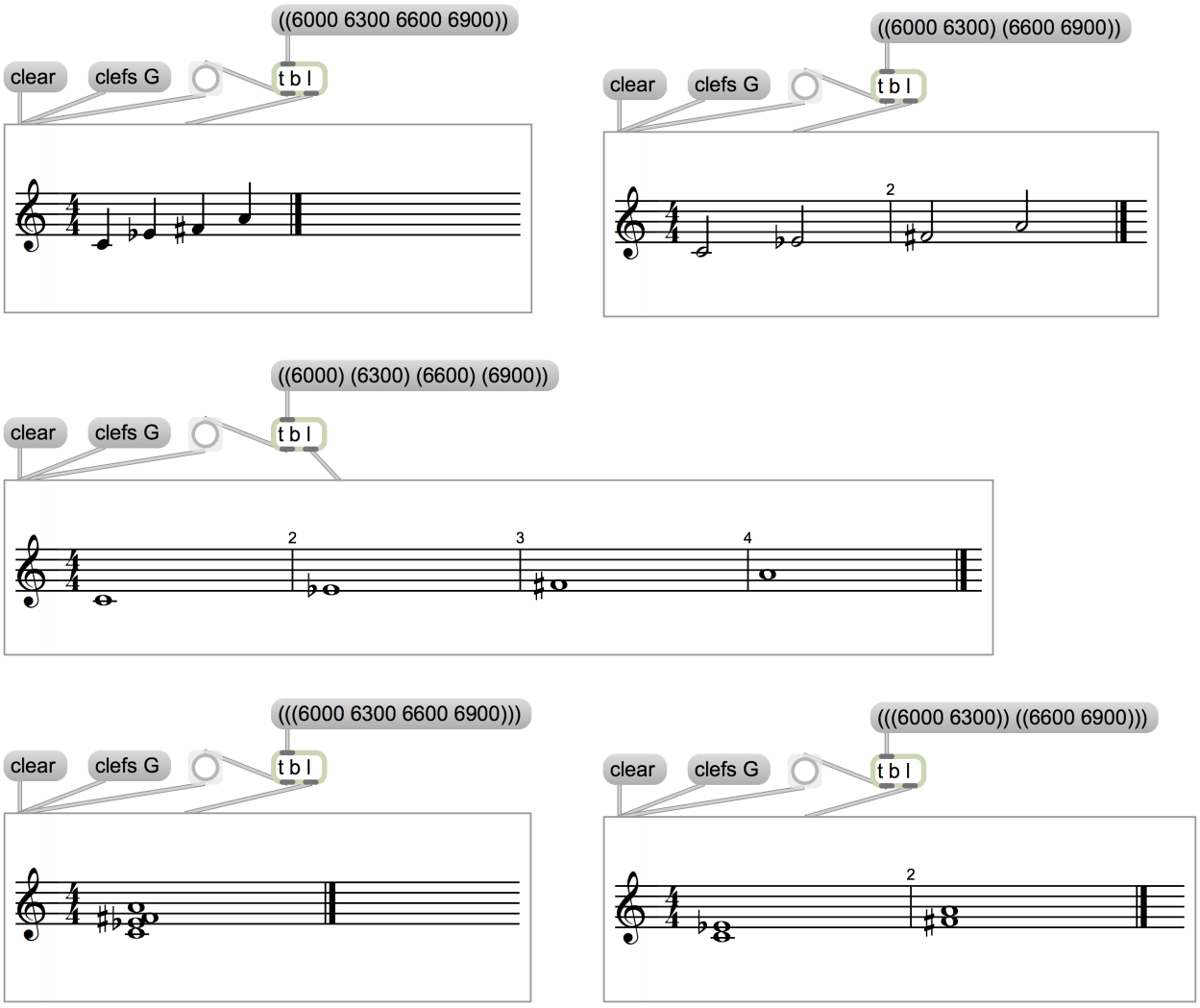


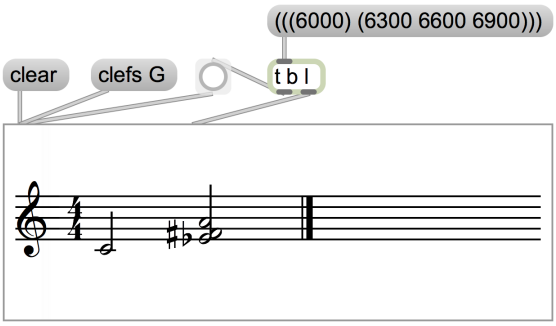
Verdoppeln der Onset-Zeit

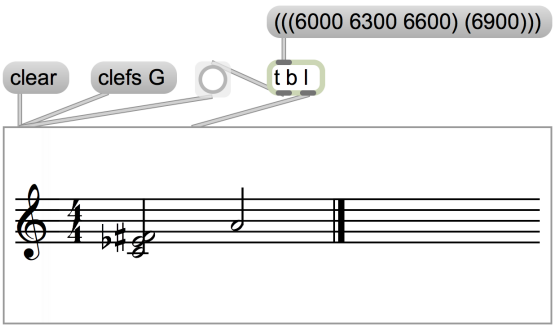


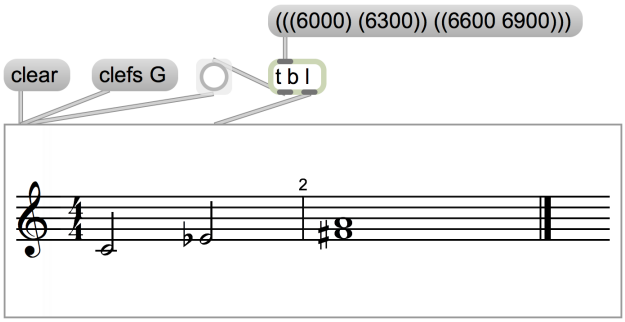
Klammern

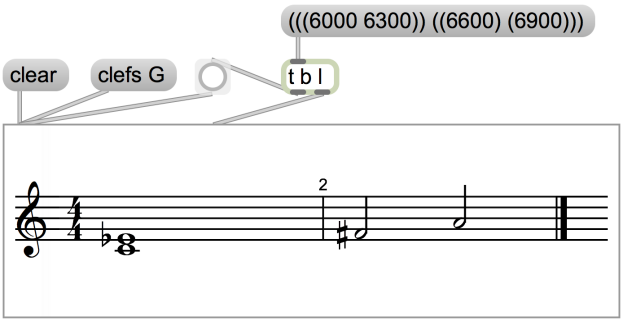
Wesentlich bei allen *bach*-„Operationen" ist die Beschaffenheit der *Listen* – im Besonderen die gezielte Platzierung der Klammern.

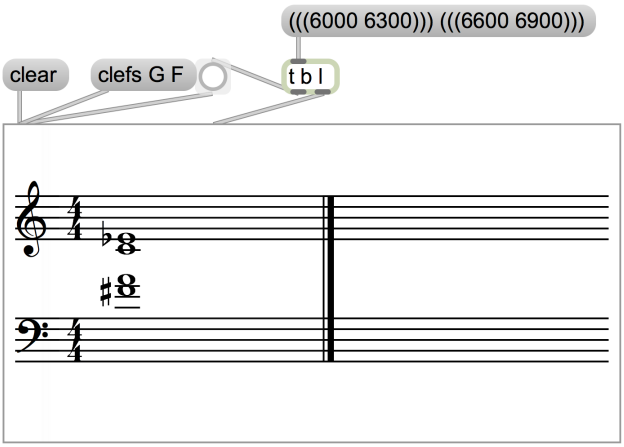


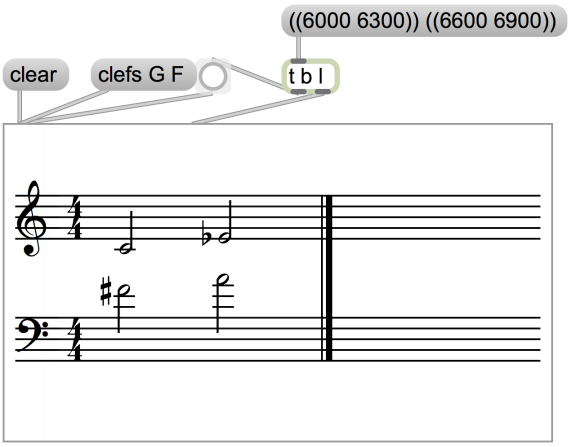

(((6000) (6300 6600 6900)))
clear
clefs G
t b l

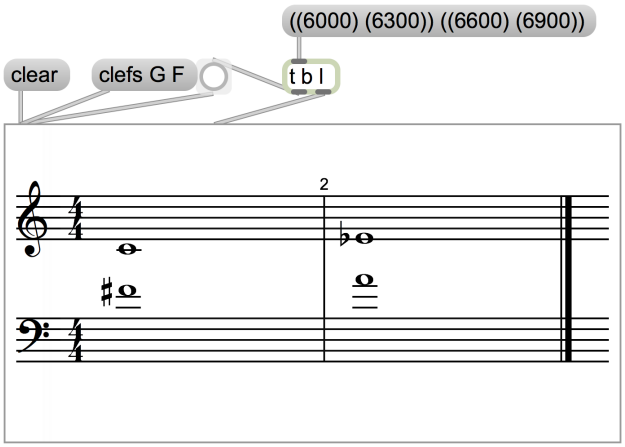

(((6000 6300 6600) (6900)))
clear
clefs G
t b l

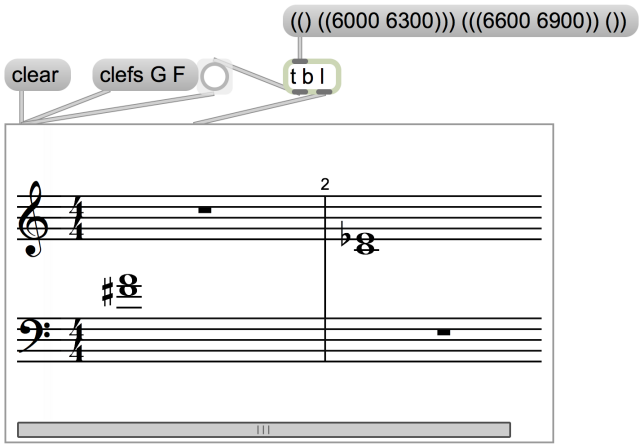

(((6000) (6300)) ((6600 6900)))
clear
clefs G
t b l
2

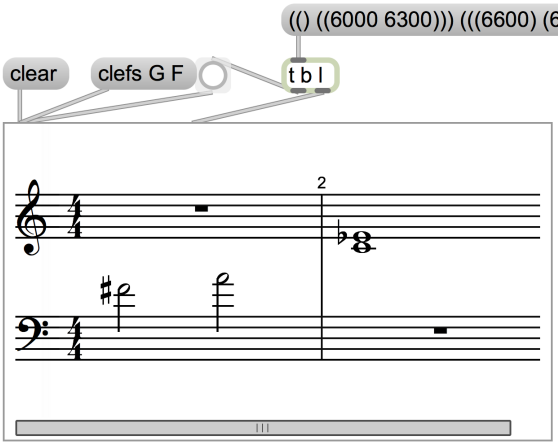

(((6000 6300)) ((6600) (6900)))
clear
clefs G
t b l
2

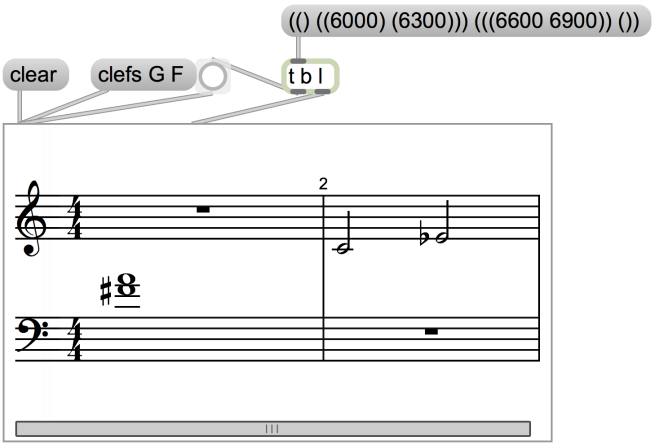

(((6000 6300))) (((6600 6900)))
clear
clefs G F
t b l

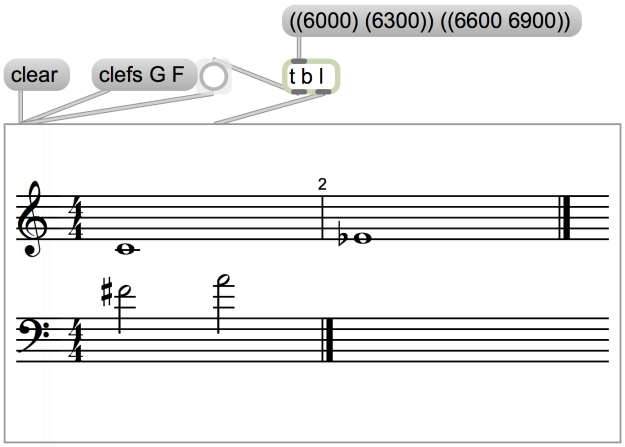

((6000 6300)) ((6600 6900))
clear
clefs G F
t b l


((6000) (6300)) ((6600) (6900))
clear
clefs G F
t b l
2

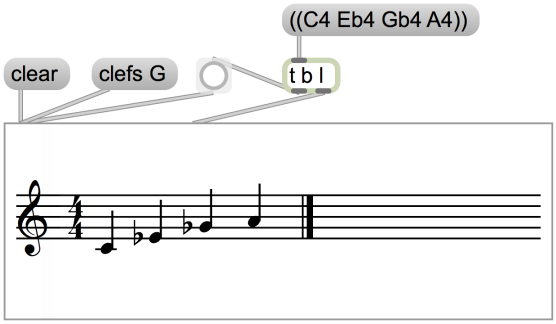

(() ((6000 6300))) (((6600 6900)) ())
clear
clefs G F
t b l
2


(() ((6000 6300))) (((6600) (6900)) ())
clear
clefs G F
t b l
2

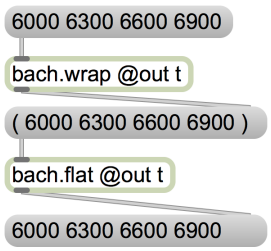

(() ((6000) (6300))) (((6600 6900)) ())
clear
clefs G F
t b l
2


((6000) (6300)) ((6600 6900))
clear
clefs G F
t b l
2

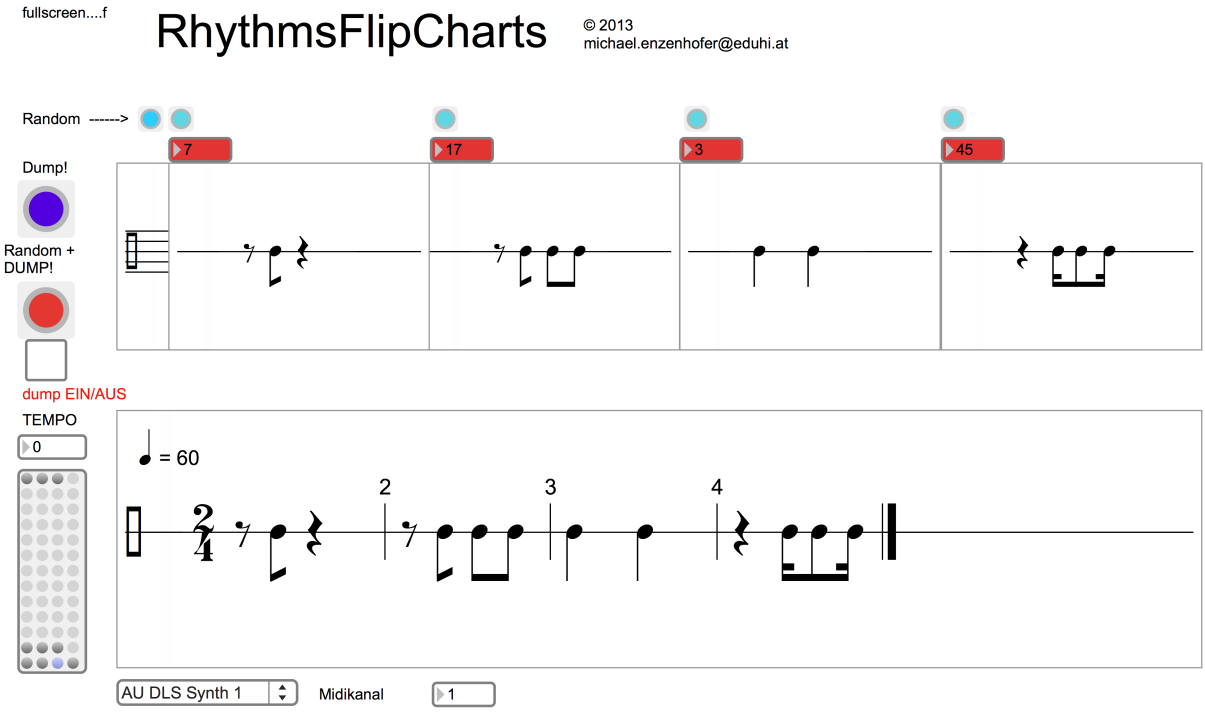
bach.roll- sowie *bach.score*-OBJECTS verstehen anstelle der Angabe in *Midicents* für die Tonhöhen auch Notennamen.



Zum Platzieren der Klammern sind die OBJECTS *bach.wrap* und *bach.flat* eine große Erleichterung:



Abschließend ein kleines Beispiel mit *bach* in MAX erstellt: RhythmsFlipCharts setzt sich aus vier 2/4-Takten zusammen. Jeder Takt beinhaltet 50 idente mögliche Rhythmen, die per Zufall ausgewählt werden. Das ergibt pro „Knopfdruck" eine von $50\ x\ 50\ x\ 50\ x\ 50 = 6\,250\,000$ – man kann es fast nicht glauben – möglichen 4-Takt-Gruppen. Für den Benützer (*User*) dieses MAX-*Patches* wurde der *Präsentationsmodus* gewählt, das heißt, man sieht nur die *UserInterface*-Objects, die für die Verwendung notwendig sind. Die Programmierung mit den *bach*-Objects und ihren Verbindungen und Vernetzungen bleiben verborgen.



4.2 bach-slots

Viele wichtige *bach*-Objects und Einstellungsmöglichkeiten in *bach* sind bisher noch nicht behandelt. Der Einfachheit halber wurde nicht ausgeführt, dass neben den *inlets* für *Measureinfo, Cents* und *Durations* natürlich auch ein *inlet*

für *Velocities* vorhanden ist.

Steuerungsmöglichkeiten, wie sie von üblichen *Sequenzern* geläufig sind – etwa Abspielen, Pausieren, Tempoveränderungen, Loops und so weiter – sind Selbstverständlichkeiten.

In den Notensystemen können auch *Marker* gesetzt werden, um Positionen zielsicher und zeitlich exakt anzuspielen. Den einzelnen Notensystemen werden Midi-Kanäle zugeordnet, die – wie sonst auch üblich – die Klänge innerhalb des Computers oder externe Klangerzeuger anspielen.

Eine Besonderheit für Liebhaber der Komplexität stellt das *Extras-inlet*, welches auch die Artikulations-Angaben aufnimmt, dar. Dieses *inlet* versteht auch sogenannte *slot*-Informationen. Bis zu 35 *slots* können jeder einzelnen Note zugewiesen werden. Diese *slot*-Informationen bewirken einerseits die Darstellung beliebiger Klangveränderungen im Zeitverlauf – beispielsweise *Hüllkurven* – und andererseits die Ausgabe der zugewiesenen Parameter beim Abspielen der Notation.

Jede Note fasst also bis zu 35 *slots*. Wobei jedem *slot* eine von 16 Kategorien (*types*) zugewiesen werden kann.

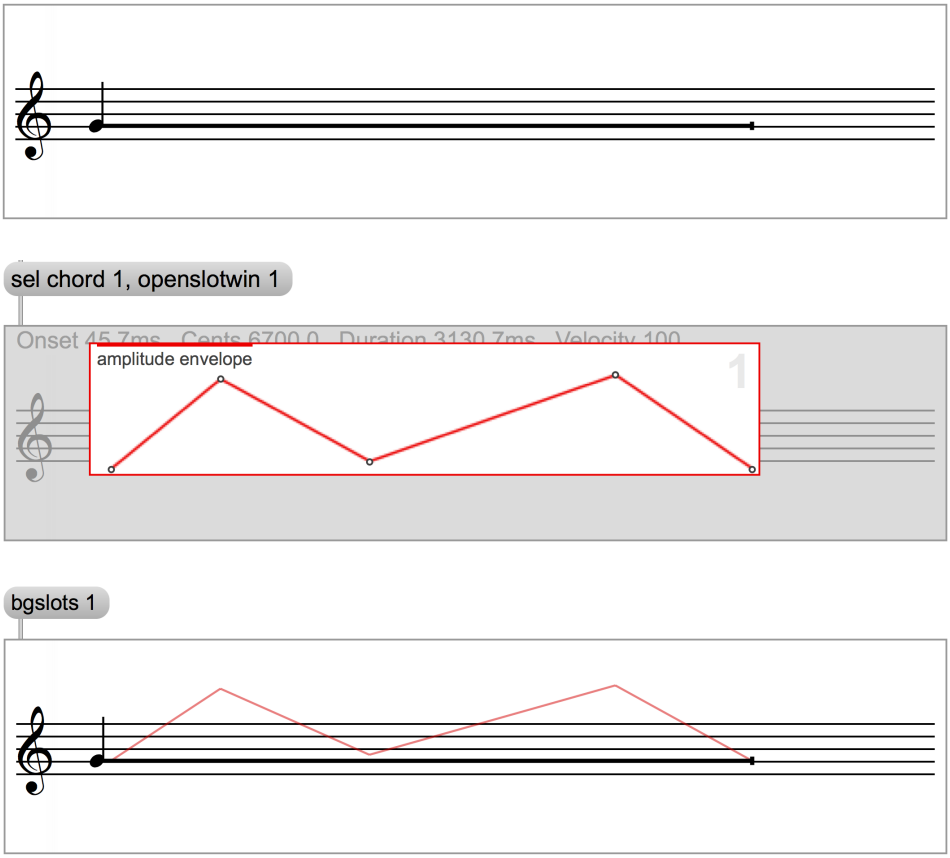
- *int* — Zum Darstellen und Ausgeben von einzelnen ganzen Zahlen.
- *float* — Zum Darstellen und Ausgeben von einzelnen Fließkommazahlen.
- *intlist* — Zum Darstellen und Ausgeben von *Listen* von ganzen Zahlen.
- *floatlist* — Zum Darstellen und Ausgeben von *Listen* von Fließkommazahlen.
- *function* — Zum Darstellen und Ausgeben von linearen Funktionskurven.

Durch die Angabe eines *slope*-Werts kann auch jedes lineare Segment zu einer Kurve werden.

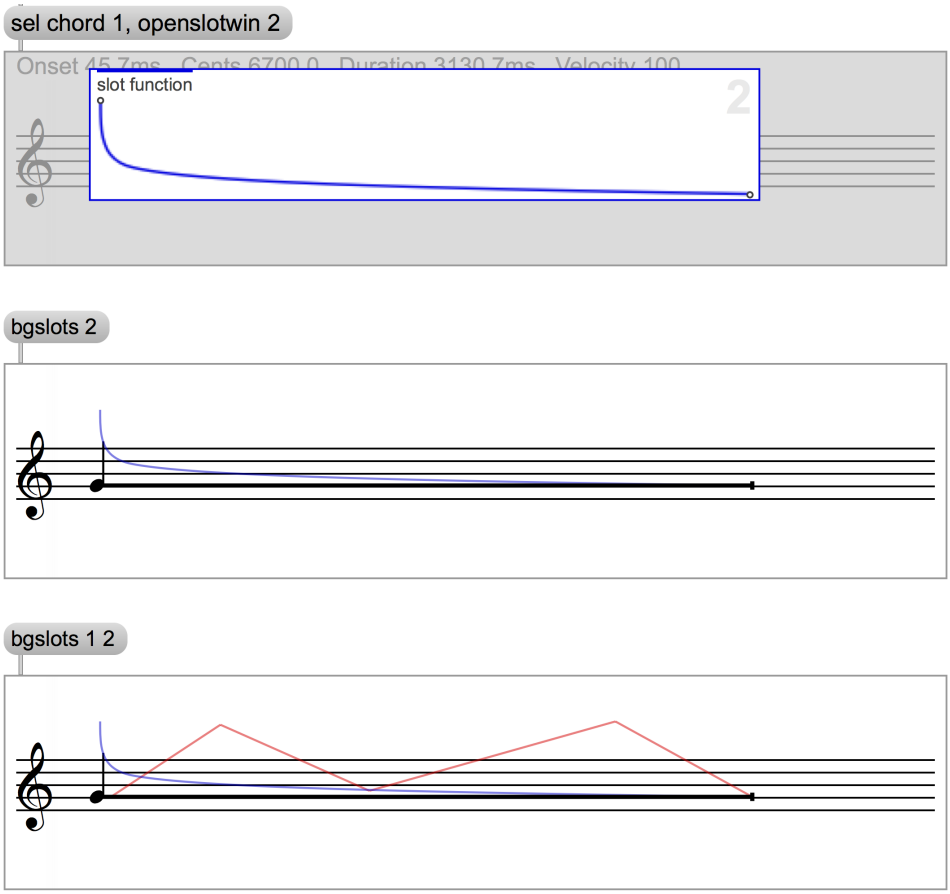
- *spat* — Zum Darstellen und Ausgeben von *Panorama*-Werten.
- *text* — Beliebiger Text zur einzelnen Note.
- *llll* — Ganze *bach*-Partituren – auch kleinere *Listen* von *Listen* – können in einer Note verpackt werden.
- *filelist* — Informationen über *Files* auf der Computer-Festplatte.
- *color* — Farbdarstellung im *RGBA* – Modus.
- *3dfunction* — Lineare Kurve mit x y z – Koordinaten für jeden Punkt. Auch hier sind zusätzliche *slope*-Werte durch einen vierten Parameter pro Funktions-Punkt möglich.
- *filter* — Einstellbare Filtertypen: *lowpass*, *highpass*, *bandpass*, *bandstop*, *peaknotch*, *lowshelf*, *highshelf* und *allpass*. Frequenzangabe in Hertz und *Gain* in Dezibel. Q ist der *resonance*-faktor oder *slope*-faktor – abhängig vom Filtertyp.
- *dynfilter* — Beinhaltet Informationen über die zeitliche Abfolge von *interpolated biquad filters*. Diese Filter-Parameter sind in erster Linie zur internen MAX (vormals MSP)-Weiterverwendung nützlich.
- *togglematrix* — Zahlen in Reihen und Spalten organisiert. Jeder Eintrag als *Toggle*-Information – Ein oder Aus.
- *intmatrix* — Jeder Matrizen-Eintrag als ganze Zahl.
- *floatmatrix* — Jeder Matrizen-Eintrag als Fließkommazahl.

Willkommen in der Zukunft.

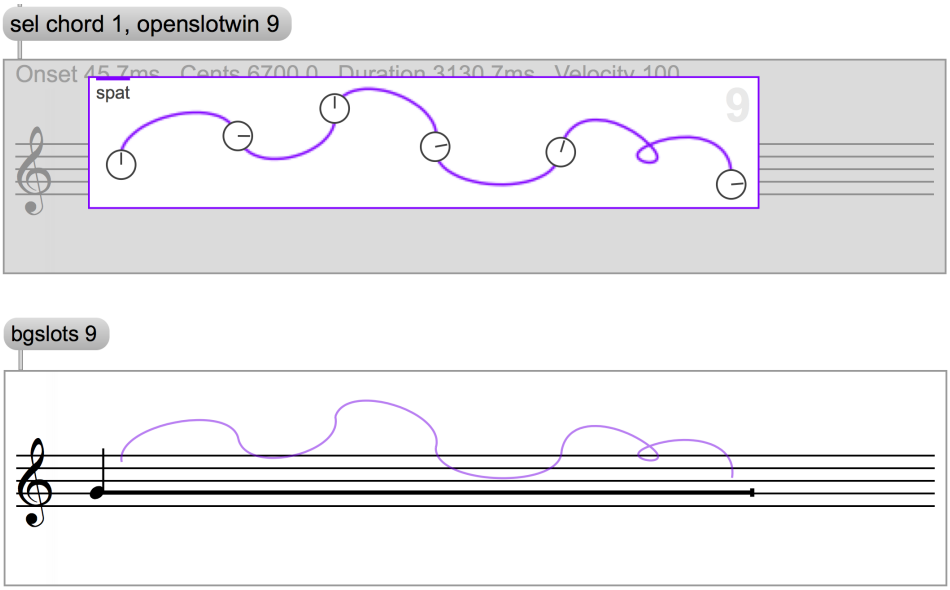
Man darf gespannt sein, wie diese neuen flexiblen Möglichkeiten der Notation von Musik-Programmierern aufgenommen werden. Und ob sich ihre Ergebnisse wiederum auf die traditionelle Notation auswirken werden.



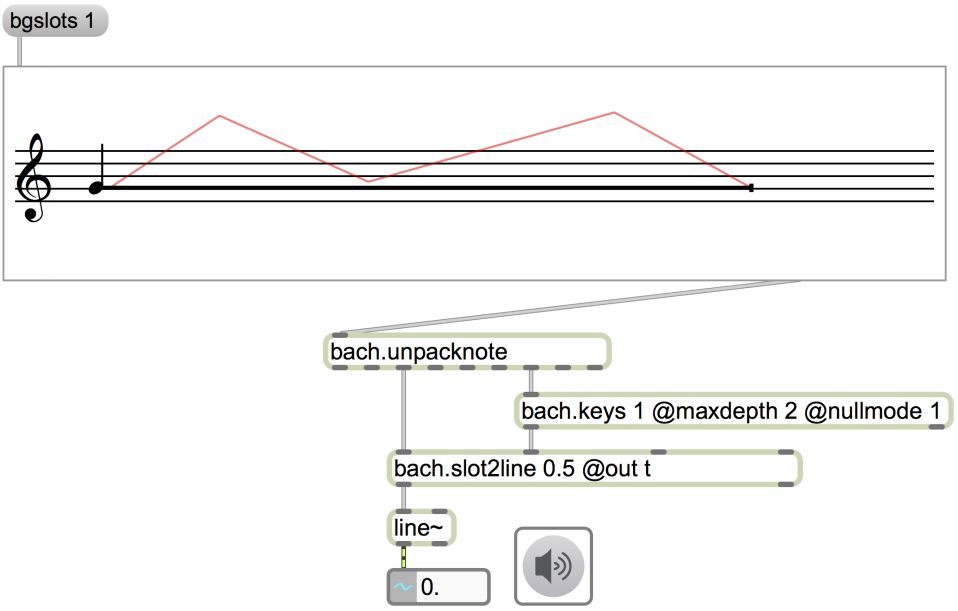
Durch Öffnen eines *slot*-Fensters (Anwählen einer Note + definiertes Tastaturkürzel) können je nach *type* die Funktionsparameter eingegeben werden. Natürlich ist es wieder möglich, sämtliche Parameter auch mit *Messages* festzulegen.

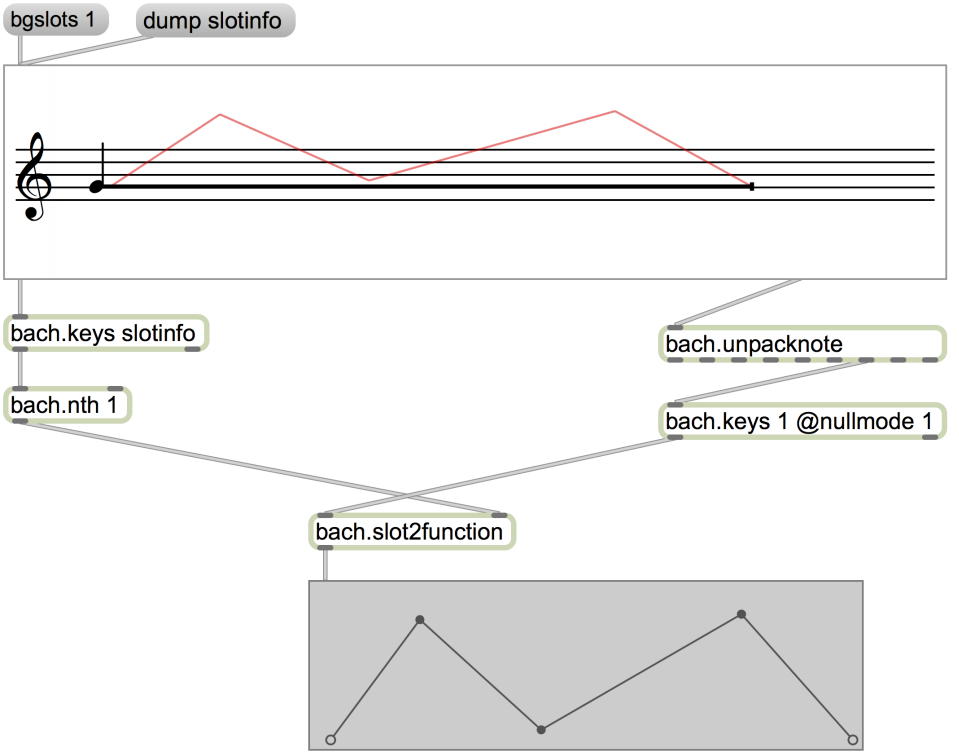


Durch die *Message bgplots*+Zahl(en) werden die entsprechenden *slots* auch im Notenbild dargestellt.

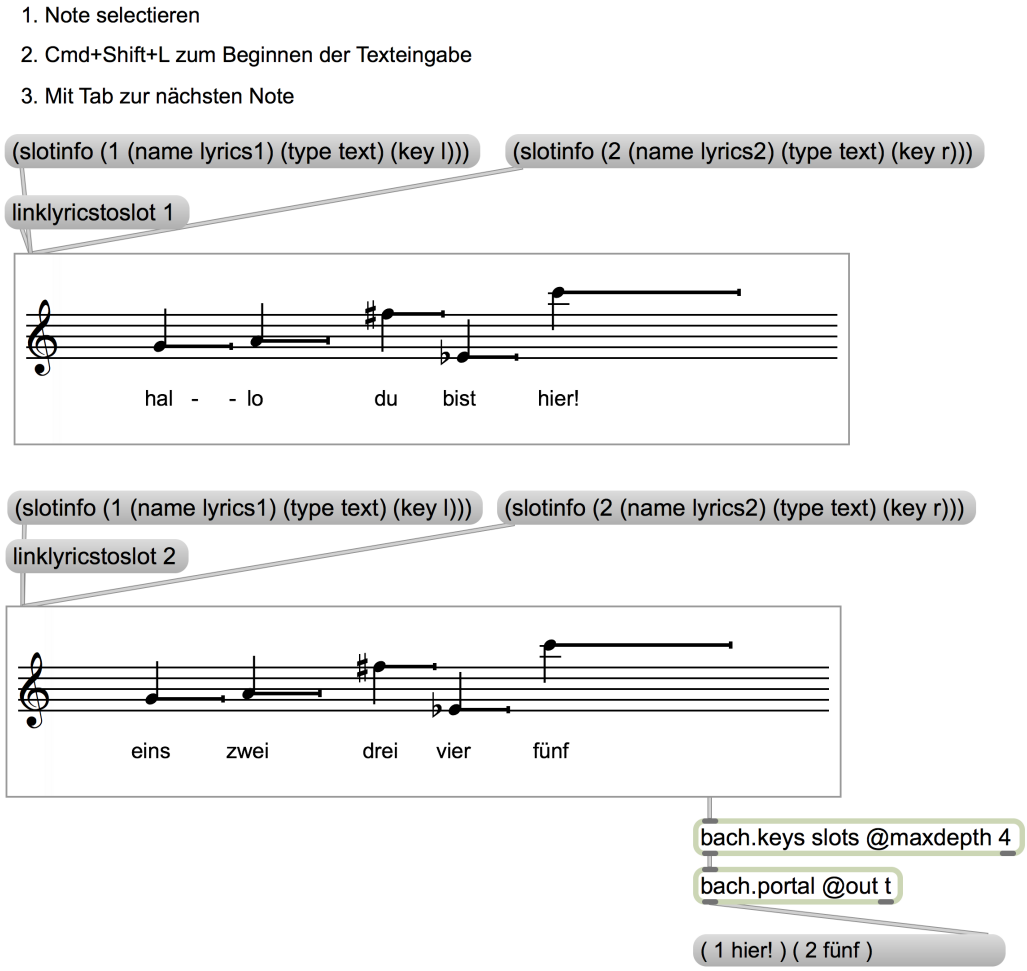


Die Daten der *slots* werden mit geeigneten OBJECTS „abgefangen“ und der weiteren Prozessierung zur Verfügung gestellt.





Selbst der Umgang mit Text beziehungsweise *Lyrics* wird in *bach* mit *slots* gelöst:

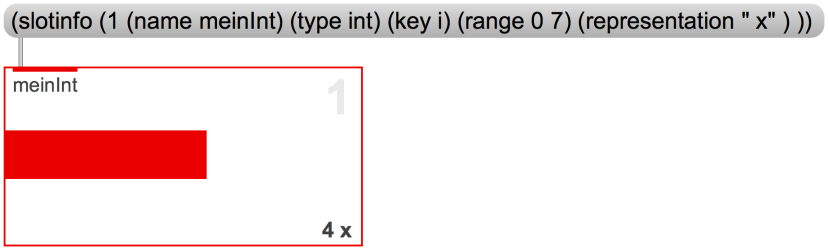


Praktisch können auch mehrere *slots* für Text bereitgestellt werden.
Mit der *Message linklyricstoslot*+Zahl wird der entsprechende *slot* mit dem dazugehörigen Text im Notenfenster von *bach.score* oder *bach.roll* angezeigt und bei Bedarf auch ausgegeben.

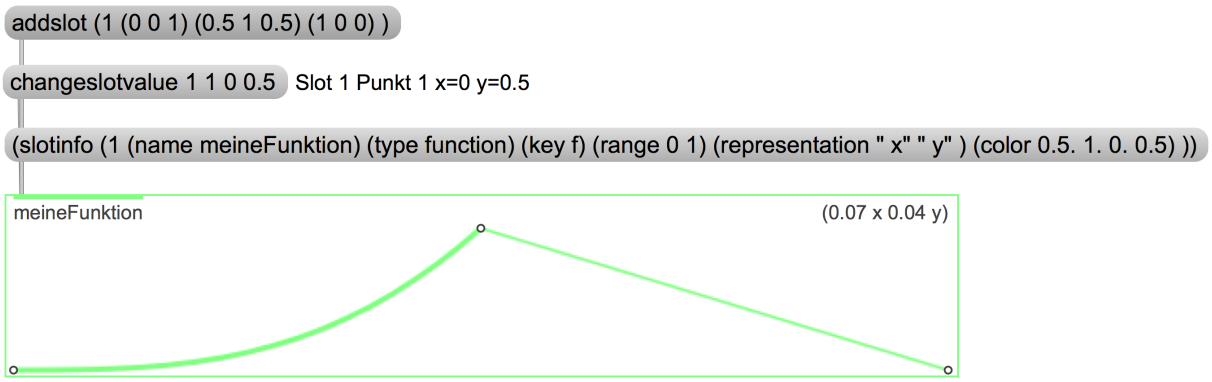
Die *slots* werden neben der *Message* für die *slot*-Nummer mit

- frei zu wählenden Namen (*name*),
- der Kategorie (*type* von *int* bis *llll*),
- dem frei zu wählenden Tastenkürzel zum Aufrufen des *slot*-Fensters (*key*),
- dem Wertebereich (*range* – je nach *type* unterschiedlich),
- der Repräsentanz der Werte (*representation* – zum Beispiel Maßeinheiten oder Ähnliches
- oder mit der Farbe des *slot*-Fensters

definiert.



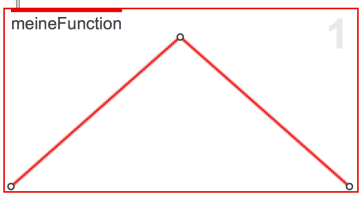
Neben der direkten Eingabe im *slot*-Fenster wird mit den *Messages adds-lot* oder *changeslotvalue* die Werteingabe und -veränderung unterstützt.



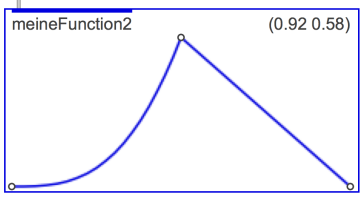
Abschließend

noch eine Übersicht mit *Minimal*-Beispielen der verschiedenen *slot*-Variationen in eigenen *slot*-OBJECTS.
Die ersten zehn *slots* sind in ihrer Kategorie vordefiniert – der Musik-Programmierer darf aber auch frei über jeden *slot* verfügen.

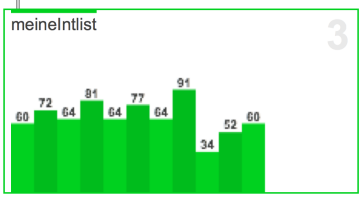
addslot (1 (0 0 0) (0.5 1 0) (1 0 0))
(slotinfo (1 (name meineFunction) (type function) (range 0 1) (key f)))



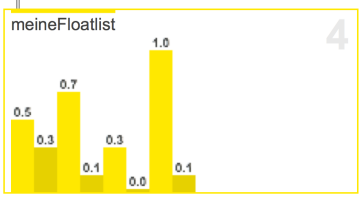
addslot (2 (0 0 0) (0.5 1 0.5) (1 0 0))
(slotinfo (2 (name meineFunction2) (type function) (range 0 1) (key f)))



addslot (3 60 72 64 81 64 77 64 91 34 52 60)
(slotinfo (3 (name meineIntlist) (type intlist) (range 0 127) (key I)))

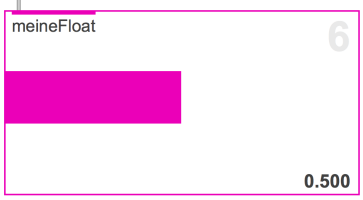


addslot (4 0.5 0.3 0.7 0.1 0.3 0. 8 0.1)
(slotinfo (4 (name meineFloatlist) (type floatlist) (range 0 1) (key F)))



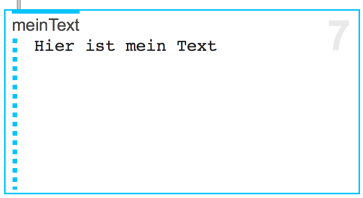
addslot (5 60)
(slotinfo (5 (name meineInt) (type int) (range 0 127) (key i)))

addslot (6 0.5)
(slotinfo (6 (name meineFloat) (type float) (range 0 1) (key f)))



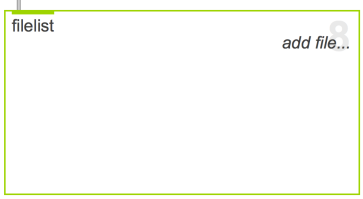
addslot (7 " Hier ist mein Text")

(slotinfo (7 (name meinText) (type text) (key t)))



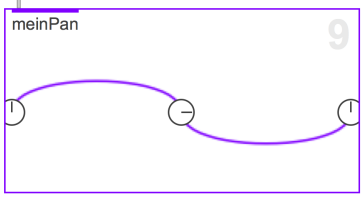
addslot (8)

(slotinfo (7 (name meinText) (type text) (key t)))



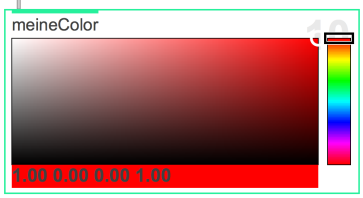
addslot (9 (0 5 0 0) (0.5 5 90 0) (1 5 0 0))

(slotinfo (9 (name meinPan) (type spat) (key p)))



addslot (10 1. 0. 0. 1.)

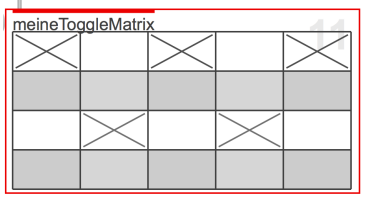
(slotinfo (10 (name meineColor) (type color) (key c)))



addslot (11 ((1 0 1 0 1) () (0 1 0 1 0) ()))

(slotinfo (11 (representation (4 5))))

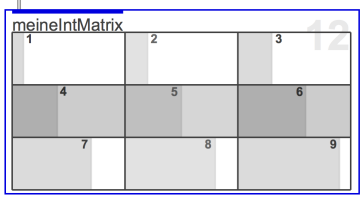
(slotinfo (11 (name meineToggleMatrix) (type togglematrix) (key m)))



addslot (12 ((1 2 3) (4 5 6) (7 8 9)))

(slotinfo (12 (representation (3))))

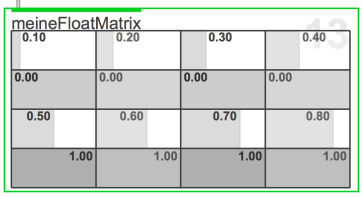
(slotinfo (12 (name meineIntMatrix) (type intmatrix) (range 0 10) (key y)))



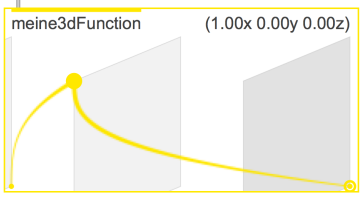
addslot (13 ((0.1 0.2 0.3 0.4) () (0.5 0.6 0.7 0.8) (1. 1. 1. 1.)))

(slotinfo (13 (representation (4))))

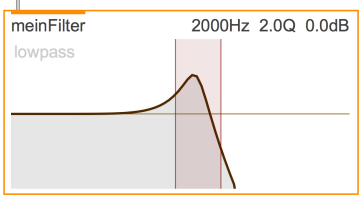
(slotinfo (13 (name meineFloatMatrix) (type floatmatrix) (range 0 1) (key v)))



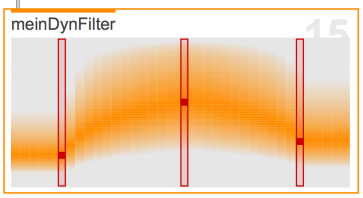
```
addslot (14 (0 0 0 0 ) (0.5 1 1 -0.5) (1 0 0 -0.5))
(slotinfo (14 (representation x y z )))
(slotinfo (14 (name meine3dFunction) (type 3dfunction) (range 0 1) (key z)))
```



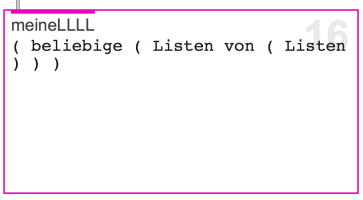

```
addslot (15 lowpass 2000 0. 2.)
(slotinfo (15 (name meinFilter) (type filter) (range -12 12) (key u)))
```




```
(slotinfo (16 (name meinDynFilter) (type dynfilter) (range 0 22050) (key r)))
```

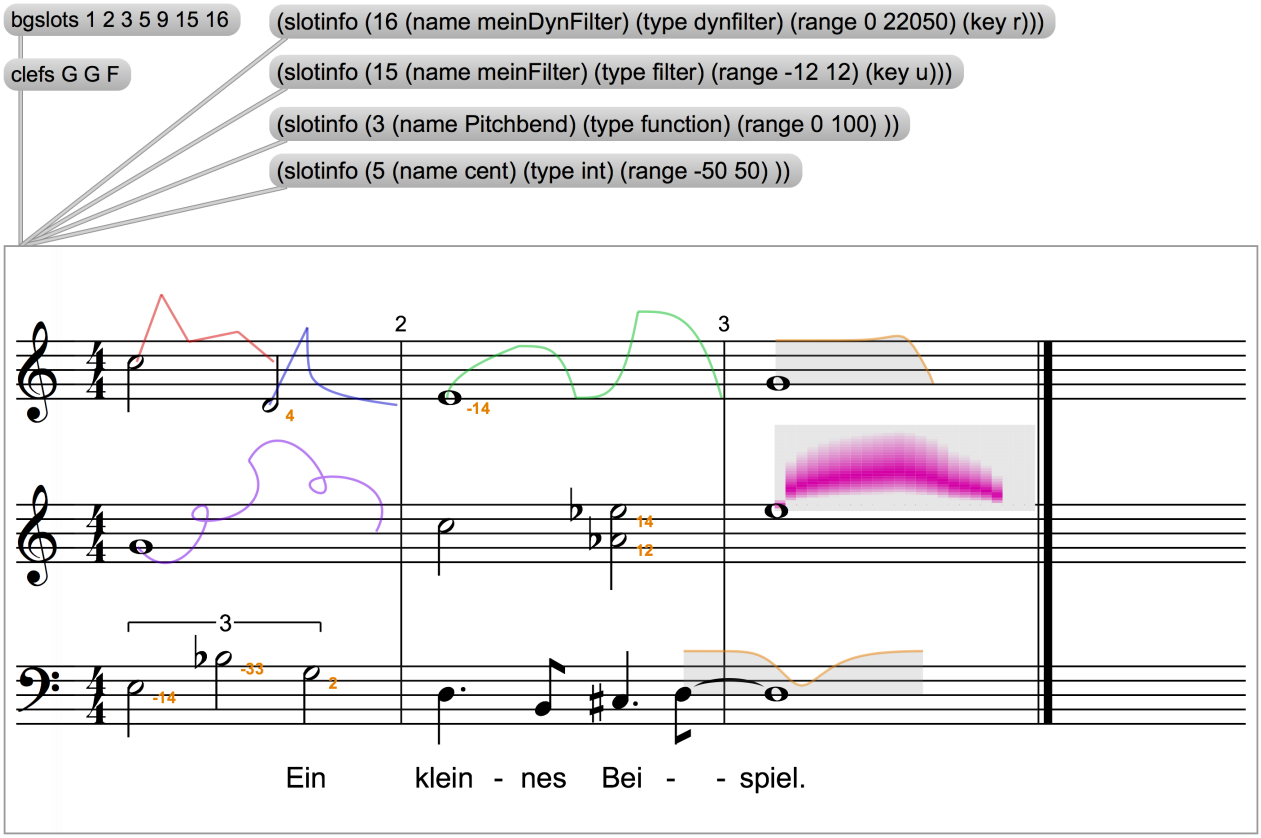



```
addslot ( 17 ((beliebige (Listen von (Listen)))))
(slotinfo (17 (name meineLLLL) (type llll) (key l)))
```



Jeder einzelnen Note stehen sämtliche *slots* zur Verfügung.

Ein zusammenfassendes Beispiel von *slot*-Darstellungsvarianten in *bach.score*:



5 LilyPond

5.1 Überblick LilyPond

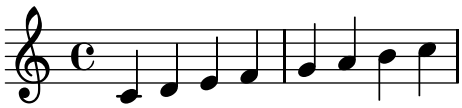
LilyPond[1] ist eine auf Texteingabe basierende Notensatz-Software. Text in Form von genau definierten Codes wird in eine hochwertige Notendarstellung umgewandelt. Der Code ist weitgehend logisch aufgebaut. So werden

- Notennamen mit Kleinbuchstaben von a – g,
- Oktavräume mit " , " (Komma) für untere Oktaven und " ’ "(Hochkomma) für obere Oktaven sowie
- Notenwerte von 1 (für Ganze Noten) bis 128 (für 128-tel)

als Text eingegeben.
Der Code für eine einfache C-Dur-Tonleiter in der eingestrichenen Oktave würde demnach so aussehen:

```
{c’ d’ e’ f’ g’ a’ b’ c’’}
```

Die resultierende Notendarstellung:



Musikalische Ausdrücke werden dabei in geschwungene Klammern gesetzt.

[1] http://lilypond.org (Februar 2015).

Die Voreinstellung sind 4-tel Noten, der Violinschlüssel, die Annahme von C-Dur und die englische Bezeichnung für die Tonhöhen (also b für unser h).

Wird kein Notenwert angegeben, bleibt bis zum nächsten Wechsel der letzte Eintrag gültig. Befehle und Kommandos werden mit einem *Backslash* "\" eingeführt. Der Befehl für den Violinschlüssel wäre beispielsweise:

```
\clef treble
```

und für den Bassschlüssel:

```
\clef bass
```

Tonarten werden mit `\key` angegeben. Beispielsweise

```
\key f \minor
```

für f-moll oder

```
\key f \major
```

für F-Dur.

Für die Einführung oder den Wechsel von Taktarten ist der Befehl `\time` zuständig:

```
\time 3/4
```

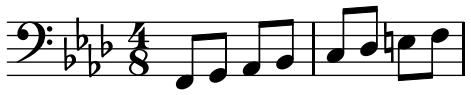
oder:

```
\time 9/8
```

Kreuz-Vorzeichen werden mit "is" an den Notennamen und Doppelkreuz mit "isis" notiert.
B-Vorzeichen mit "es" oder "eses".

Eine ausführlichere CODIERUNG könnte dann so aussehen:

```
{
\clef bass
\time 4/8
\key f \minor
f,8 g, aes, bes, c des e f
}
```



Mit LILYPOND verhält es sich sehr ähnlich wie mit LaTeX[1].

LILYPOND steht zu gängigen Notenschreibprogrammen wie FINALE oder SIBELIUS im selben Verhältnis wie LaTeX zu MICROSOFT WORD.

Die eine Art Software erweckt den Eindruck der Einfachheit, die andere erweckt den Eindruck des Komplizierten.

Bei näherer Beschäftigung stellt sich heraus, dass der erste Eindruck oft täuscht.

Im Gegensatz zu den gängigen Textverarbeitungsprogrammen, die bezeichnender Weise nach dem WYSIWYG-Prinzip (What-you-see-is-what-you-get) funktionieren, werden die Verfahrensweisen von LILYPOND oder LaTeX gerne mit WYSIWYM (What-you-see-is-what-you-mean) umschrieben.

LILYPOND-CODE kann in LaTeX-CODE nahtlos eingefügt werden, um Text mit Notendarstellungen zu verbinden. Des Weiteren erlaubt OPENOFFICE[2] (mit dem Zusatz OOoLILYPOND[3]) oder auch SCRIBUS[4] LILYPOND-CODE

[1] LaTex ist eine Abkürzung für **La**mport **TeX** und eine Weiterentwicklung des von DONALD E. KNUTH entwickelten Textsatzsystems TeX durch LESLIE LAMPORT.
[2] http://www.openoffice.org/de/ (Februar 2015).
[3] http://ooolilypond.sourceforge.net (Februar 2015).
[4] SCRIBUS ist ein Open Source DTP (Desktop-Publishing)-Programm zur Erstellung von professionellem Layout. http://www.scribus.net (Februar 2015).

einzufügen – und damit Notation in Text oder Grafik einzubinden.

Zum Eingeben des *Quelltextes* ist jeder beliebige Text-Editor geeignet. Viele Erleichterungen und Unterstützungen bei der Texteingabe inklusive einer pdf-Vorschau, bietet das Programm FRESCOBALDI[1].

Zu guter Letzt ist LILYPOND kostenfrei und steht sämtlichen Computer-Plattformen zur Verfügung.

5.2 Geschichte von LilyPond

> *„We knew what 'publication quality' engraving meant, and were determined to perfect Lily into producing that."*[2]
>
> Han-Wen Nienhuys

LILYPOND wird von HAN-WEN NIENHUYS[3] und JAN NIEUWENHUIZEN[4] seit 1997 entwickelt. Im Wesentlichen arbeiten sie in ihrer Freizeit ehrenamtlich an Lilypond.[5] Dank vieler Spender steht mittlerweile ein großes Entwicklerteam dahinter. Bei der derzeitigen LILYPOND-Version 2.18 haben laut Homepage über 70 Leute mitgearbeitet.

Seit 2010 ist es mit WEBLILY[6] möglich, Partituren direkt online zu erstellen.

Seit 2013 kann man in WIKIPEDIA Notenbeispiele auch direkt mit LILYPOND-CODE erstellen.

[1] http://frescobaldi.org (Februar 2015).
[2] http://www.all-day-breakfast.com/cannam/linux-musician/lilypond.html (Februar 2015).
[3] HAN-WEN NIENHUYS, geboren 1975 ist holländischer Programmierer. Er hat an der Universität Utrecht studiert und ist derzeit Programmierer bei GOOGLE. http://hanwen.home.xs4all.nl (Februar 2015).
[4] JAN NIEUWENHUIZEN, geboren 1968 ist holländischer Software-Entwickler. http://joyofsource.com/about.en.html (Februar 2015).
[5] Vergleiche: http://de.wikipedia.org/wiki/LilyPond (Februar 2015).
[6] http://www.weblily.net (Februar 2015).

Jan Nieuwenhuizen startete auch das Mutopia-Project[1] als Teil von LilyPond. Das Mutopia-Project veröffentlicht frei verfügbare Musikalien, welche alle im LilyPond-Format erstellt werden. So kann jeder mit Hilfe der LilyPond-Software die Stücke beliebig weiterverarbeiten. Derzeit sind um die 1900 Stücke (Stand November 2014) abrufbar.

5.3 Verwandte von LilyPond

Bei LilyPond spricht man von einer „Musik-Auszeichnungssprache“ oder von einem „Markup-Notensatzprogramm“.
Ähnliche Ansätze werden auch bei folgenden Software-Paketen verfolgt:

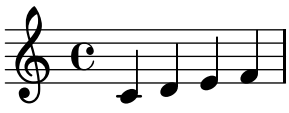
- MusiXTeX[2]
  Die Integration von Notation in LaTeX ist mit MusiXTeX ganz nahtlos möglich.
- ABC notation[3]
- Common Music Notation (CMN)[4]
  ist ein Lisp-Programm.
- Guido Music Notation[5]
- Score[6]

5.4 LilyPond Snippets

Anschließend einige LilyPond Code-Schipsel mit ihrer resultierenden Notendarstellung.

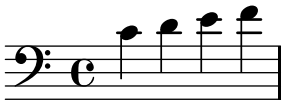
[1] http://www.mutopiaproject.org (Februar 2015).
[2] http://icking-music-archive.org (Februar 2015).
[3] http://abcnotation.com (Februar(2015).
[4] https://ccrma.stanford.edu/software/cmn/cmn/cmn.html (Februar 2015).
[5] http://science.jkilian.de/salieri/GUIDO/ (Februar 2015).
[6] http://scoremus.com (Februar 2015).

Voreinstellung



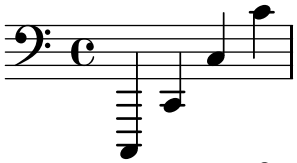
```
{ c’ d’ e’ f’ }
```

Notenschlüssel

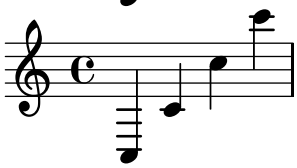


```
{ \clef bass c’ d’ e’ f’ }
```

Oktavräume

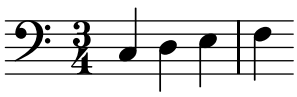


```
{ \clef bass c,, c, c c’ }
```



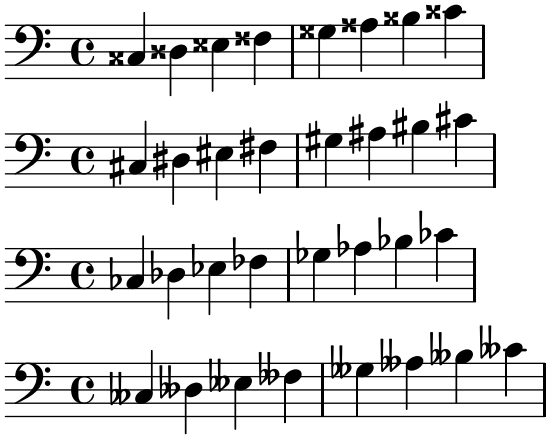
```
{ \clef treble c c’ c’’ c’’’ }
```

Taktart



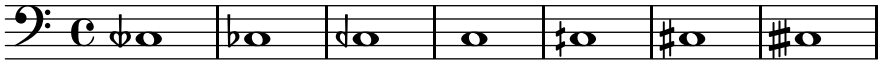
```
{
\clef bass
\time 3/4
c d e f
}
```

Vorzeichen



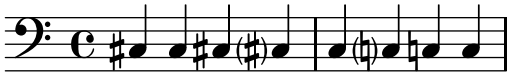
```
{
\clef bass
cisis disis eisis fisis
     gisis aisis bisis cisis'
cis dis eis fis gis ais bis cis'
ces des ees fes ges aes bes ces'
ceses deses eeses feses
     geses aeses beses ceses'
}
```

Vorzeichen Mikrotöne



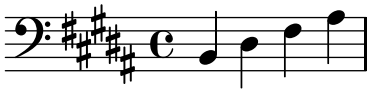
```
{
\clef bass
ceseh1 ces ceh c cih cis cisih
}
```

Vorzeichen Erinnerungen

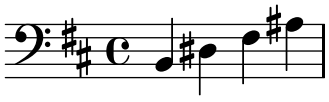


```
{
\clef bass
cis cis cis! cis? c c? c! c
}
```

Vorzeichnung

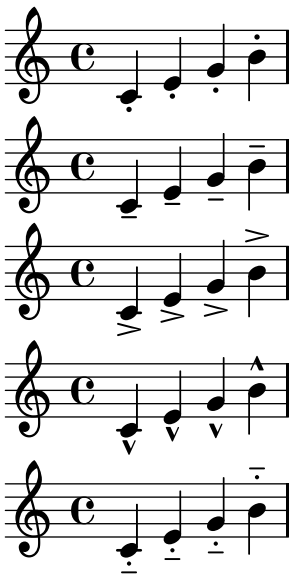


```
{
\clef bass
\key b \major
b, dis fis ais
}
```



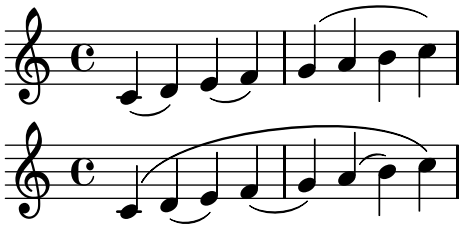
```
{
\clef bass
\key b \minor
b, dis fis ais
}
```

Artikulationen



```
{ c'-.  e'-.  g'-.  b'-. }
{ c'--  e'--  g'--  b'-- }
{ c'->  e'->  g'->  b'-> }
{ c'-^  e'-^  g'-^  b'-^ }
{ c'-_  e'-_ g'-_  b'-_  }
```

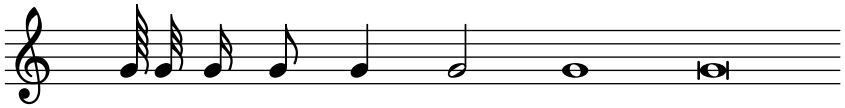
Legato & Phrasierung



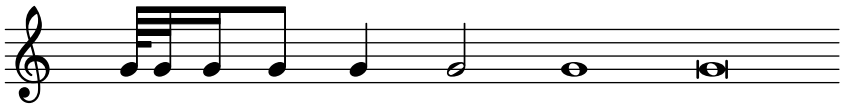
```
{ c’( d’) e’( f’) g’( a’ b’ c’’) }
```

```
{ c’\( d’( e’) f’( g’) a’( b’) c’’\) }
```

Notenwerte

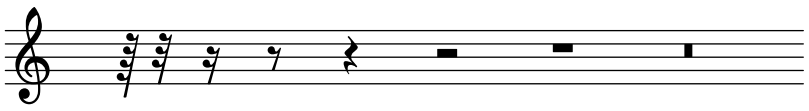


```
{
\override BarLine #’transparent = ##t
\override Staff.TimeSignature #’transparent = ##t
\cadenzaOn
g’64 g’ 32 g’16 g’8 g’4 g’2 g’1 g’\breve
}
```



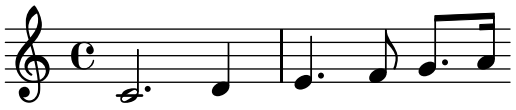
```
{
\autoBeamOn
\override BarLine #’transparent = ##t
\override Staff.TimeSignature #’transparent = ##t
\cadenzaOn
g’64 g’ 32 g’16 g’8 g’4 g’2 g’1 g’\breve
}
```

Pausenwerte



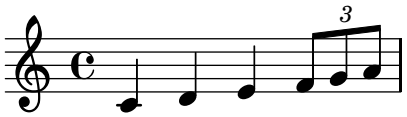
```
{
\override BarLine #'transparent = ##t
\override Staff.TimeSignature #'transparent = ##t
\cadenzaOn
r64 r32 r16 r8 r4 r2 r1 r\breve
}
```

Punktierte



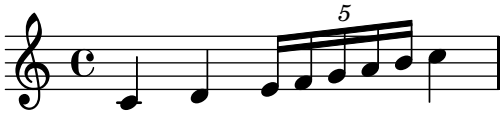
```
{ c'2. d'4 e'4. f'8 g'8. a'16 }
```

Triolen



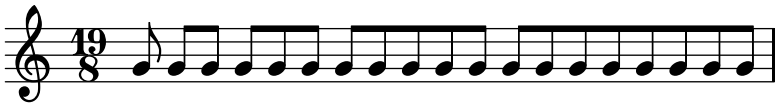

```
{
c' d' e'
\times 2/3 {f'8 g' a'}
}
```

N-Tolen

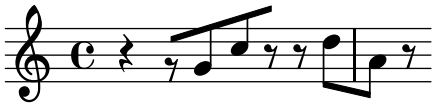



```
{
c'4 d'4
\times 4/5
{e'16 f'16 g'16 a'16 b'16} c''4
}
```

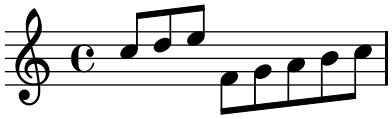
Balken



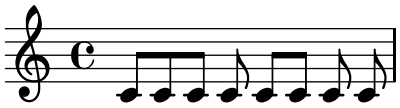
```
{
\time 19/8
\set Staff.beatStructure = #'(1 2 3 5 8)
\repeat unfold 19 {g'8}
}
```



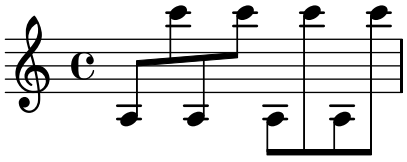
```
{
r4 r8[ g' c'' r] r d''[ | a'] r
}
```



```
{
c''8^[ d'' e''] f'_[ g' a' b' c'']
}
```

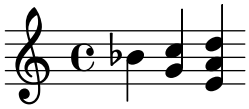


```
{
c'8 c'8 c'8 c'8\noBeam
c'8 c'8 c'8\noBeam c'8
}
```



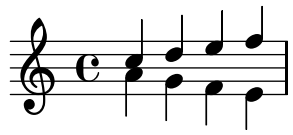
```
{
a8 c'''8 a8 c'''8
\override Beam #'auto-knee-gap = #8
a8 c'''8 a8 c'''8
}
```

Akkorde

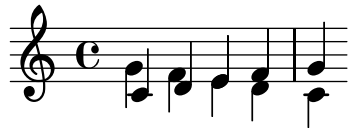


```
{
bes'   <g' c''>   <e' a' d''>
}
```

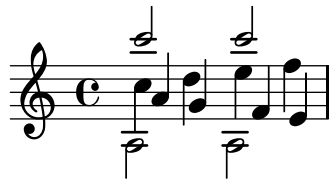
Mehrstimmigkeit



```
<<{ c'' d'' e'' f'' } \\ { a' g' f' e' }>>
```

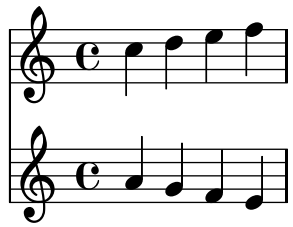


```
<<{ c' d' e' f' g' } \\ { g' f' e' d' c' }>>
```



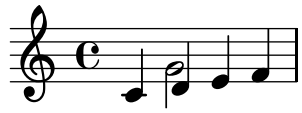
```
<< { c'''2  c''' } \\  { c''4 d'' e'' f'' } \\
   { a' g' f' e' } \\  { a2  a }>>
```

Notensysteme



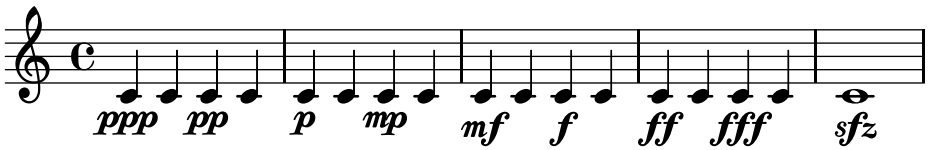
```
<<{ c'' d'' e'' f'' }  { a' g' f' e' }>>
```

Versteckte Noten

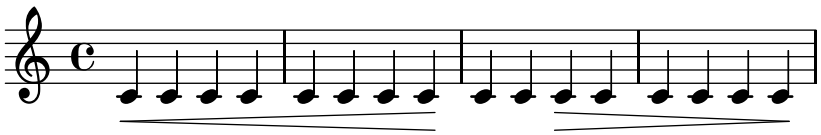


```
<<{ c' d' e' f' } \\ { s  g'2 }>>
```

Dynamische Zeichen

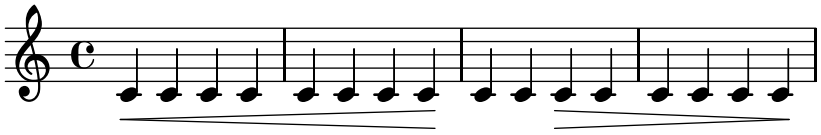



```
{
c'4\ppp c' c'4\pp c' c'4\p c'4 c'4\mp c' c'4\mf c'
c'4\f c' c'4\ff c' c'4\fff  c' c'1\sfz
}
```



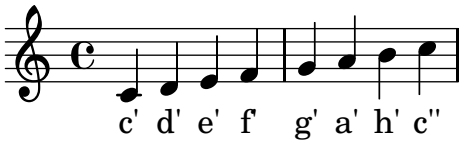
```
{
c'4\< c'4 c'4 c'4   c'4 c'4 c'4 c'4\!
c'4 c'4 c'4\> c'4   c'4 c'4 c'4 c'4\!
}
```

oder:



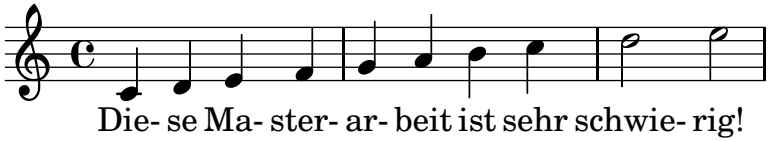
```
\relative c'
{
c4\< c c c   c c c c\!
c c c\> c   c c c c\!
}
```

Text




```
<<
{
{c’ d’ e’ f’ g’ a’ b’ c’’}
}
\addlyrics
{
c’ d’ e’ f’ g’ a’ h’ c’’
}
>>
```

oder:

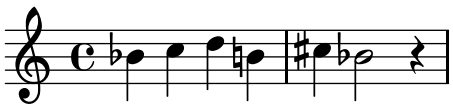



```
<<
{
{c’ d’ e’ f’ g’ a’ b’ c’’ d’’2 e’’}
}
\addlyrics
{
Die- se Ma- ster- ar- beit ist sehr schwie- rig!
}
>>
```

Manipulation von Motiven:

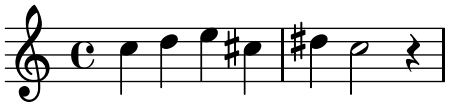
In LilyPond selbst sind Ansätze der Möglichkeit *algorithmischer* Manipulation von Notations-Code vertreten.

Motiv



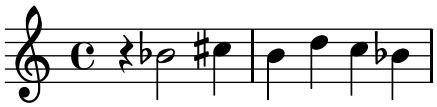
```
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Transposition



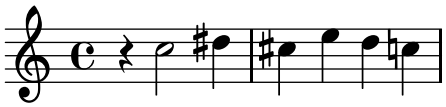
```
\transpose c' d'
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Krebs



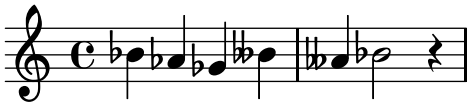
```
\retrograde
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Krebs + Transposition



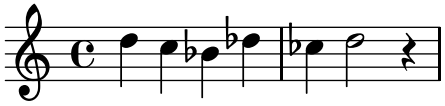
```
\transpose c' d'
\retrograde
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Umkehrung



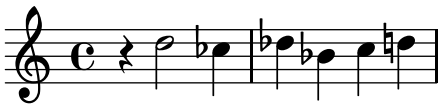
```
\inversion bes' bes'
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Umkehrung 2



```
\inversion c'' c''
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

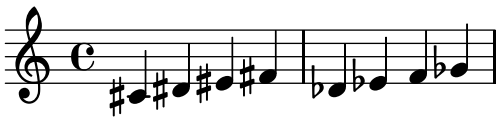
Krebsumkehrung



```
\inversion c'' c''
\retrograde
{ bes' c'' d'' b' cis'' bes'2 r4 }
```

Motive

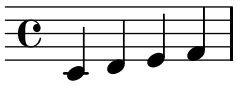
Musikinformation als *Variable* definieren.



```
music = \relative c' { c d e f }
\new Staff
{
   \transpose c cis { \music }
   \transpose c des { \music }
}
```

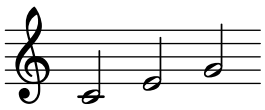
Verstecken von Notationssymbolen

Notenschlüssel verstecken



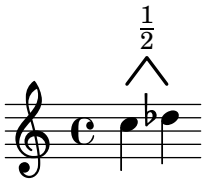
```
{
\override Staff.Clef #'stencil = ##f
c' d' e' f'
}
```

Verstecken der Taktart



```
\layout {
    \context {
        \Staff
  \remove Time_signature_engraver
  \remove Bar_engraver
    }
}
{ c'2 e' g' }
```

Grafische Einträge



```
samplePath =
  #'( (moveto 0.5 0)
      (rlineto 1.5 2)
      (rlineto 1.5 -2) )
{
c''
^\markup
{
   \path #0.25 #samplePath
}
^\markup
{\hspace #1.5 \fontsize #-1.5 \fraction 1 2}
des''
}
```

6 LilyPond meets MAX

Es liegt die Schlussfolgerung nahe, die Vorzüge von LilyPond – mittels Code zu notieren – mit den Stärken von MAX – Code zu generieren – zu kombinieren.

Ziel in diesem Kapitel ist, einen möglichen Weg aufzuzeigen, wie wir von den *Listen* und *Messages* in MAX zu dem für LilyPond verständlichen Code gelangen können.

Ein weiteres Ziel ist, diesen Code als *Text-File* abspeichern zu können, damit wir in der Lage sind, diesen in LilyPond zu öffnen.

Anschließend wird dieser *Text-File* mit dem LilyPond-Code in eine Notendarstellung umgewandelt – im Idealfall als vollständige Partitur mit perfektem *Layout*.

Automatisiert – auf „Knopfdruck".

Fürs Erste ein paar einfache Vorschläge, wie wir LilyPond-Code mit MAX-*Messages* und der Aneinanderreihung und/oder Verschachtelung von MAX-*Listen* erzeugen könnten.

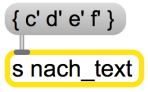
Wir bilden auch einige *Abstractions* – nicht um den kreativen MAX-*User* zu bevormunden[1] – sondern um am Bildschirm sowie am Papier Platz zu sparen.

[1] Jeder, der sich mit MAX auseinandersetzt, hat seine eigenen Vorlieben, die Objects zu verbinden und *Algorithmen*, beziehungsweise *Patches* zu bilden – es gibt dabei immer viele unterschiedliche Möglichkeiten, die zum Ziel führen.

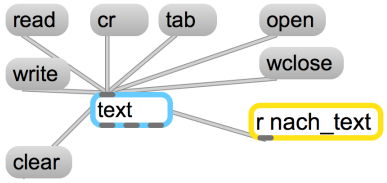
6.1 Max2LilyPond

Um LilyPond-*Files*, welche nach ihrer Erzeugung im LilyPond-Programm geöffnet werden können, von MAX aus in einen Ordner auf der Festplatte des Computers schreiben zu können, benötigen wir das *text*-Object.

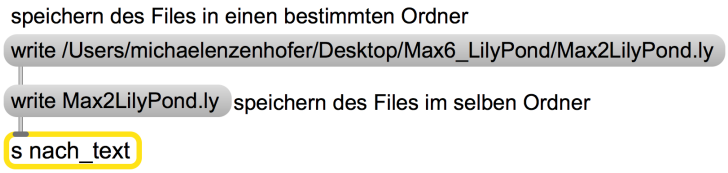
Eine MAX-*Message*, die direkt oder über ein *send*-Object zum *text*-Object gesendet wird, wird im *text*-Object gespeichert. Diese *Message* wird nach einer eventuell vorausgegangenen *Message* platziert und mit einem *Leerzeichen* abgeschlossen.



Das *text*-Object versteht sich mit "cr"- und "tab"-*Messages* für ***c**arriage **r**eturn* (Zeilenumschalter) und *Tabulator*.
Text im *text*-Object kann durch einen *Doppelklick* oder durch die "open"-*Message* geöffnet, betrachtet und anschließend auch geändert werden.
Durch eine "write"-*Message* öffnet sich eine Standard-Dialog-Box, um den Inhalt des *text*-Objects auf einen beliebigen Ort auf der Computer-Festplatte zu speichern.



Wird die "write"-*Message* mit einem Namen verbunden, wird der Inhalt mit dem *File*-Namen in den Ordner des MAX-*Patches* geschrieben. Ist die "write"-*Message* mit einer absoluten *Pfad*-Angabe verbunden, wird der Inhalt des *text*-Objects an dem entsprechenden Ort gespeichert.



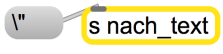
6.2 MidiNoteNumbers2LilyPondNoteNames

Ein wesentlicher Schritt ist die Umwandlung der Midi-Noten-Nummern zu den Notennamen, die LilyPond versteht. Also müssen Zahlen zu Buchstaben mit Beistrichen oder Hochkommas gewandelt werden.

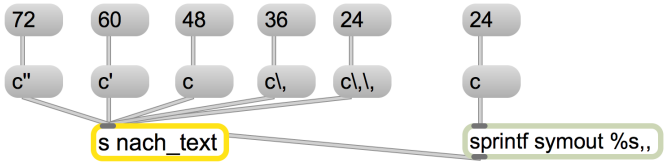
Dabei muss beachtet werden, dass ein *text*-Object manche Zeichen nicht einfach erkennt. Diese müssen daher mit einem vorangestelltem *Backslash* ("\") platziert werden.

Der Beistrich und der *Backslash* selbst – zwei wesentliche Zeichen in LilyPond – sind von dieser Notwendigkeit betroffen.
Wird also ein Beistrich gebraucht, muss in der *Message*, die einen Beistrich beinhalten soll, vor dem Beistrich ein *Backslash* platziert werden.
Wird ein *Backslash* gebraucht, muss ein *Backslash* vor dem *Backslash* – müssen also zwei *Backslashes* – eingefügt werden.

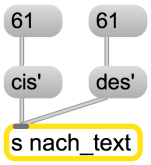
Wird Text über oder unter einer Note oder in der Folge als Schriftzug in der Partitur benötigt, muss er in Anführungszeichen gesetzt werden. Bei Anführungszeichen gilt die gleiche Vorgangsweise – ihnen muss in der *Message-Box* ein *Backslash* vorangestellt werden.



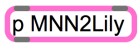
Die Umwandlung von Zahlen nach LilyPond-Tonhöhenbezeichnungen geschieht etwa folgendermaßen:



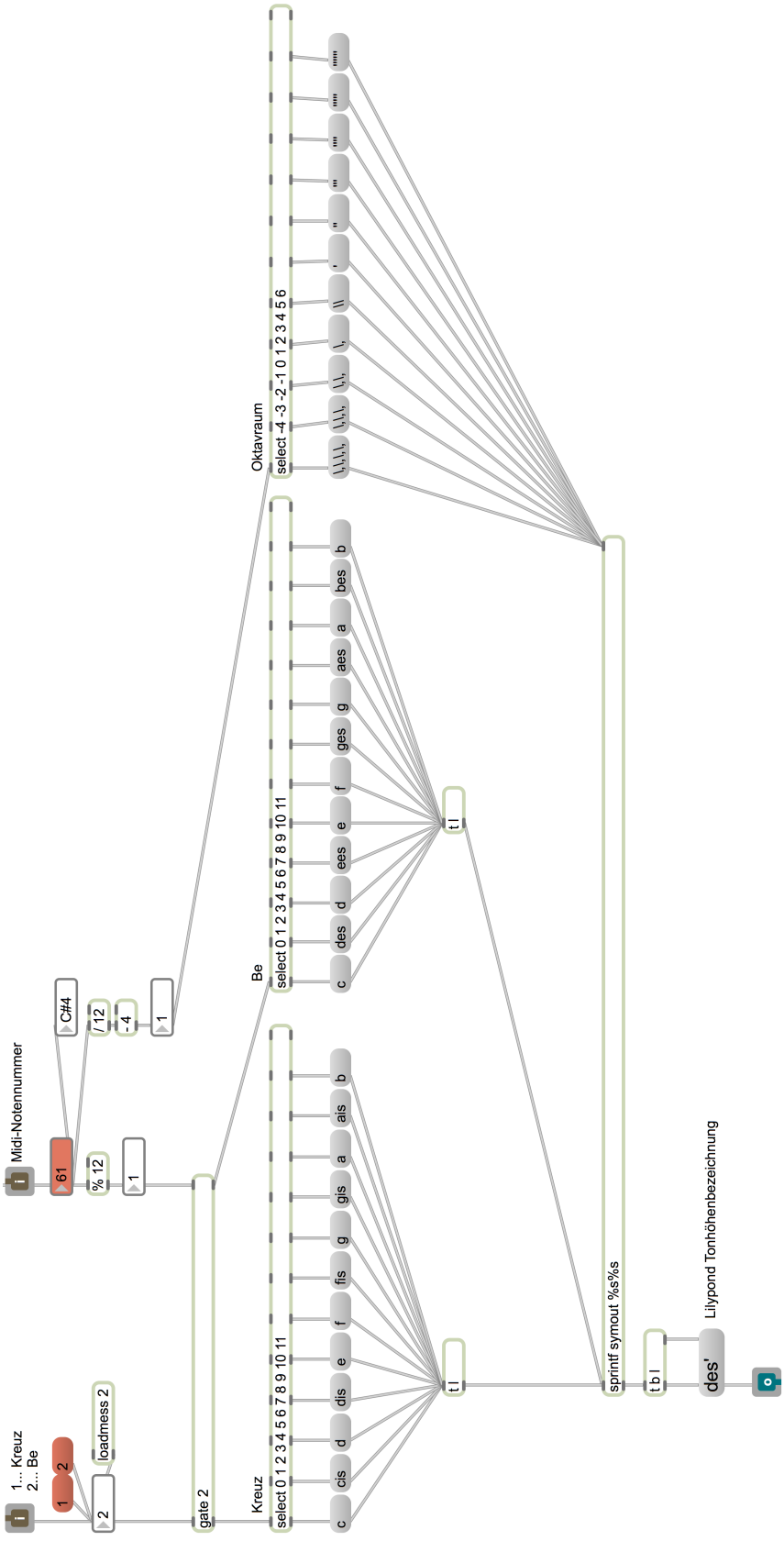
Zu beachten ist noch, dass Midi-Noten-Nummern keine Vorzeichen beinhalten. Die Midi-Noten-Nummer 61 kann also ein *eingestrichenes cis* oder ein *eingestrichenes des* bedeuten. Hier muss der programmierende Komponist bewusst vorgehen – je nachdem, was er braucht.



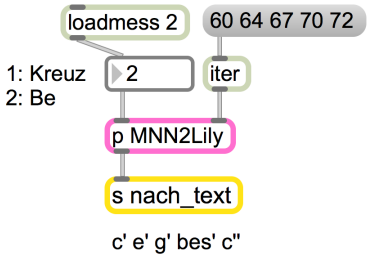
Für diese Thematik wäre ein Object, welches sämtliche Midi-Noten-Nummern von 0–127 in die Tonhöhenbezeichnung von LilyPond umwandelt, ideal. Deswegen mein Vorschlag für ein *Abstraction*-Object:

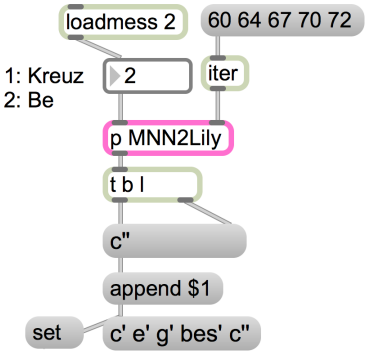


Das Innere des Objects sieht folgendermaßen aus und erfüllt die Anforderungen:

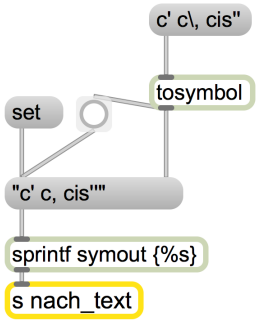

1... Kreuz
2... Be
loadmess 2
gate 2
Midi-Notennummer
61
C#4
% 12
/ 12
- 4
- 1
- 1
Kreuz
select 0 1 2 3 4 5 6 7 8 9 10 11
c cis d dis e f fis g gis a ais b
Be
select 0 1 2 3 4 5 6 7 8 9 10 11
c des d ees e f ges g aes a bes b
Oktavraum
select -4 -3 -2 -1 0 1 2 3 4 5 6
t l
t l
sprintf symout %s%s
t b l
Lilypond Tonhöhenbezeichnung
des'

Midi-Noten-Nummern-*Listen* werden auf folgende Art in Tonhöhenbezeichnung für das *text*-OBJECT umgewandelt:
Links werden die resultierenden Tonhöhen gleich in das *text*-OBJECT befördert. Auf der rechten Seite wird dargestellt, wie eine Midi-Noten-Nummern-*Liste* zu einer Tonhöhen-*Liste* wird.





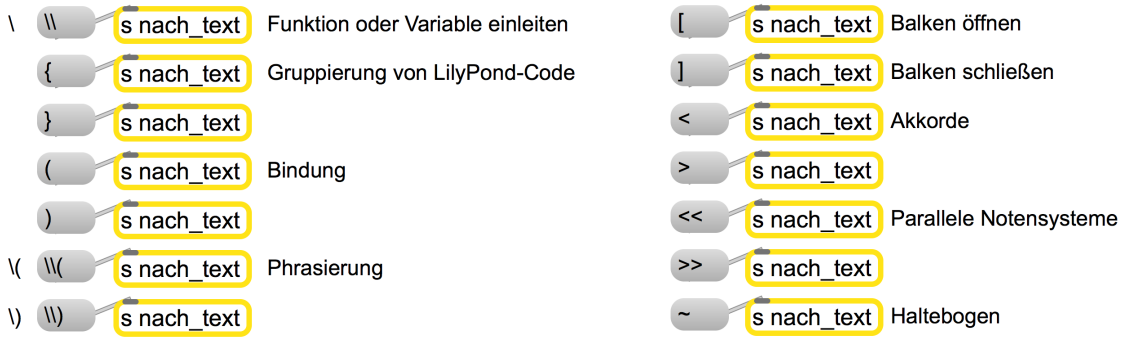
Das *sprintf*-OBJECT zeigt sich in vielen Fällen sehr nützlich. Meist muss man dem OBJECT das *Argument* "symout" beifügen, um zu erhalten, was man benötigt. Hier ist *sprintf* dazu nützlich, die Tonhöhen-*Liste* mit geschwungenen Klammern zu umrahmen.



6.3 MaxMessages4LilyPondSyntax

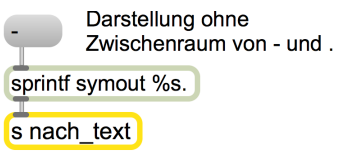
Nachfolgend sind wesentliche MAX-*Messages* aufgelistet, wie sie zum Erstellen von LILYPOND-CODE verwendet werden könnten.

Zeichen zum Notensatz

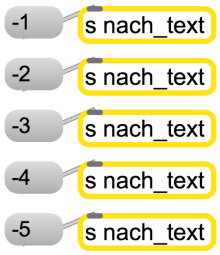


Befehle zur Artikulation

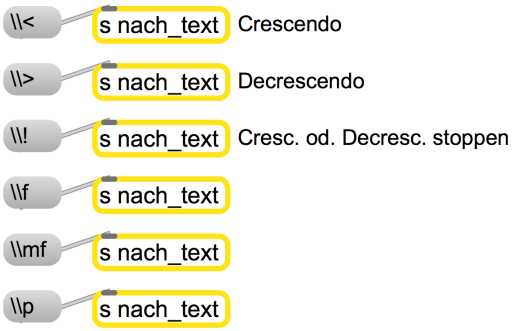
Zum Notieren von Artikulationszeichen sind obige Kurzzeichen sowie die wörtliche Ausschreibung der Befehle möglich.
Beim Zeichen für *Staccato* mit dem Bindestrich und anschließendem Punkt ("-.") muss in der MAX-*Message* nach dem Bindestrich ein Leerzeichen eingefügt werden. Mit dem *sprintf*-OBJECT könnte eine andere Lösung eingesetzt werden.



Kommandos für den Fingersatz

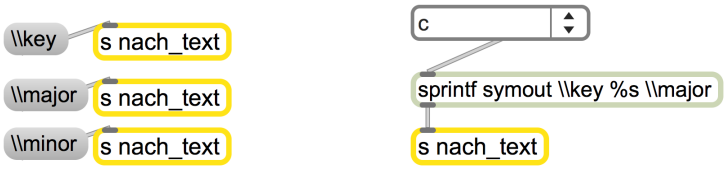


Befehle für dynamische Zeichen



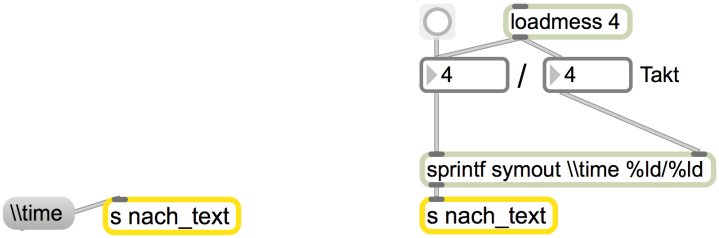
Tonart mit Vorzeichnung

LilyPond-Befehle für Tonarten mit Vorzeichnung können auch praktisch mit dem *sprintf*-Object aufbereitet werden.



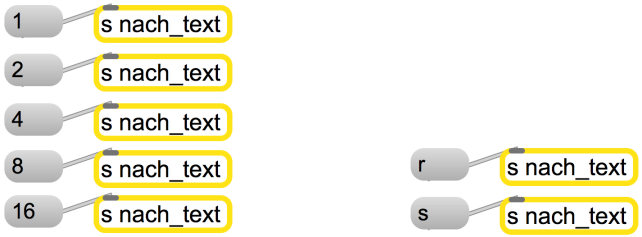
Taktart

Hier werden LilyPond-Befehle für Taktarten mit dem *sprintf*-Object aufbereitet.



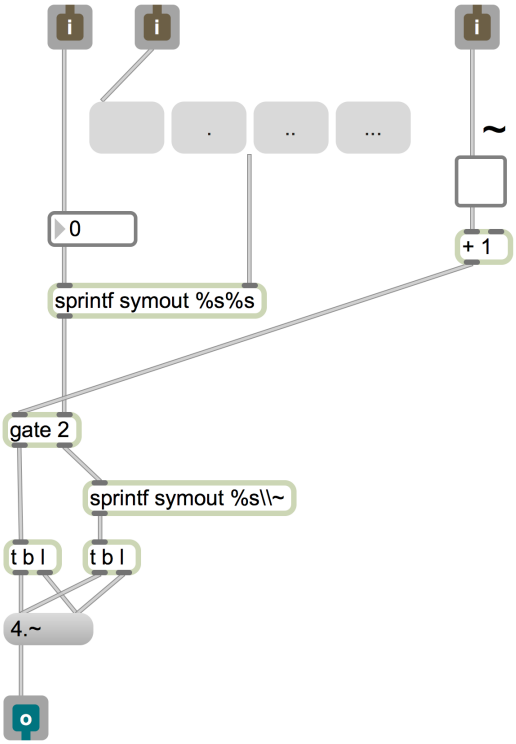
Noten- und Pausenwerte

Werte für Noten, Pausen und versteckte Noten werden mit entsprechenden ganzen Zahlen festgelegt.
Wenn zwischen Notennamen und Zahl sonst nichts angegeben wird, handelt es sich um einen Ton, wenn vor der Zahl ein "r" steht – (für *rest*) – handelt es sich um eine Pause. Wenn zwischen Notennamen und Zahl ein "s" steht – (für *space*) – wird ein entsprechender Abstand eingefügt.

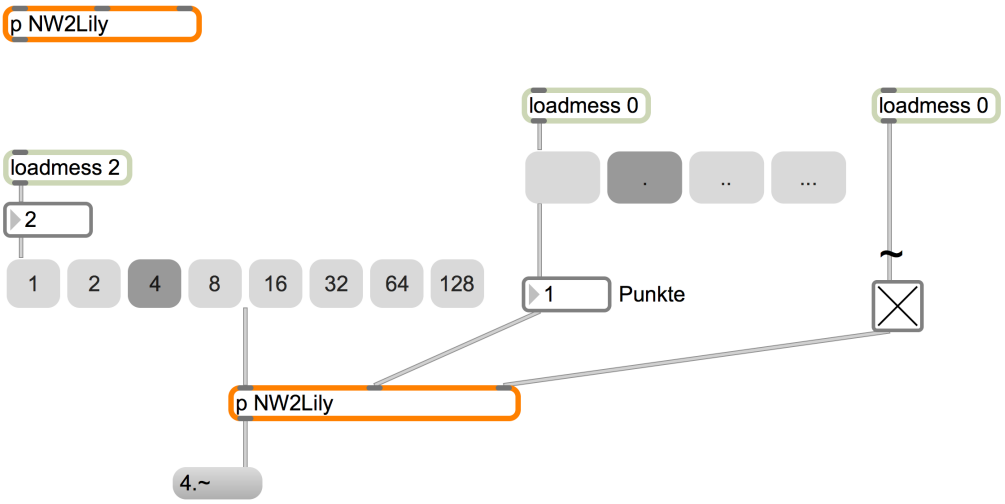


6.4 Duration2LilyPondNW

Da Notenwerte in LilyPond nicht nur mit den Zahlen 1 für Ganze, 2 für Halbe, 4 für Viertel, 8 für Achtel und so weiter definiert werden, sondern auch *punktiert* oder mit *Haltebogen* verbunden werden können, wäre ein eigenes Object dafür eine Erleichterung.

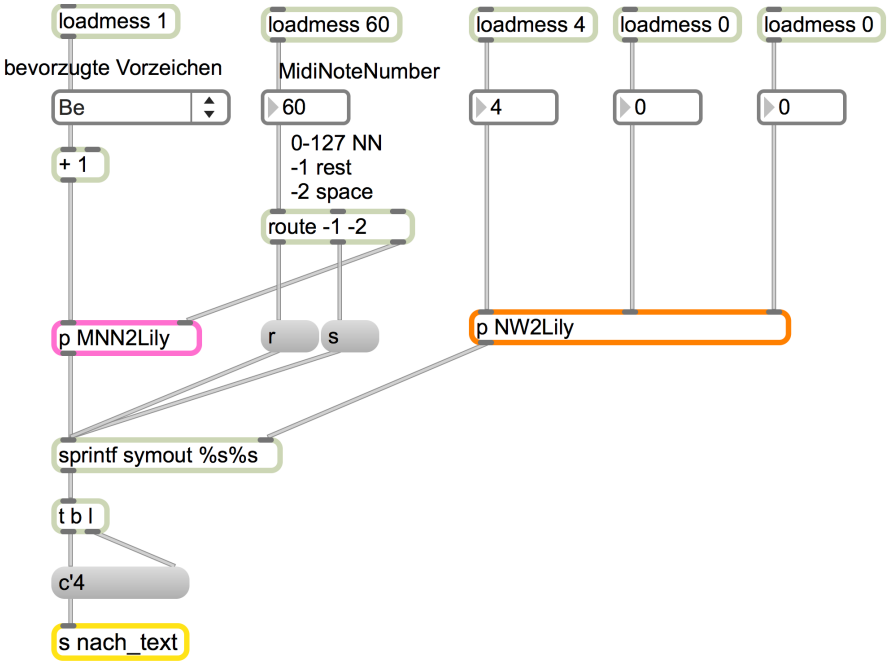


Diesen *Algorithmus* packen wir am besten in ein eigenes OBJECT, welches wieder als *Abstraction* verwendet werden kann.



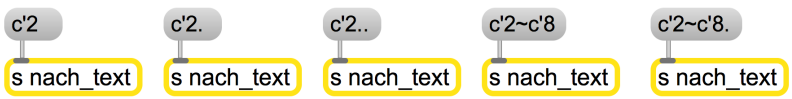
6.5 CombineNoteNamesWithRyhthmValues

Die Verbindung von Tonhöhe und Notenwert gestaltet sich beispielsweise folgendermaßen:

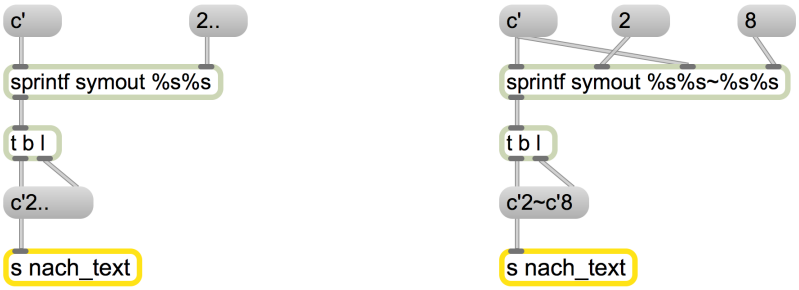


Da anstatt der Tonhöhe auch "r" für Pause oder "s" für Space vor den Notenwert platziert werden kann, wurde von mir die Umwandlungsmöglichkeit von -1 nach "r" und -2 nach "s" vorgenommen.

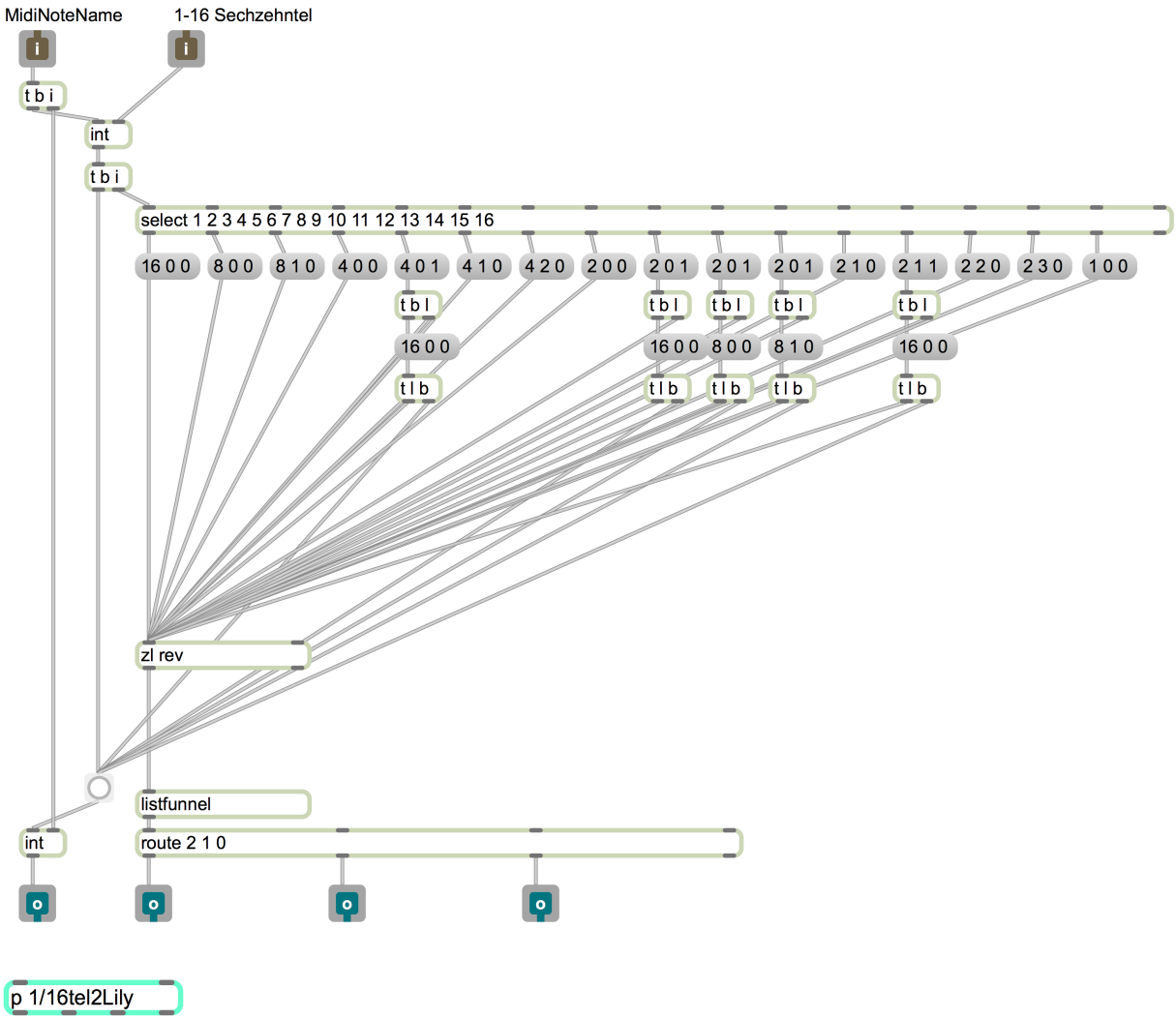
Wenn beispielsweise eine Anzahl von 16-tel-Notenwerten in die richtige Anordnung von Notenwert, eventuellem Punkt und/oder Haltebogen gebracht werden soll, muss darauf geachtet werden, nach einem Haltebogen die Tonhöhenbezeichnung wiederholt zu platziert.



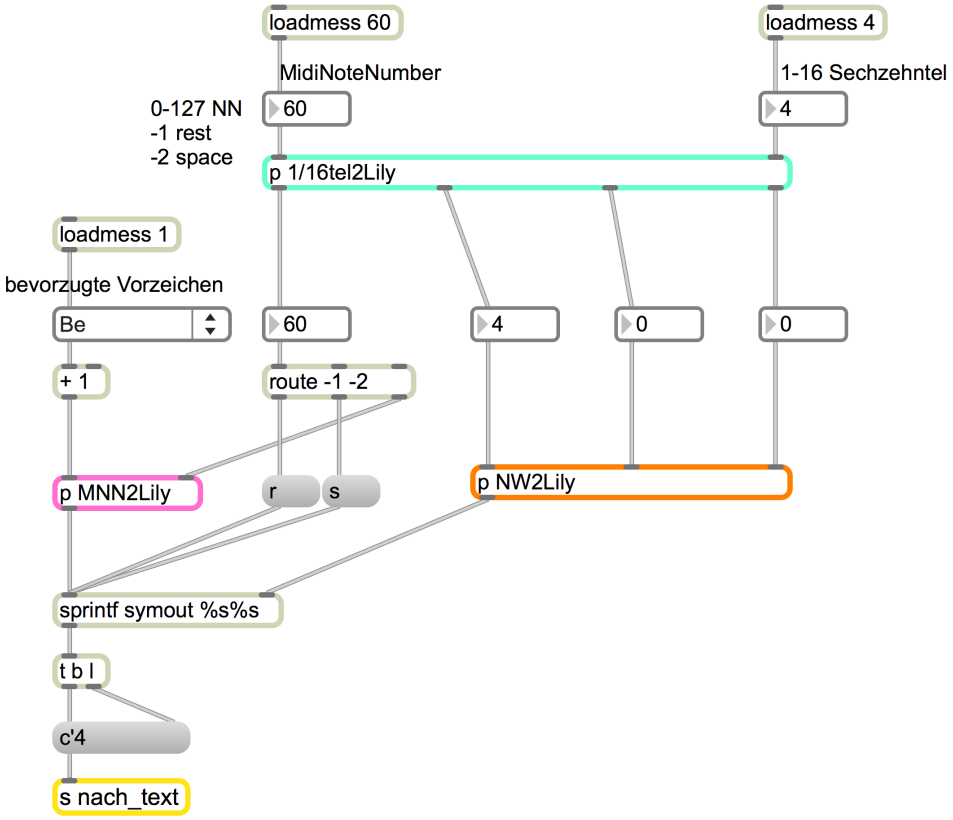
Mit *sprintf* könnte das Problem so gelöst werden:



Ein Object, welches die Anzahl von 1–16 Sechzehntel in die richtige Anordnung von Notenwert, Punkt und Haltebogen bringt, könnte folgende Verknüpfungen beinhalten:

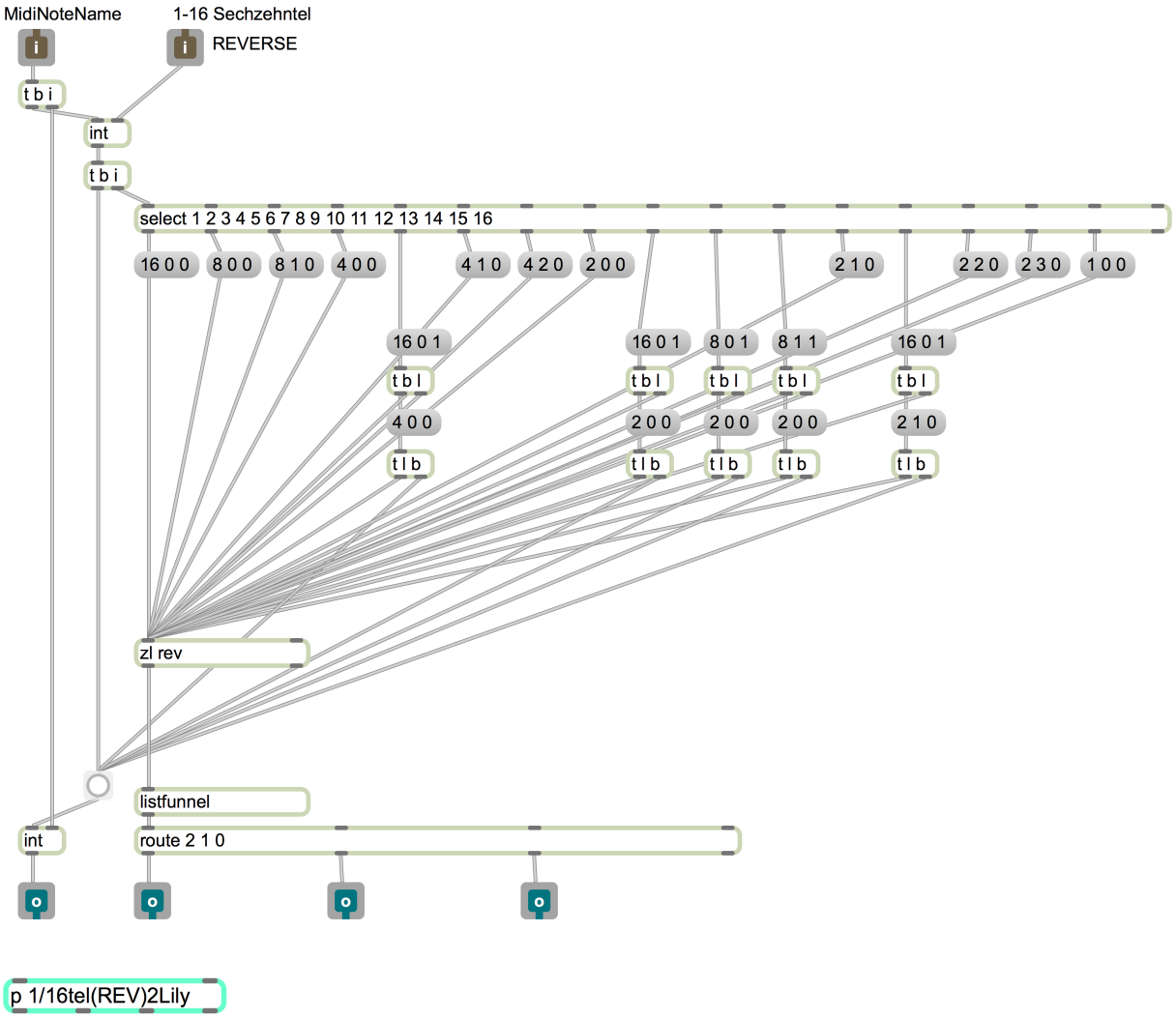


Eine mögliche Verbindung mit den bisherigen *Abstractions* wäre:

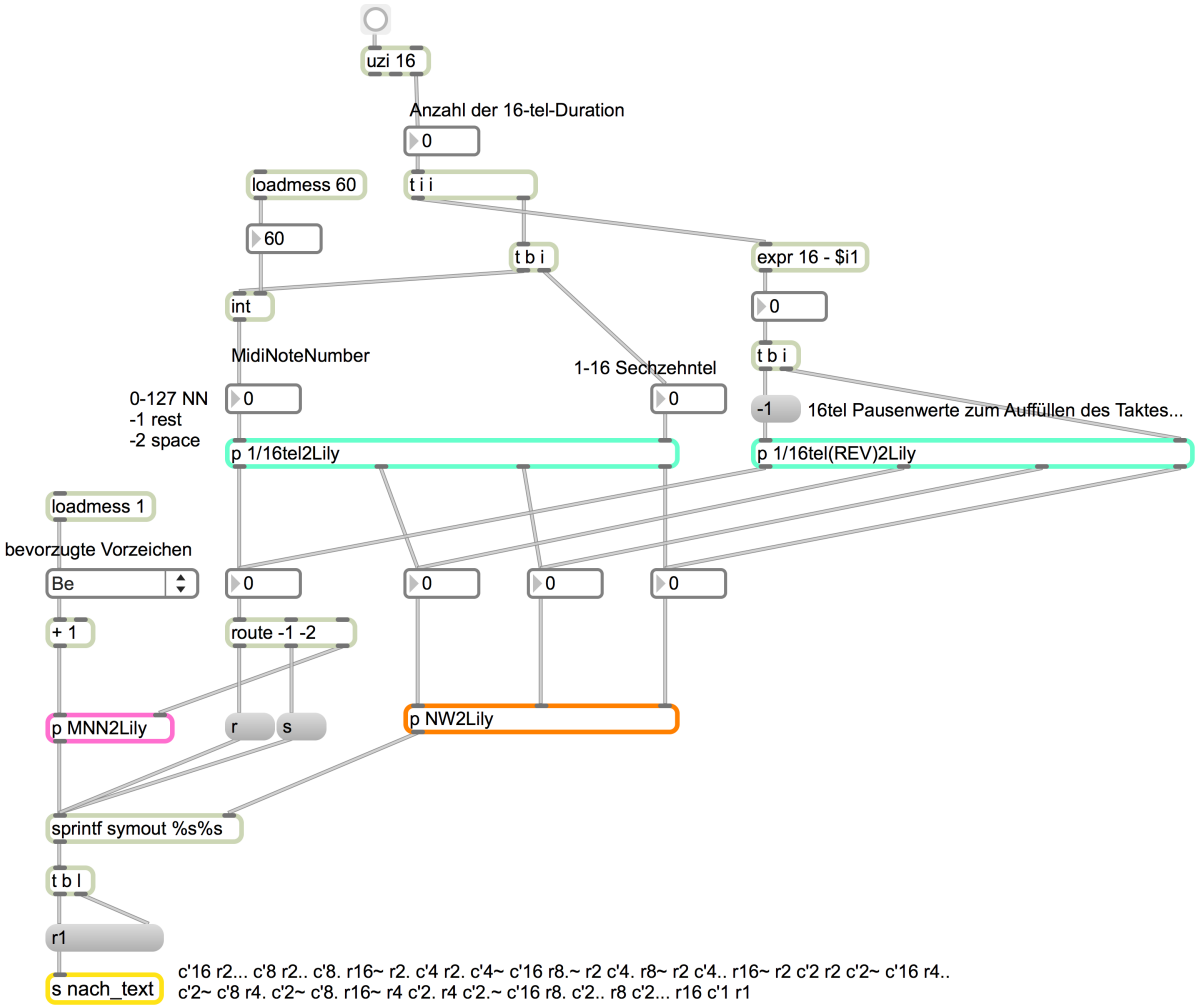


Bei manchen Versuchen stellt sich heraus, dass das generierte Notenbild nicht der Erwartung entspricht. Oft entstehen die *Algorithmen* in den *Abstractions* durch *Versuch und Irrtum*, bis sich die Erwartungen erfüllen.

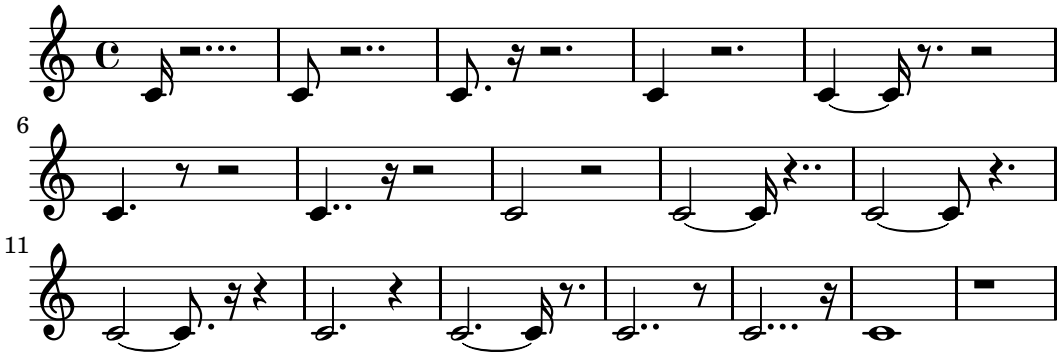
So ist es auch mit dem nächsten OBJECT, das auf ähnliche Weise die 16-tel aneinanderreiht, aber einen notwendigen Unterschied in der Anordnung macht, je nachdem, ob die Note vor der Pause oder die Pause vor der Note steht.



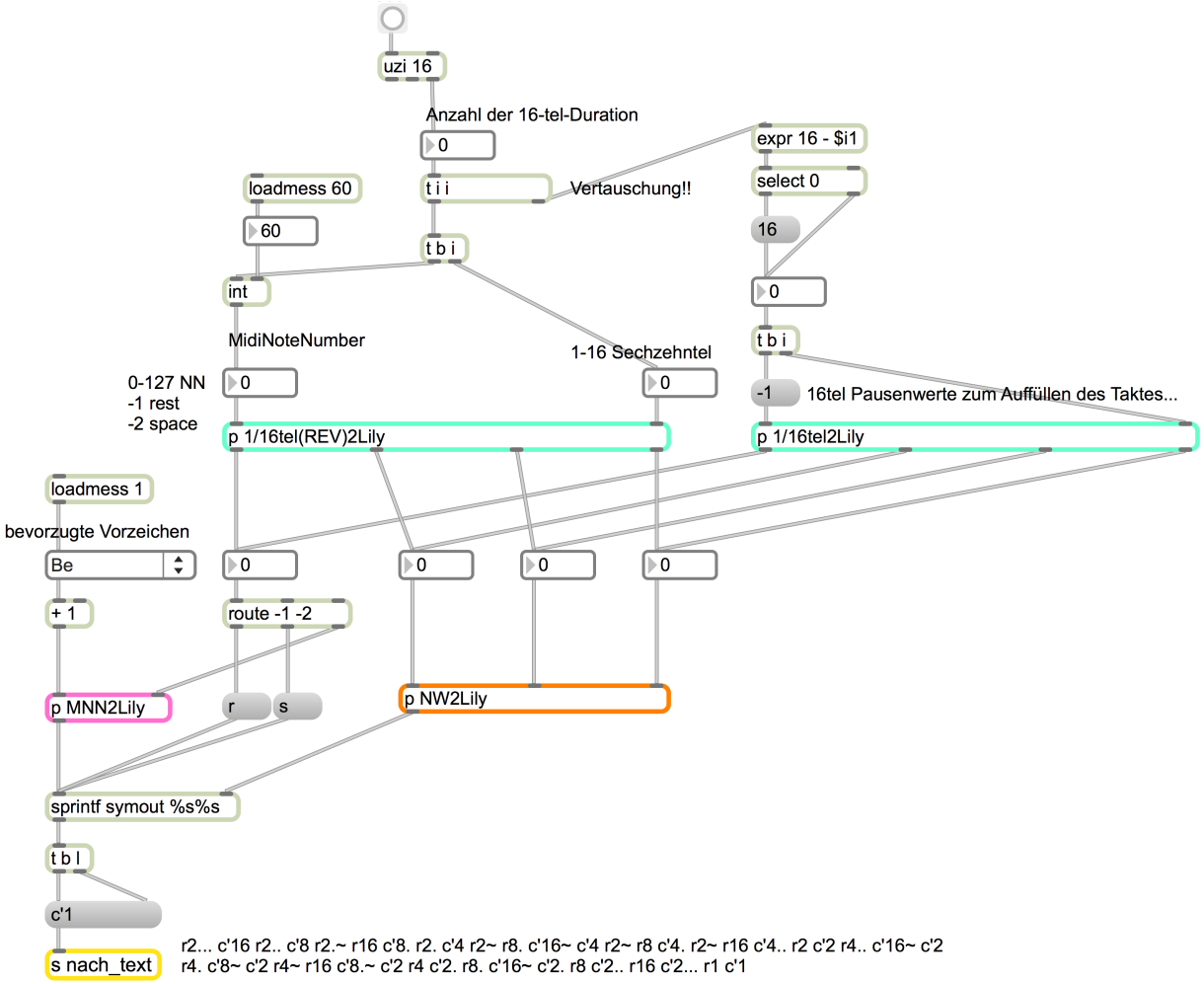
So liefert der CODE der ersten Anordnung...



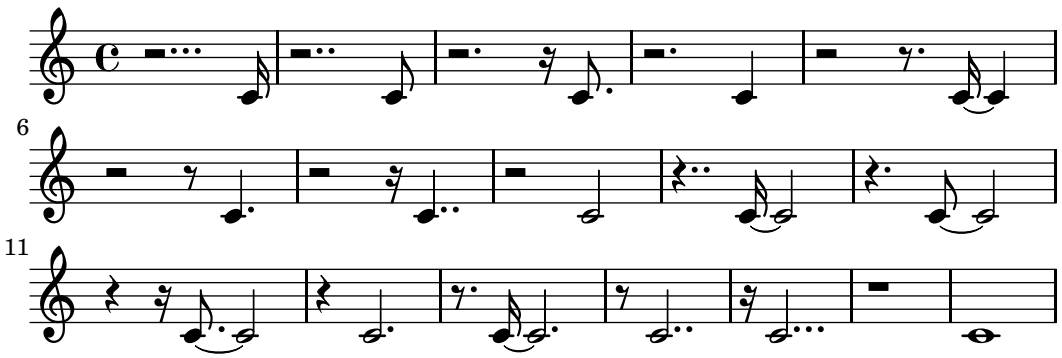
...dieses Notenbild:



Und der Code der zweiten Anordnung...



...dieses Notenbild:

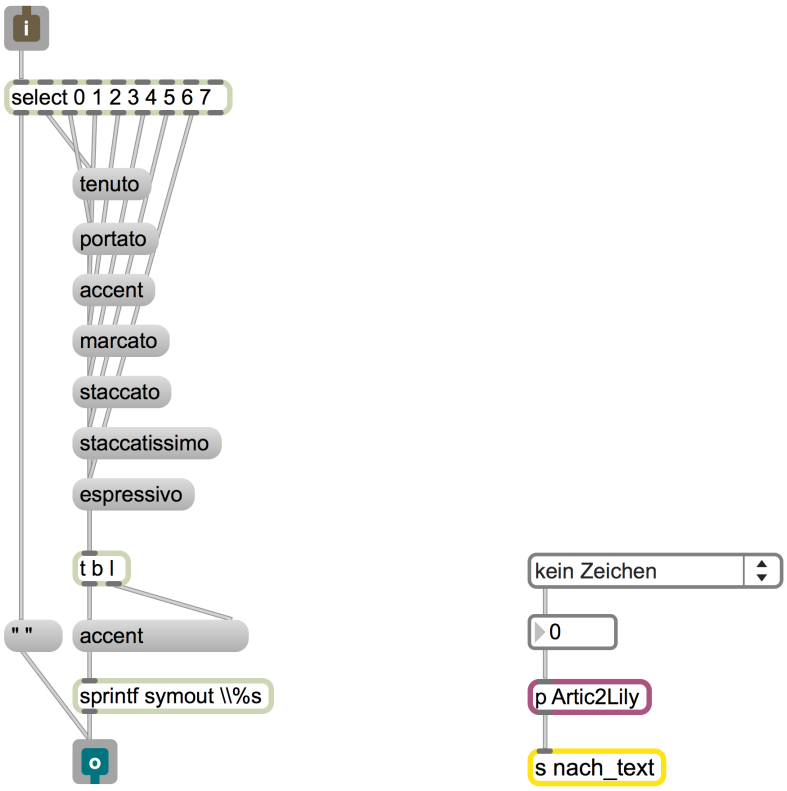


Meine Vorschläge zu den *Abstractions* sollen hier in erster Linie als Anregung oder mögliche *beta*-Entwicklungen verstanden werden. Den größten Nutzen kann man aus meinen Abhandlungen ziehen, wenn man

- gute Ideen zum Notieren oder Komponieren hat.
- mit MAX vertraut ist.
- mit LILYPOND vertraut ist.
- sich die Mühe macht, *Algorithmen* für seine Ideen zu entwickeln.

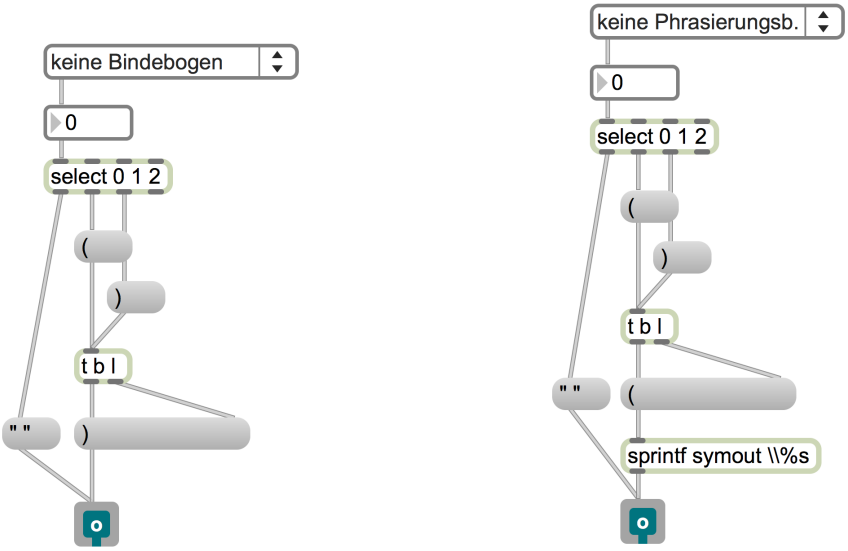
6.6 Articulations2LilyPond

Des Weiteren sind *Abstractions* für Angaben zur Artikulation und Dynamik sinnvoll.

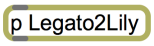


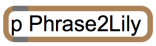
Obiger *Algorithmus* befindet sich im Artic2Lily-OBJECT.

Bindungen und Phrasierungen



Obige *Algorithmen* mit den resultierenden OBJECTS:

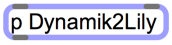




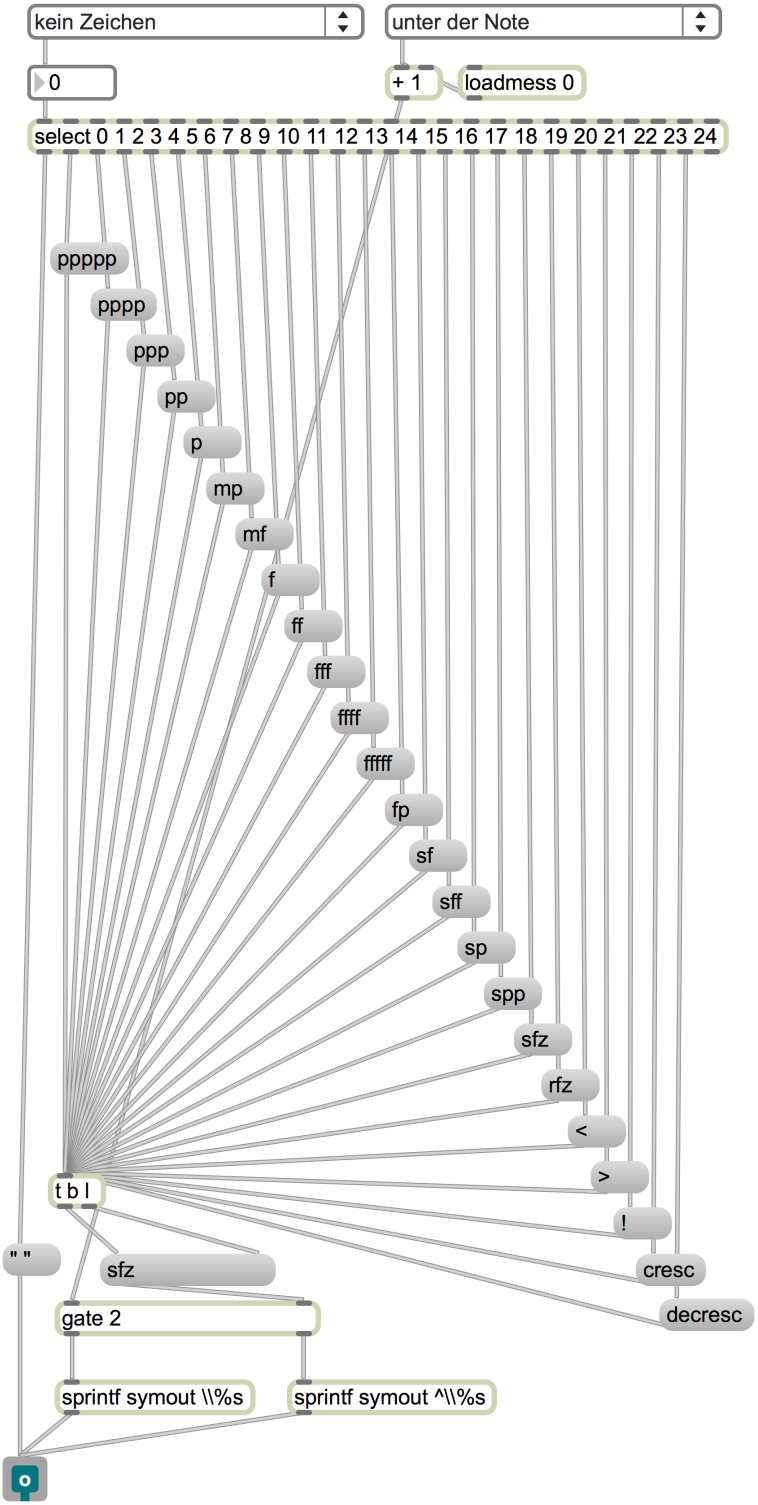
6.7 Dynamics2LilyPond

Dynamische Zeichen vom fünffachen *piano* bis zum fünffachen *forte* mit den *crescendo*- und *decrescendo*-Pfeilen und Bezeichnungen sind ähnlich zu CODIEREN.

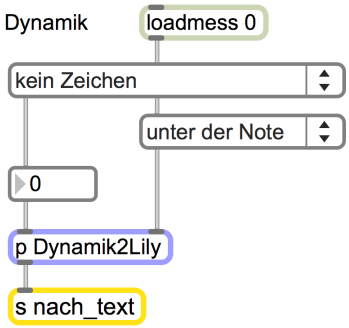
Mein Vorschlag für ein *Abstraction*-OBJECT



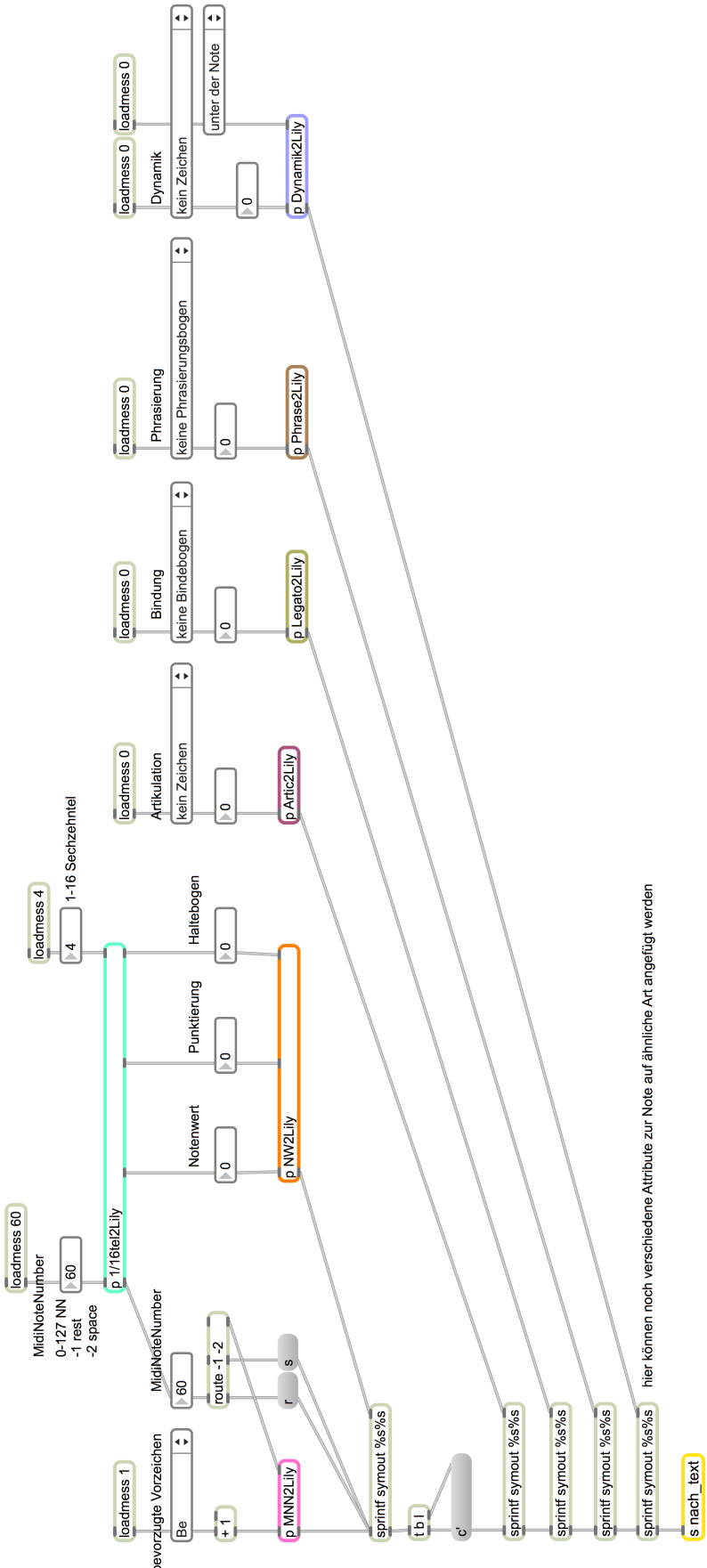
mit folgedem *Algorithmus*:


kein Zeichen
unter der Note
0
+ 1
loadmess 0
select 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
ppppp
pppp
ppp
pp
p
mp
mf
f
ff
fff
ffff
fffff
fp
sf
sff
sp
spp
sfz
rfz
<
>
!
t b l
" "
sfz
cresc
decresc
gate 2
sprintf symout \\%s
sprintf symout ^\\%s

Dynamische Zeichen können über oder unter der Note angebracht werden:



Die wichtigsten zusätzlichen Informationen zu einer Note in einem *Patch* zusammengefasst würde sich kompakt folgendermaßen zeigen.
Die Reihenfolge, wie dynamische Zeichen, Artikulationszeichen, Bindebögen, Phrasierungsbögen oder auch ähnliche Beifügungen nach einer Noteninformation zu reihen sind, spielt dabei für LilyPond keine Rolle.
Der Programmierer in MAX muss „nur“ darauf achten, die *Messages* und *Listen* richtig zu formen.


loadmess 60
MidiNoteNumber
0-127 NN
-1 rest
-2 space
60
p 1/16tel2Lily
loadmess 4
4
1-16 Sechzehntel
MidiNoteNumber
loadmess 1
bevorzugte Vorzeichen
Be
60
+ 1
route -1 -2
r
s
p MNN2Lily
Notenwert
0
p NW2Lily
Punktierung
0
Haltebogen
0
loadmess 0
Artikulation
kein Zeichen
0
p Artic2Lily
loadmess 0
Bindung
keine Bindebogen
0
p Legato2Lily
loadmess 0
Phrasierung
keine Phrasierungsbogen
0
p Phrase2Lily
loadmess 0
Dynamik
kein Zeichen
0
p Dynamik2Lily
loadmess 0
unter der Note
sprintf symout %s%s
t b l
c'
sprintf symout %s%s
sprintf symout %s%s
sprintf symout %s%s
hier können noch verschiedene Attribute zur Note auf ähnliche Art angefügt werden
sprintf symout %s%s
s nach_text

7 LilyPond LayOut

Die Struktur eines LilyPond-Dokuments ist genau definiert und muss klar sein, wenn wir das Layout für unser Notenbild ebenfalls mit Code generieren wollen. Die bisherigen Code-Schnipsel oder LilyPond-*Snippets*, wie sie genannt werden, müssen im vollständigen LilyPond-Dokument auch richtig platziert werden.

Die oberste Ebene in LilyPond ist die `\book`-Ebene. Innerhalb von `\book` werden ein oder mehrere *Scores* mit `\score` platziert. Jeder *Score* kann eigene Überschriften, die mit `\header` eingeleitet werden, tragen. Jede `\score`-Ebene kann mit eigenen Codes für das *Layout* mit der Einleitung `\layout` versorgt werden. Zwischen den `\score`-Abschnitten kann beliebig Text mit `\markup` eingefügt werden.

7.1 Die Abschnitte eines LilyPond-Dokuments

Jeder Abschnitt wird nach dem Befehl mit einer geschwungenen Klammer `"{"` eingeleitet und mit einer schließenden geschwungenen Klammer `"}"` beendet.

- `\version`
  Die Eingabe der Versionsnummer in das LilyPond-Dokument ist ratsam, damit die Kompatibilität unter den Versionen von LilyPond gewährleistet bleibt.

- `\book`
  steht auf der obersten Ebene des Dokuments. Für jeden `\book`-Befehl im LilyPond-Dokument wird ein eigener pdf-*File* ausgegeben. Wird kein

\book-Befehl angegeben, wird er vom LilyPond-Programm automatisch angenommen.

- \bookpart
  Das \book kann in verschiedene Abschnitte unterteilt werden. Mit jedem \bookpart-Befehl wird ein Seitenumbruch eingeleitet. Nach \bookpart könnten auch neue Befehle für die Seitenformatierung angegeben werden.

- \score
  Jeder \score-Befehl muss zumindest einen musikalischen Ausdruck in geschwungenen Klammern tragen. In einem \book können mehrere \score-Abschnitte aneinandergereiht werden.

- \markup
  Dieser Befehl kann zwischen die \score-Abschnitte gesetzt werden, um beliebigen Text einzufügen.

- \paper
  Mit diesem Befehl werden über 90 verschiedene Papiergrößen mit der Angabe von Hoch- oder Querformat definiert.

- \header
  Kopf- und Fußzeilen, wie Überschriften und sonstige Angaben zur Komposition und zum Komponisten werden in diesen Abschnitt geschrieben. Steht ein \header-Abschnitt in derselben Ebene wie \book – also auf der obersten Ebene – gelten die Überschriften für das gesamte Dokument. Steht er innerhalb eines \score-Abschnitts, dann gelten sie für den jeweiligen musikalischen Abschnitt. Zu beachten ist, dass die \header-Information innerhalb von \score, nach (!) dem musikalischen Ausdruck stehen muss.

- \layout
  Der Abschnitt \layout kann auf oberster Ebene mit \book oder auch innerhalb des \score-Abschnitts (hier ebenfalls jeweils unter (!) den mu-

sikalischen Ausdrücken) stehen. Je nach Platzierung gelten die Angaben für das ganze Dokument oder für den jeweiligen `\score`-Abschnitt.

- `\midi`
  Mit dieser Angabe wird das Notierte als Midi-*File* ausgegeben.

- Scheme-Befehle
  Scheme[1] ist die Computer-Sprache, die LilyPond versteht. Scheme-Befehle werden mit einem "`#`"-Zeichen eingeleitet und können an jeder Stelle des Dokuments platziert werden. Zum Beispiel werden Veränderungen der Zeichengröße üblicherweise auf diese Weise notiert.

- Kommentare können an jeder Stelle des Dokuments mit einem "`%`"-Zeichen eingeleitet werden. Die Gültigkeit besteht die restliche Länge der Zeile. Größere Abschnitte an Kommentaren werden mit "`% {`" begonnen und mit "`% }`" beendet.

- `\include`
  An jeder Stelle im Dokument können mit `\include` im selben Ordner abgespeicherte " .ly"-*Files* eingefügt werden. Das ist für größere Vorhaben eine wichtige Option.

- *Variable*
  Ganz ähnlich wie mit `\include` verhält es sich mit *Variablen* – die allerdings im Dokument selbst *deklariert* werden müssen.

Anschließend ist ein umfangreiches LilyPond-Dokument skizziert:

```
\version "2.16.2"

\book
{
  \header { ... }
  \layout { ... }

% \header und \layout gelten in dieser Ebene für das gesamte Dokument.
```

[1] http://www.schemers.org (Februar 2015).

```
  \bookpart
  {
    \paper { ... }
    \score
    {
      { ... } % musikalischer Ausdruck
      \header { ... }
      \layout { ... }
      \midi { ... }
    }

% weitere \score- oder \markup-Abschnitte hier einfügen

    \markup { ... }

    \score
    {
      { ... } % musikalischer Ausdruck
      \header { ... }
      \layout { ... }
      \midi { ... }
    }

% weitere \score- oder \markup-Abschnitte hier einfügen

  }

% Ende des \bookpart-Abschnitts

% weitere \bookpart-Abschnitte anschließen
% Seitenumbruch mit neuem \bookpart
  \bookpart
  {
    .
    .
    .
  }
}
% Abschluss des gesamten \book-Abschnitts

% eventuelle weitere \book-Abschnitte hier anschließen
% jeder \book-Abschnitt erzeugt einen eigenen pdf-File

\book
```

```
{
     .
     .
     .
}

\book
{
     .
     .
     .
}
```

Nach dem Befehl `\bookpart` wird ein Seitenumbruch vorgenommen.
Jeder `\book`-Block generiert einen eigenen pdf-*File*.

Es müssen nicht immer alle Befehle gesetzt werden. Wenn es sich um „nur“ ein `\book` handelt, kann der `\book`-Befehl auch weggelassen werden. Genügen die Voreinstellungen, müssen auch keine `\layout`- oder eventuelle `\paper`-Einstellungen vorgenommen werden.
LilyPond erstellt die fehlenden Angaben programmintern.

Gäbe man in das LilyPond-Dokument nur einen musikalischen Ausdruck ein – dieser könnte in kleinerem oder durchaus größerem Umfang sein – zum Beispiel:

```
{ c'4 d' e' f' }
```

so würde in LilyPond intern folgender Code automatisch angenommen:

```
\book
{
  \score
  {
    \new Staff
    {
      \new Voice
      {
        { c'4 d' e' f' }
      }
    }
```

```
  \layout { }
  }
  \paper { }
  \header { }
}
```

Wie die geschwungenen Klammern gesetzt und die Einrückungen vorgenommen werden, ist Geschmacksache, soll aber der Übersichtlichkeit dienen.
Mehr als eine Leerzeichen (*space*) oder eine Tabulator-Einrückung (*tab*) oder eine Zeilenschaltung (*new line*) wird ignoriert. Hier spricht man auch von "whitespace insensitive" – im Gegensatz zu "case sensitive" – wobei LilyPond zwischen Groß- oder Kleinschreibung unterscheidet.

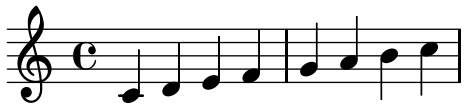
Vor und nach geschwungenen Klammern sollte ein Leerzeichen stehen, es könnte ansonsten fallweise zu Fehlermeldungen kommen.

So wie bei allen Programmiersprachen ist das richtige Setzen der Klammern eine heikle Angelegenheit. Ihre Position ist aussagekräftig und die Anzahl der öffnenden muss mit der Anzahl der schließenden Klammern exakt übereinstimmen.

Es liefern folgende minimale `\score`-Beispiele das gleiche Ergebnis.

```
\score
{
  { c' d' e' f' g' a' b' c'' }
}

\score
{
  { { c' d' e' f' g' a' b' c'' } }
}

\score
{
  { { { c' d' e' f' g' a' b' c'' } } }
}

\score
{
```

```
  { { c' d' e' f' } { g' a' b' c'' } }
}

\score
{
  { { { c' } { d' } { e' } { f' } } { g' a' b' c'' } }
}
```



7.2 LilyPond Score

Wesentlich ist das Verständnis der Gliederungen des `\score`-Abschnitts. Notensysteme (*Staves*) werden mit ihrer Funktion hier bestimmt. Handelt es sich um ein einzelnes System, muss nichts weiter angegeben werden. Handelt es sich um ein mehrstimmiges Stück, muss mit `\new Staff` jede zusätzliche Notenzeile eingeleitet werden. Sind mehrere Stimmen in einer Notenzeile – also beispielsweise bei Klavierauszügen – muss mit `\new Voice` eine neue "Ebene"[1] definiert werden.
Gleichzeitig erklingende Notenabschnitte werden dabei immer mit "<<" eingeleitet und mit ">>" beendet.

```
\score
{
<<
  \new Staff
  {
  <<
    \new Voice
    {
    \voiceOne { ... }
    }
    \new Voice
    {
    \voiceTwo { ... }
    }
    .
```

[1]Bei anderen Notationsprogrammen wird zu diesem Zwecke eine neue "Ebene" eingeführt.

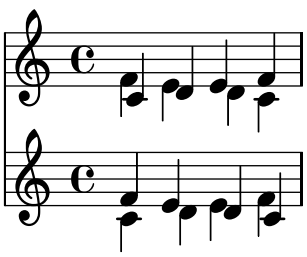
```
    .
    .
  >>
  }
  \new Staff
  {
  <<
    \new Voice
    {
    \voiceOne {...}
    }
    \new Voice
    {
    \voiceTwo {...}
    }
    .
    .
    .
  >>
  }
  .
  .
  .

>>
}
```

Zum Beispiel:

```
\score
{
<<
  \new Staff
  {
  <<
    \new Voice
    {
    \voiceOne { c' d' e' f' }
    }
    \new Voice
    {
    \voiceTwo { f' e' d' c' }
    }
```

```
  >>
  }
  \new Staff
  {
  <<
    \new Voice
    {
    \voiceTwo { c' d' e' f' }
    }
    \new Voice
    {
    \voiceOne { f' e' d' c' }
    }
  >>
  }

>>
}
```

...woraus resultiert:



7.3 Variable

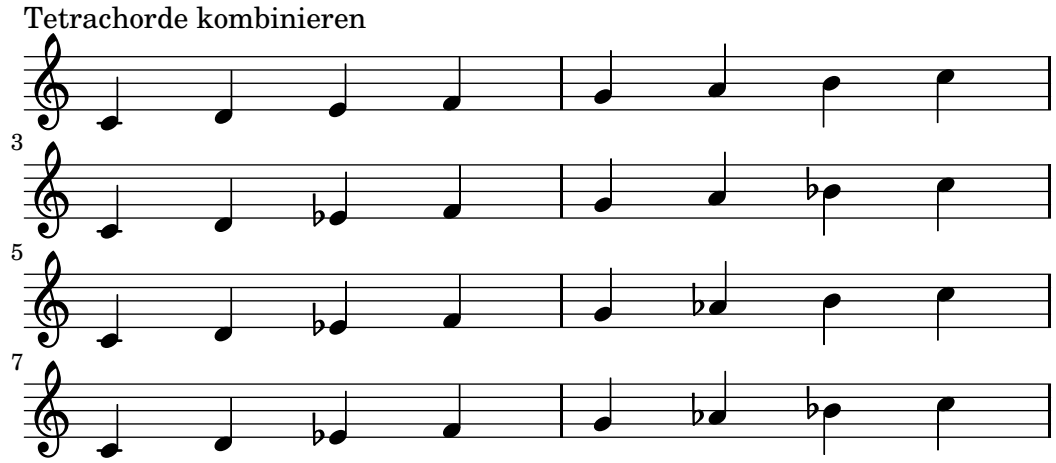
Als vorteilhaft kann sich der Umgang mit *Variablen* erweisen. *Variable* definieren musikalische Abschnitte und können im Verlauf des *Scores* beliebig eingesetzt werden. Zu beachten ist, dass der *Variable*n-Name nur Buchstaben und keine Zahlen, Unter- oder Bindestriche enthalten darf.

Beispielsweise führt:

```
tetraDUR = { c' d' e' f' }
tetraMOLL = { c' d' ees' f' }
tetraHM = { c' des' e' f' }
```

```
tetraINVDUR = { c' des' ees' f' }

\header { piece = "Tetrachorde kombinieren" }
\markup { \vspace #1 }
\score
{
  \new Staff \with
  {
  \remove "Time_signature_engraver"
  }
  {
  \tetraDUR \transpose c g \tetraDUR
  \break
  \tetraMOLL \transpose c g \tetraMOLL
  \break
  \tetraMOLL \transpose c g \tetraHM
  \break
  \tetraMOLL \transpose c g \tetraINVDUR
  }

}
```

...zu diesem Ergebnis:



7.4 \paper

Für das Seiten-Layout sind die `\paper`-Einstellungen verantwortlich. Sie stehen auf der obersten Ebene mit `\book` oder können unter `\bookpart` in einem Dokument variiert werden.

Hier können viele Einstellungen vorgenommen werden, die für das ganze Dokument gelten.
Zum Beispiel:

```
\paper
{
top-margin = 30
bottom-margin = 30
left-margin = 40
right-margin = 25
}
```

...für die Seitenränder.
Oder:

```
\paper
{
indent = #0
line-width = #120
ragged-right = ##t
ragged-bottom = ##t
}
```

...für Zeileneinzüge, Zeilenlängen und Zeilenabschlüsse.

Wenn keine Maßeinheiten angegeben sind, gelten die Angaben in Millimeter.

Um Notensysteme auf dem Notenblatt richtig zu verteilen, sind `\paper`-Angaben wie
`max-systems-per-page = #...` oder
`min-systems-per-page = #...` oder
`systems-per-page = #...` sowie
`system-count = #...`
zu notieren.

In diesem Zusammenhang könnten auch Feinjustierungen wie
`markup-system-spacing = #...`
`score-markup-spacing = #...`
`score-system-spacing = #...`
`system-system-spacing = #...`
`markup-markup-spacing = #...`
`last-bottom-spacing = #...`
`top-system-spacing = #...`
`top-markup-spacing = #...`
eine Rolle spielen.

Für Einstellungen der Abstände zwischen den Notensystemen hilft beispielsweise:

```
\paper
{
between-system-space = 1.7\cm
between-system-padding = #1
page-limit-inter-system-space = ##t
page-limit-inter-system-space-factor = 1.2
}
```

7.5 \layout

Um beispielsweise die Größe des Notensystems mit der einhergehenden Skalierung aller Zeichensätze zu bestimmen, kann auf oberster Ebene der *Scheme*-Befehl
`#(set-global-staff-size 14)`
geschrieben werden.
Auf dieser Ebene wird auch das Papierformat für das gesamte Dokument festgelegt. Zum Beispiel:
`#(set-default-paper-size "a4" 'landscape)`
`'landscape` für Querformat und `'portrait` für Hochformat.

Sollen einzelne Abschnitte in unterschiedlichen Zeichengrößen erscheinen, muss innerhalb der `\score`-Ebene, unter (!) der Musikinformation, in einem `\layout`-Block, mit folgendem CODE

```
\score
{
  { ... Musikinformation ... }
  \header { ... eventuelle Angaben ... }
  \layout
  {
  #(layout-set-staff-size 16)
  }
}
```

die Zeichengröße verändert werden.

7.6 \header

Der `\header`-Block trägt Angaben zu:
`title = "Mein großes Meisterwerk"`
oder auch:
`subtitle = " ... "` und
`subsubtitle = " ... "`

oder Angaben zu:
`composer = " ... "`
`arranger = " ... "`
`instrument = " ... "`
`piece = " ... "` oder
`poet = " ... "`

Jede Rubrik hat ihren voreingestellten Platz, sein Größenverhältnis und ist zentriert, links oder rechts angeordnet.

Des Weiteren werden im `\header`-Block Einstellungen zu
`copyright = " ... "`

oder
`tagline = ##t` oder `##f` für ***t****rue* oder ***f****alse*
getroffen.

Für den optimalen Umgang mit LilypPond ist ein Studium der dazugehörigen Dokumentationen unerlässlich.
Neben diversen Büchern zur Einführung werden in drei großen Werken, welche online als HTML-Dokumente oder in Form von pdf-*Files* zum Download zur Verfügung stehen, sämtliche Notations- und Formatierungsmöglichkeiten auf über 1500 Seiten mit unzähligen Beispielen erläutert.

- Notation Reference[1]
- Usage[2]
- Snippets[3]

Ich habe hier versucht, für unsere Thematik das vorerst Wichtigste zu umreißen und herauszufiltern.

[1] http://lilypond.org/doc/v2.18/Documentation/notation.pdf (Februar 2015).
[2] http://lilypond.org/doc/v2.18/Documentation/usage.pdf (Februar 2015).
[3] http://lilypond.org/doc/v2.18/Documentation/snippets.pdf (Februar 2015).

8 Die Partitur auf „Knopfdruck“

Hier werde ich die wesentlichen Bestandteile einer Partitur[1] in LilyPond anhand eines Minimalbeispiels mit Hilfe von MAX zusammenführen.
Die Abschnitte wie `\paper`, oder `\header` werden idealerweise mit dem *texte-dit*-Object erstellt. Für die musikalischen Abschnitte gewöhnt man sich besser an, *Variable* einzuführen.

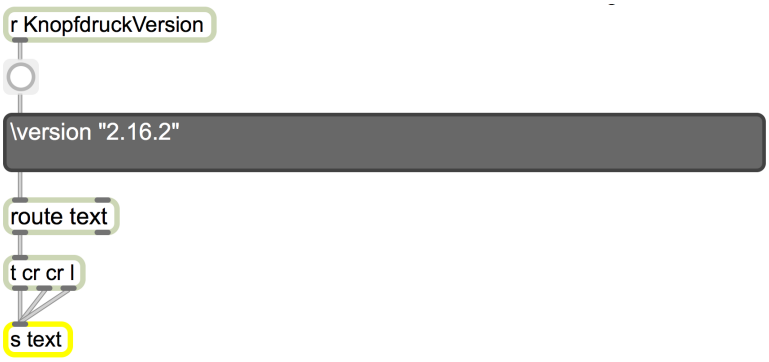
Die Code-Teile der Partitur werden in MAX organisiert und automatisiert in ein *text*-Object befördert. Der resultierende Code kann von dort kopiert, in LilyPond eingesetzt und anschließend mit der Tastenkombination "cmd+R" für "Typeset file" in LilyPond *kompiliert* werden. Oder man schickt mit einem abschließenden *Bang* – automatisiert oder „händisch“ – eine "write"-*Message* zum *text*-Object. Wir erstellen damit einen Text-*File* mit der Endung ".ly". Dieser *File* kann anschließend wieder in LilyPond geöffnet werden.

In einem abschließenden Kapitel werden wir die Prozedur des Abspeicherns und wieder Öffnens samt der Umwandlung in einen pdf-*File* noch automatisieren.

Im folgende Beispiel werden wir die gesamte Partitur mit einem „Knopfdruck“ in das abgebildete *text*-Object befördern. Dort angelangt, lässt sich die Partitur mit einer *Pfad*-Angabe an einen bestimmten Ort der Festplatte des Computers oder auch ohne *Pfad*-Angabe – so wie im Bild – in den Ordner des MAX-*Patches* speichern.

[1] ital. *partitura* „Einteilung“.

Die einzelnen Abschnitte der Partitur werden in ein *textedit*-OBJECT geschrieben.



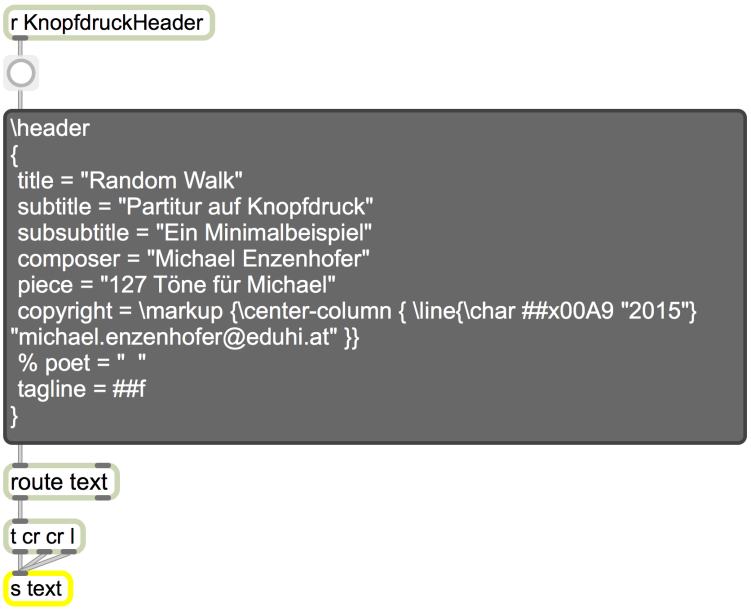
Beim *textedit*-OBJECT ist wichtig, die voreingestellte Eigenschaft mit der *Message* "outputmode 1" umzuschalten. Damit wird im *textedit*-OBJECT "Output as One Symbol" aktiviert.
Das ist notwendig damit "\" und "," sowie Zeilenumschaltung (*carriage return* oder "cr") in der richtigen Darstellung zum *text*-OBJECT gesendet werden.

Im *Help-Patch* des *textedit*-OBJECTS ist diese Einstellung schon vorgenommen. Zur Verdeutlichung dieses notwendigen Schrittes, sind in meinen angeführten Beispielen die *textedit*-OBJECTS im selben Grauton eingefärbt wie im *Help-Patch*.

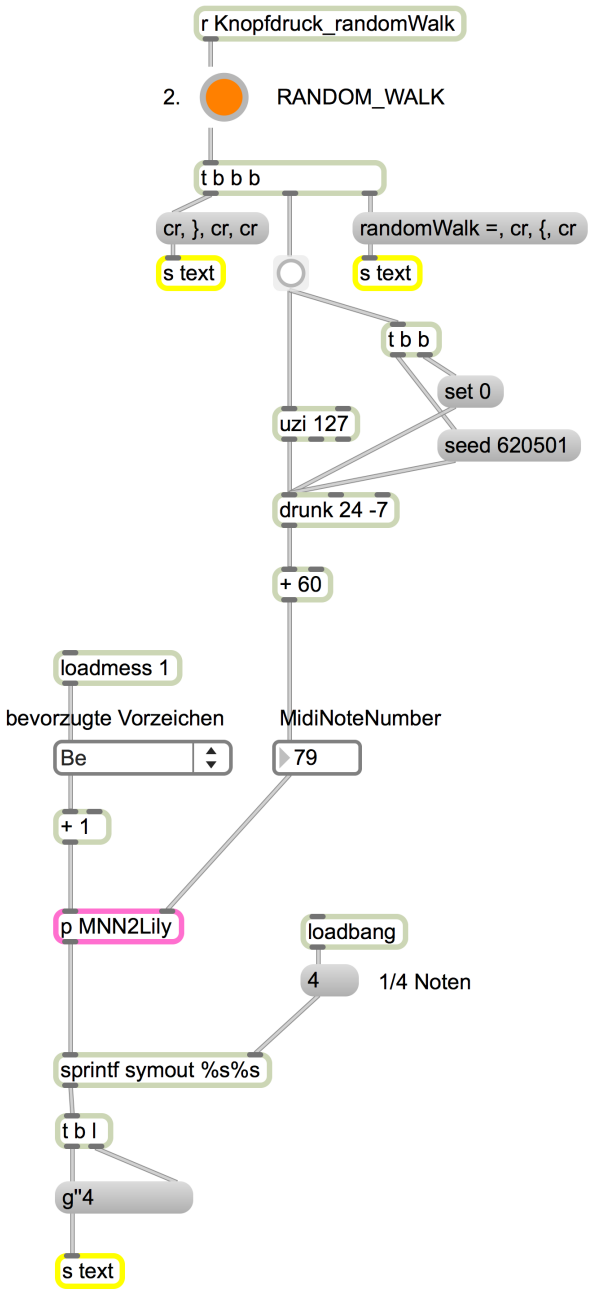
Jeder von LILYPOND gewünschte Abschnitt kann auf ähnliche Weise in einem weiteren *textedit*-OBJECT definiert werden.
Mein Vorschlag für globale Einstellungen:

Einstellungen für `\paper`

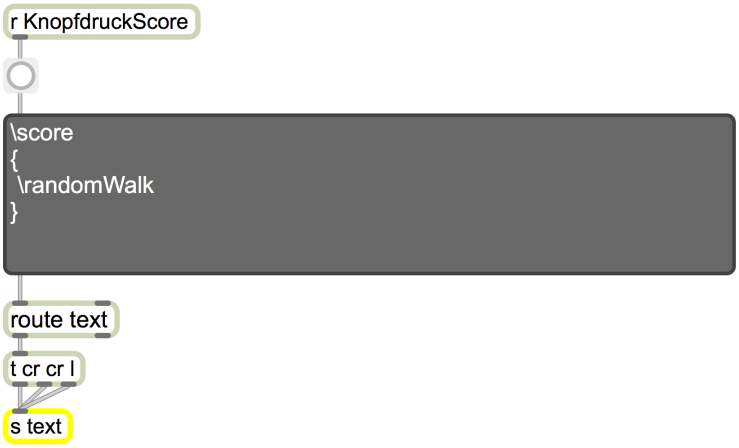
. . .und `\header`



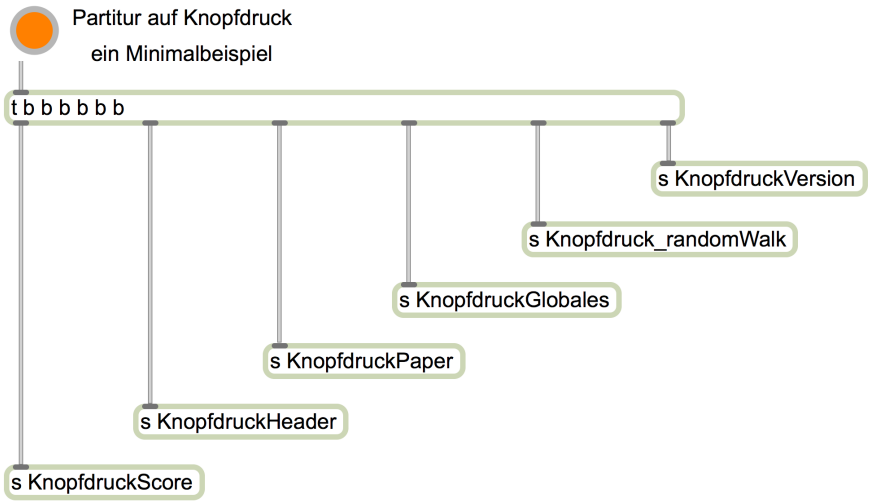
Die eigentliche musikalische Notation wird im Beispiel in MAX generiert und einer *Variablen* zugewiesen, in diesem Fall dem frei gewählten *Variable*n-Namen "randomWalk".



Die *Variable* wird in einem weiteren Feld für `\score` im *textedit*-Object organisiert eingefügt. Bei größeren Partituren würden hier mehrere bis viele *Variable* im „Spiel“ sein.



Beim entscheidenden „Knopfdruck“ werden die einzelnen Bestandteile der Partitur an die richtige Position im *text*-Object befördert.



Nach dem großen *Bang* werden mit dem *trigger*-Object die einzelnen *Bangs* von RECHTS nach LINKS ausgegeben.

Der in das *text*-Object beförderte LilyPond-Code sieht genau so aus:

```
\version "2.16.2"

randomWalk =
{
f'4 a'4 ees''4 a''4 e''4 d''4 ees''4 e''4 bes''4 g''4 des''4 d''4 f''4
e''4 g''4 b''4 f''4 b'4 des''4 g''4 c'''4 g''4 des''4 ees''4 aes''4
ees''4 des''4 e''4 bes'4 ges'4 bes'4 aes'4 a'4 ges'4 a'4 g'4
bes'4 f'4 aes'4 bes'4 b'4 f''4 des''4 g''4 f''4 g''4 des''4 ees''4
aes''4 d''4 aes''4 d''4 des''4 f''4 ees''4 f''4 b''4 aes''4 ees''4
f''4 a''4 e''4 f''4 b'4 aes'4 ges'4 g'4 des'4 e'4 a'4 ees'4 aes'4
e'4 d'4 c'4 ees'4 f'4 aes'4 bes'4 ees''4 bes'4 ees''4 a'4 ees'4
f'4 aes'4 d''4 b'4 des''4 g''4 ges''4 c'''4 ges''4 e''4 g''4 d''4 f''4
c''4 ees''4 aes''4 ees''4 d''4 ees''4 g''4 ees''4 e''4 ges''4 des''4
aes'4 d''4 b'4 des''4 bes'4 aes'4 c''4 bes'4 g'4 f'4 b'4 a'4 b'4
f''4 des''4 ees''4 a''4 aes''4 g''4
}

#(set-global-paper-size "a4" 'portrait)
#(set-global-staff-size 14)

\paper
{
  indent = 0
  ragged-right = ##f

  left-margin = 50
  right-margin = 50
  bottom-margin = 30
  top-margin = 80

  %line-width = 150
}

\header
{
  title = "Random Walk"
  subtitle = "Partitur auf Knopfdruck"
  subsubtitle = "Ein Minimalbeispiel"
  composer = "Michael Enzenhofer"
  piece = "127 Töne für Michael"
  copyright = \markup {\center-column { \line{\char ##x00A9 "2015"}
"michael.enzenhofer@eduhi.at" }}
  % poet = "  "
```

```
  tagline = ##f
}


\score
{
  \randomWalk
}
```

Der CODE kann wie beschrieben in LILYPOND geöffnet und *kompiliert* werden.

Er liefert nachfolgenden pdf-*File*:

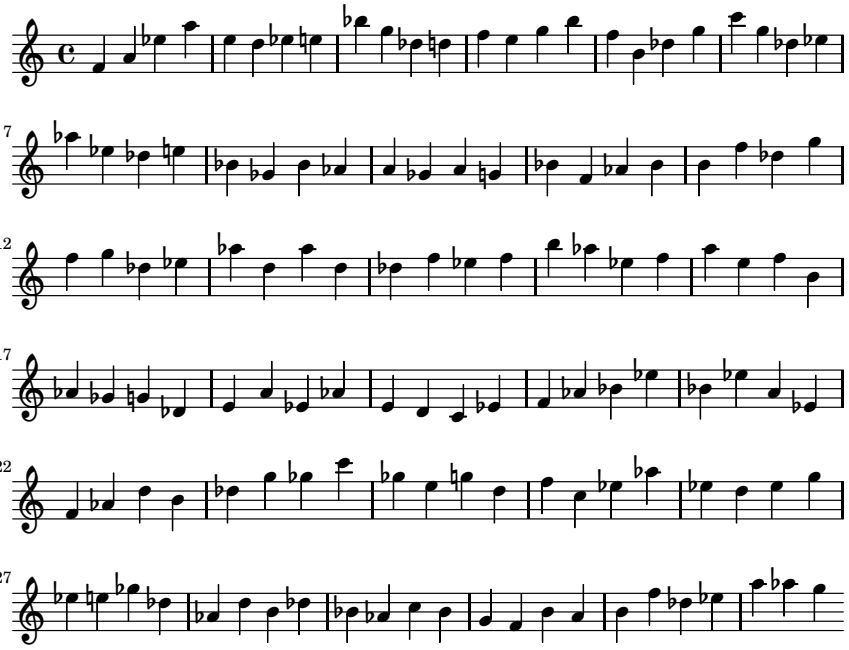
Random Walk

Partitur auf Knopfdruck

Ein Minimalbeispiel

Michael Enzenhofer

127 Töne für Michael



michael.enzenhofer@eduhi.at

9 Mit Shell zum Ziel

Oder: UNIX in MAX

Meist wird man noch „Hand“ anlegen wollen oder müssen, um den CODE den Wünschen und Bedürfnissen nach zu korrigieren, zu ordnen, zu ergänzen, zu manipulieren, zu verschachteln oder um ihm noch den abschließenden „Feinschliff“ zu verpassen.

Es liegt auf der Hand, dass mit jedem „Knopfdruck“ – insbesondere wenn die programmierten *Algorithmen* den Zufall zulassen – eine neue, aber vielleicht ähnliche Partitur erzeugt werden kann. Es wäre nicht nur sportlicher Ehrgeiz, wenn mit jedem „Knopfdruck“ eine neue Partitur auf der Festplatte gespeichert und eventuell auch gleich geöffnet würde oder wenn gewünscht – mit oder ohne sich zu öffnen – gleich ausgedruckt würde.

Realisierbar werden diese Vorhaben, wenn wir die Möglichkeiten, die uns von der LINUX[1]- oder UNIX[2]-Welt bekannt sind, integrieren.
Befehle in UNIX konnten ursprünglich vom Anwender nur per Tastatureingabe über eine Kommandozeile eingegeben werden. Erst später wurde eine grafische Benutzeroberfläche und auch die Möglichkeit der Bedienung mit der Maus implementiert.

[1] LINUX oder GNU/LINUX ist ein UNIX-ähnliches Computer Betriebssystem.
http://de.wikipedia.org/wiki/Portal:Linux (Februar 2015).
[2] UNIX ist eines der verbreitetsten und einflussreichsten Betriebssysteme der Computergeschichte.
http://de.wikipedia.org/wiki/Unix (Februar 2015).

„*Everything is a file*“[1] beschreibt eine wesentliche Eigenschaft von UNIX oder LINUX.
Das heißt, dass wesentlichste Einstellungen und auch Prozeduren im System als *File* abgespeichert sind. Diese *Files* können von der Kommandozeile aufgerufen oder auch manipuliert werden. Mit anderen Worten lässt sich der Computer mit seiner Software und auch seiner Hardware mit Kommandos steuern.

Die Benutzerschnittstelle mit der Eingabemöglichkeit der Kommandos wird als Terminal, SHELL[2], Kommandozeileninterpreter, Eingabeaufforderung oder auch Konsole bezeichnet.

Der Benutzer tippt in einer Eingabezeile Kommandos – oder Befehle – ein, die der Computer nach einem ENTER ausführt.

Auch wenn es im gewöhnlichen Betrieb kaum auffällt, ist beim etwas mehr verbreiteten APPLE-Computer samt Betriebssystem seit den Versionen von OSX UNIX unter der grafischen Oberfläche. Dieser Unterbau, den Apple als «UNIX Foundation» mit dem Namen «Darwin» bezeichnet, gilt als die eigentlich revolutionärste Entwicklung mit OSX.[3]

Das „Tor“ zur UNIX-Welt ist bei einem APPLE-Computer im *Dienstprogramme*-Ordner zu finden und nennt sich erwartungsgemäß *Terminal*.

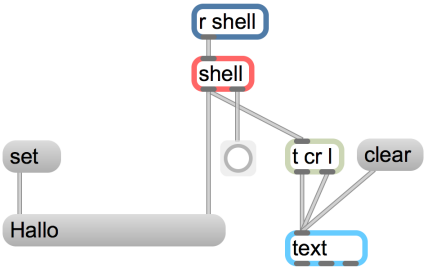
Über den Weg des *Terminals* lässt sich also der Rechner steuern, und damit lassen sich auch Programme – beziehungsweise *Applications* – starten oder beenden, Ordner erzeugen oder löschen, *Files* schreiben, verschieben oder auch manipulieren und beispielsweise Druckaufträge senden.

Die Bedienung über das *Terminal* mit den UNIX-Kommandos wäre für un-

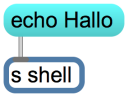
[1] Vergleiche: http://de.wikipedia.org/wiki/Everything_is_a_file (Februar 2015).
[2] englisch für Hülle oder Schale
[3] Vergleiche: Andreas Heer: *Panther für Profis*. Midas (Zürich 2004) S. 17.

ser Vorhaben doch auch umständlich, hätte nicht JEREMY BERNSTEIN[1] das *shell*-OBJECT für MAX entwickelt, welches SHELL-Befehle[2] versteht und auszuführen in der Lage ist. Es ist das mächtigste – das wage ich hier zu behaupten – OBJECT, das für MAX jemals zur Verfügung gestellt wurde. Es eröffnet Möglichkeiten, die zu beschreiben viele Bücher nicht imstande wären.

Wir haben somit mit SHELL-Kommandos erstens die Allmacht über den Rechner und zweitens von MAX ausgehend.



Wir begnügen uns mit einigen wenigen Befehlen aus der UNIX-Welt, um unser Vorhaben realisieren zu können.



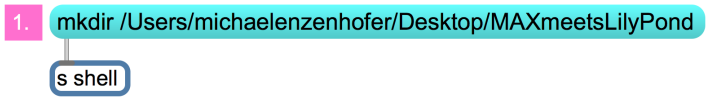
Einer der gängigsten Befehle ist `echo`. Dieser gibt den Inhalt, der nach dem Befehl steht, direkt bei der *Standardausgabe* (kurz: *stdout*) aus. Die Ausgabe von *stdout* erfolgt beim *shell*-OBJECT an seinem linken Ausgang. Wenn ein Befehl „geschrieben“ wurde, gibt es am rechten Ausgang einen *Bang*.

[1] JEREMY BERNSTEIN ist geboren in Kolumbien, studierte Komposition am *College Conservatory of Music* (Ohio) und lebt seit 2004 in Berlin. Er ist MAX-User seit 1991, C74 co-worker seit 1999, C74 Entwickler seit 2010.

[2] Manche Befehle benötigen besondere Rechte am Computer. Um diese Rechte zu aktivieren ist hier die Vorgangsweise zu finden:
http://support.apple.com/de-at/ht1528 (Februar 2015).

Die Ausgabe von *stdout* wird in eine *Message*-Box (zur Anzeige der letzten Zeile) oder zur vollständigen Einsicht wieder in ein *text*-OBJECT geschickt. Hier kann `echo` zur Überprüfung dienen, ob das *shell*-OBJECT soweit schon funktioniert.

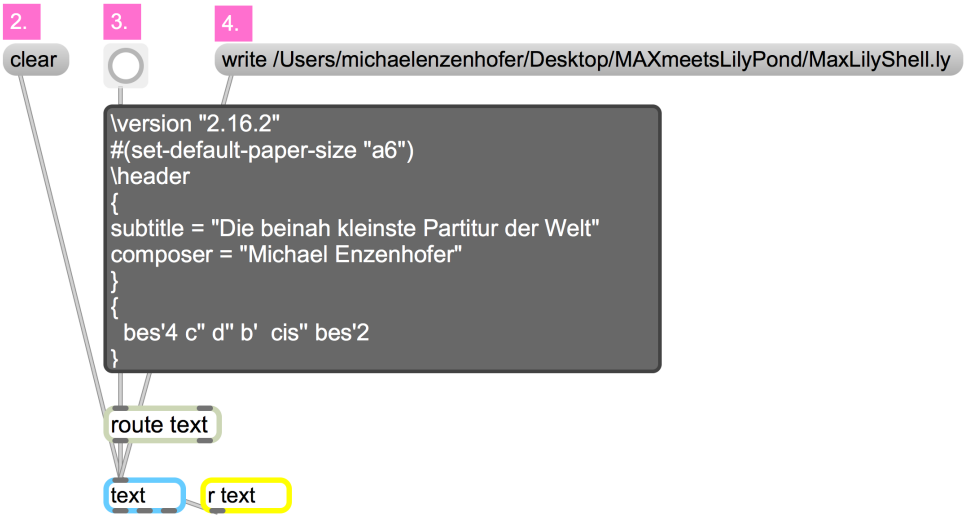
Ein erster nützlicher Befehl ist `mkdir`. Dieser steht für "**mk**e **dir**ectories"



Zugegebender Maßen ist diese Vorgangsweise nicht wirklich notwendig, weil ein Ordner auch auf übliche Weise erstellt werden kann. Als erste einfache Übung mit unserem *shell*-OBJECT ist sie gut geeignet.

`mkdir` erstellt also einen leeren Ordner am Computer mit dem angegebenen Namen – in unserem Fall: `MAXmeetsLilyPond`. Vor dem Namen des Ordner muss der *Pfad* zum Ordner ohne Abstand angeführt werden – in unserem Fall soll er auf dem *Schreibtisch* zu liegen kommen –
also: `/User/michaelenzenhofer/Desktop/`

Zur Veranschaulichung der gesamten Prozedur ein kleines Beispiel:



Die „Partitur“ wird wieder in ein *text*-OBJECT geschrieben und mit der "write"-*Message* – mit Angabe des *Pfades* und des *File*-Namens– gleich in unseren neu erstellten Ordner transferiert.

Der wesentlichste Schritt besteht nun in der Umwandlung des erzeugten ".ly"-Text-*Files* in einen pdf-*File*.

Damit dieses Vorhaben von der SHELL und somit auch von unserem dazugewonnenen *shell*-OBJECT möglich wird, ist Voraussetzung, dass sich LILYPOND selbst von einer *Kommandozeile* steuern lässt.

In diesem Fall besteht Hoffnung, weil man bei der Suche nach *Command-Line* auf der LILYPOND-Online-Dokumentation fündig wird. Etliche Kommandos zur Steuerung von LILYPOND stehen zur Verfügung.[1]

Mit sehr sparsamer Auskunft ist hier folgender Hinweis zu finden:
«The `lilypond` executable may be called as follows from the command line.»
und dann dieser etwas kryptische *Pseudo*-CODE:
`lilypond [option]... file...`

Was für langjährige UNIX-*User* ein „Spaziergang“ ist, ist für „normale“ Computer-Anwender ein Hürdenlauf.

Man muss wissen, dass in diesem vorgegebenen CODE `lilypond` der eigentliche Befehl oder das Kommando ist.
Unter einer Menge von angeführten *Optionen* zum `lilypond`-Befehl auf der Online-Dokumentation ist eine vielversprechend:
`-o, --output=FILE or FOLDER`

[1] http://www.lilypond.org/doc/v2.18/Documentation/learning/command_002dline (Februar 2015).
http://www.lilypond.org/doc/v2.18/Documentation/usage/command_002dline-usage (Februar 2015).

Der Hinweis zu dieser *Option* lautet:
«Set the default output file to FILE or, if a folder with that name exists, direct the output to FOLDER, taking the file name from the input file. The appropriate suffix will be added (e.g. .pdf for pdf) in both cases.»

Bei UNIX-Kommando-*Optionen*, die üblicherweise nach dem Befehl geschrieben werden, gibt es meist eine Kurz- und eine Langform.

Die Kurzform für unsere *Option* heißt `-o` und die Langform `--output`. Die Langform wird üblicherweise mit zwei Bindestrichen vor der *Option* eingeführt, die Kurzform mit einem.

Unser Kommando muss also mit unserer gewünschten *Option* sinngemäß so lauten:
`lilypond -o FOLDER filename.ly`
oder gleichbedeutend:[1]
`lilypond --output=FOLDER filename.ly`

`filename.ly` ist stellvertretend der Name von einem von uns erstelltem ".ly"-*File*.

Das sieht soweit noch übersichtlich aus, es muss allerdings noch eines beachtet werden:
Da in UNIX "alles *File*" ist, ist auch der Befehl ein *File* und liegt auf der Festplatte. *Files* auf der Festplatte haben einen genauen *Pfad*, deswegen muss der *Pfad* des Befehls eingegeben werden.

LilyPond-Pfade

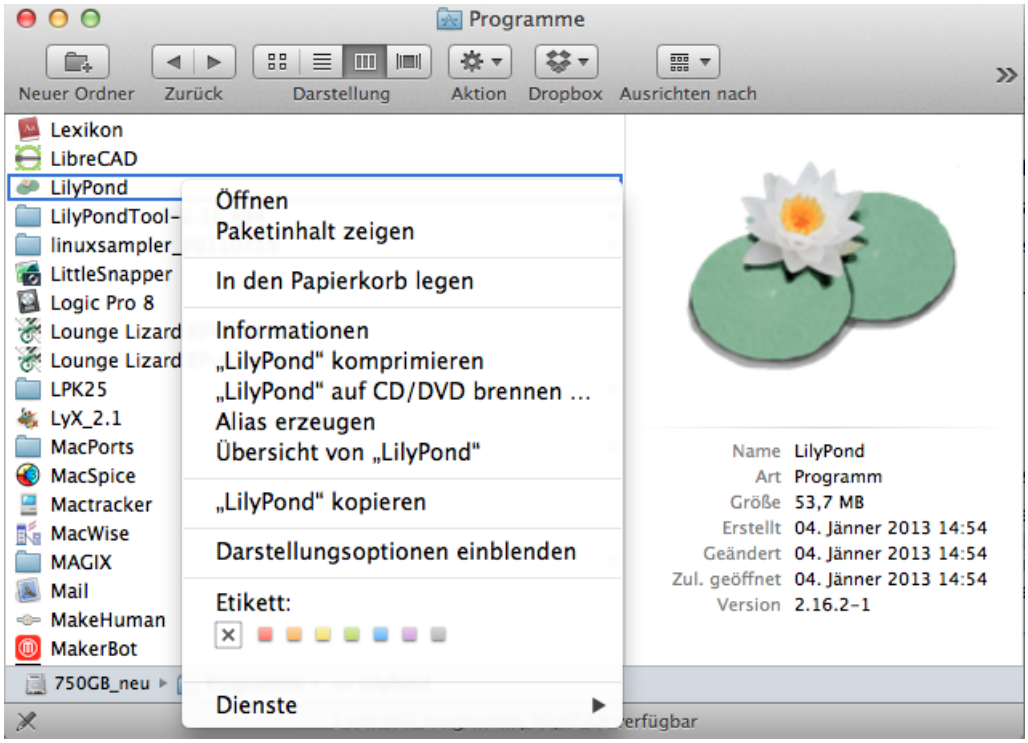
OSX möchte sich als sicheres Betriebssystem erweisen. Um das zu gewährleisten, versteckt es unter anderem Programm- oder Systembestandteile vor

[1] (Nach `-o` darf ein Leerzeichen stehen – muss aber nicht [durch Ausprobieren gefunden]).

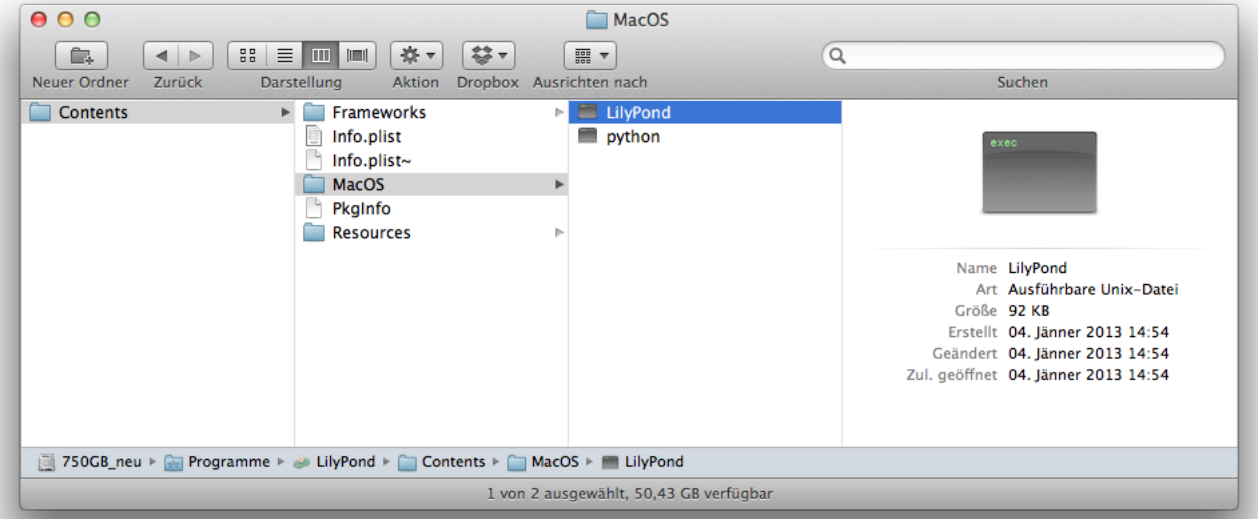
dem vielleicht unachtsamen Benützer. Es könnte zu Verschiebungen oder versehentlichen Löschungen von zur Funktion oder zur System-Sicherheit wichtigen Bestandteilen kommen.

So ist oft das Programmsymbol im *Programme*-Ordner nicht das eigentliche Programm, sondern eine Verknüpfung – obwohl es nicht so scheint.

Die *Pfade* der verborgenen Befehle und Programmbestandteile finden wir, indem wir den Inhalt des Programms offenlegen.
Dazu ist es notwendig, das Programmsymbol – in unserem Fall LilyPond – anzuwählen und mit der rechten Maustaste "Paketinhalt zeigen" auszuwählen.



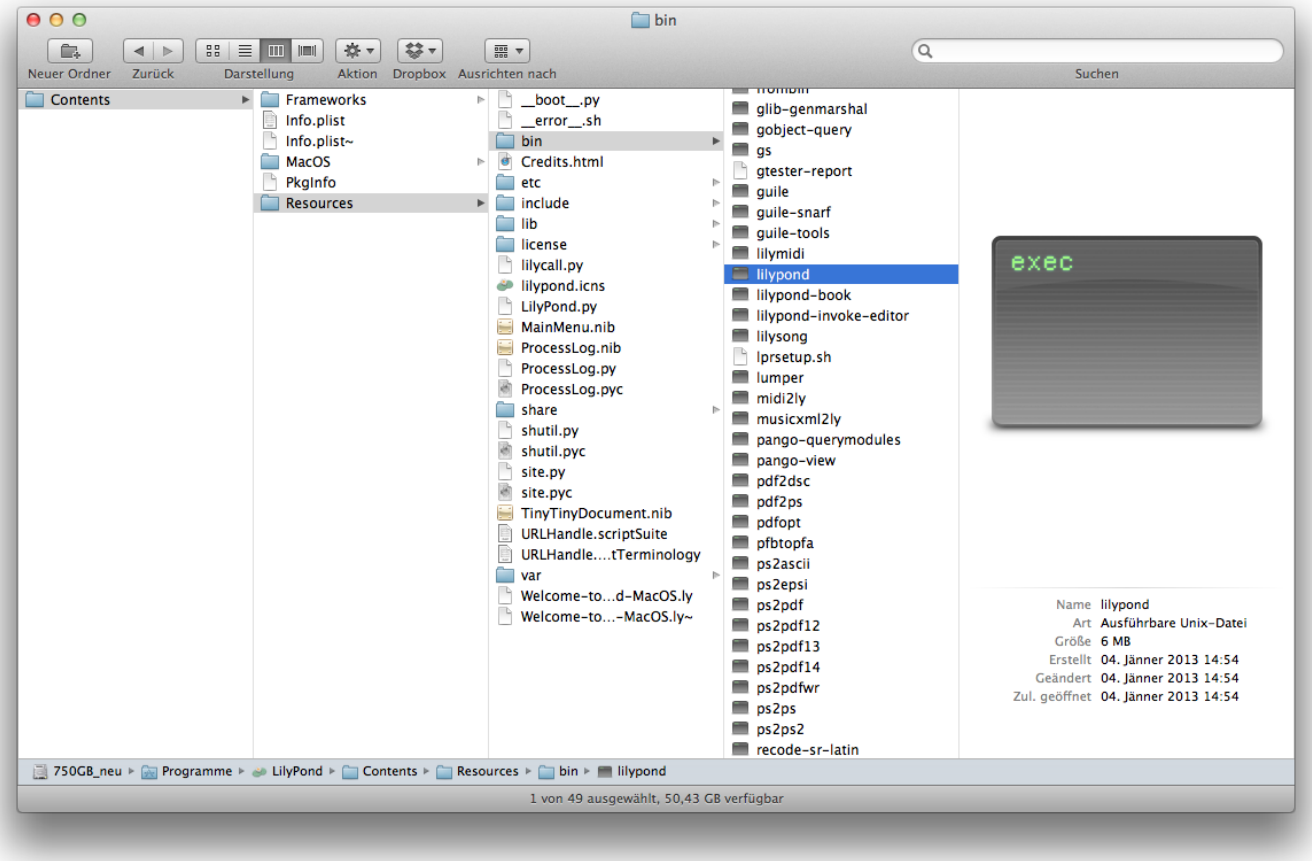
Darauf zeigen sich die *Contents*.
Es sollte bewusst sein, dass der *Pfad* zum eigentlichen Programm hier liegt:



Also:

`"persönlicheFestplatte"/Programme/LilyPond/Contents/MacOS/LilyPond`

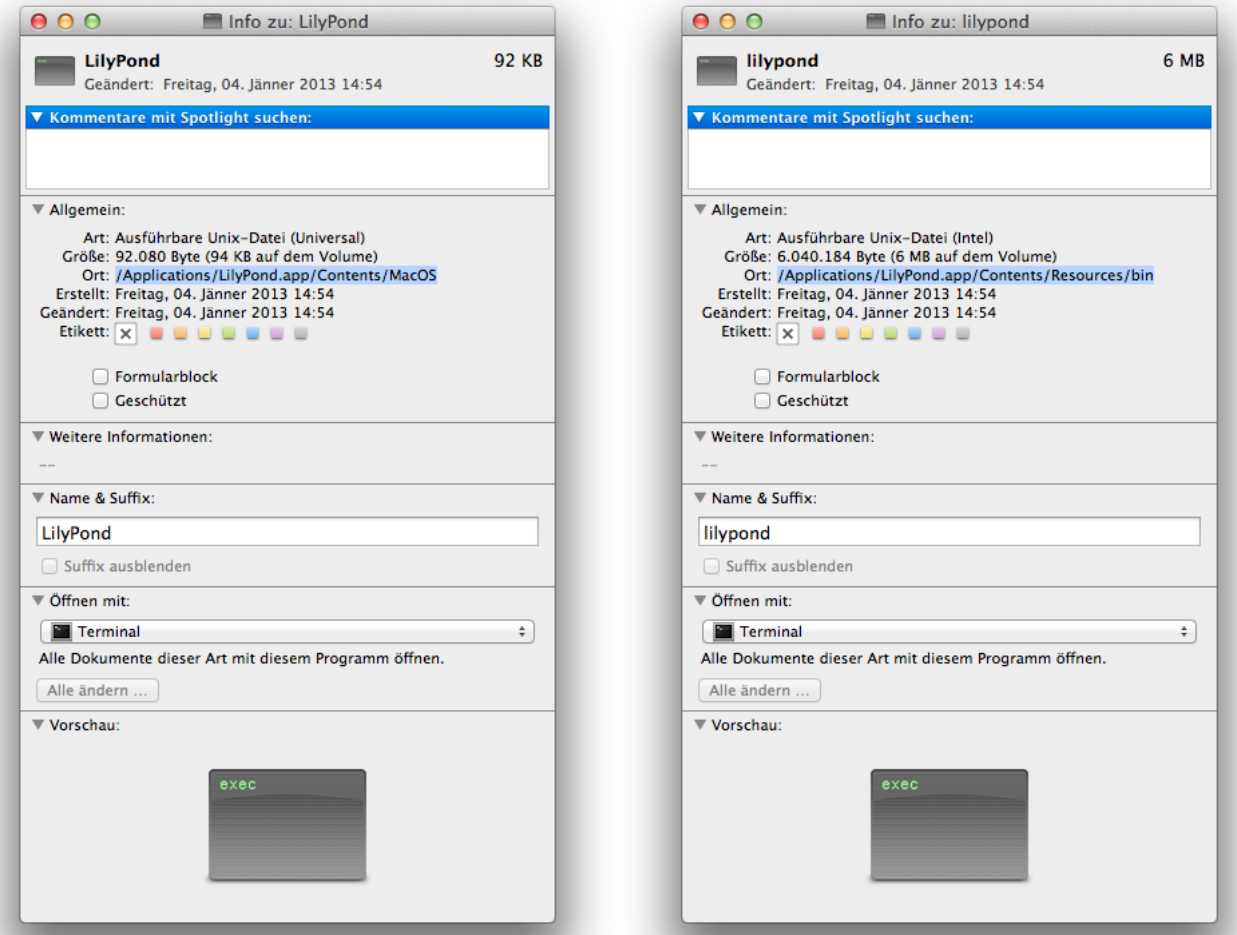
Der *Pfad*, den wir zur Umwandlung des LilyPond-Codes in einen pdf-*File* benötigen – also der `lilypond`-Befehl – ist hier zu finden:



Mit dem *absoluten Pfad*:

```
"persönlicheFestplatte"/Programme/LilyPond/Contents/Resources/bin/lilypond
```

Die Pfade sind jeweils am unteren Rand der Fenster ersichtlich.
Allerdings sind es nicht die englischen Bezeichnungen, die wir letztlich brauchen.
Deswegen müssen wir *Information* bemühen, um über ihre Fenster die letztlich verwertbaren *Pfad*-Angaben zu finden.



Also über den Pfad
`/Applications/LilyPond/.app/Contents/MacOS/LilyPond`
zum LilyPond-Programm und über
`/Applications/LilyPond/.app/Contents/Resources/bin/lilypond`
zum `lilypond`-Befehl.

Unix-Kommandos

In dem in der *LilyPond*-Dokumentation gefundenen Kommando
`lilypond --output=FOLDER filename.ly`
müssen wir also `lilypond` durch seinen *Pfad*
`/Applications/LilyPond.app/Contents/Resources/bin`
ersetzen.

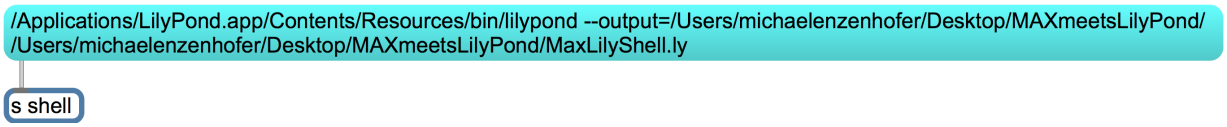
Bei der *Option*
`--output=FILE or FOLDER`
ersetzen wir die Zuweisung
`FOLDER`
durch den *Pfad* des *Folders*, den wir schon mit unserem *shell*-OBJECT erzeugt haben.
In diesen Folder wollen wir auch den resultierenden pdf-*File* befördern.
Den *Pfad* wissen wir, weil wir ihn mit dem Befehl `mkdir` und der *Pfad*-Angabe in MAX erzeugt haben. Ansonsten könnte er analog durch *Anwählen* des Ordners und über *Information* erfahren werden.
Auf meinem Rechner ist das demnach:
`/Users/michaelenzenhofer/Desktop/MAXmeetsLilyPond`

Nun brauchen wir noch den *Pfad* zu dem ".ly"-*File*, welchen wir in einen pdf-*File* umwandeln wollen:
Der in MAX generierte `MaxLilyShell.ly`-*File* befindet sich bereits im `MAXmeetsLilyPond`-*Folder* und hat somit den absoluten *Pfad*:
`/Users/michaelenzenhofer/Desktop/MAXmeetsLilyPond/MaxLilyShell.ly`

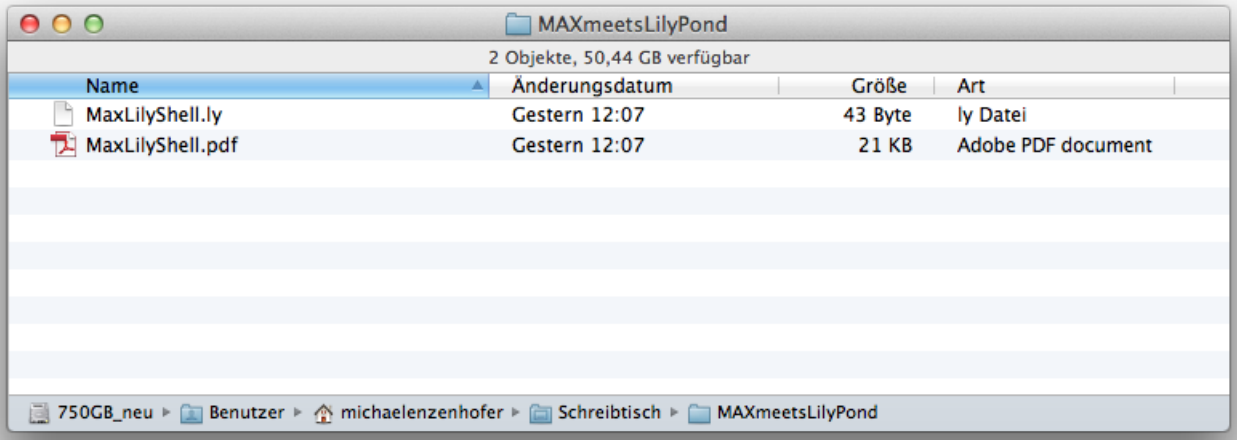
Folglich ersetzen wir den Befehl
`lilypond --output=FOLDER filename.ly`
durch:
`/Applications/LilyPond.app/Contents/Resources/bin`
`--output=/Users/michaelenzenhofer/Desktop/MAXmeetsLilyPond`
`/Users/michaelenzenhofer/Desktop/MAXmeetsLilyPond/MaxLilyShell.ly`

Die Zeilenumbrüche sind hier aus Platzgründen notwendig. Der Befehl ist in einem Zuge mit den *Leerzeichen* – hier genau beim Zeilenumbruch – einzufügen.

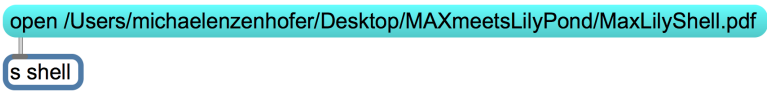
Wieder in MAX sieht der Befehl, den wir zum *shell*-OBJECT senden, so aus:



Dieser erzeugt in unserem Ordner aus dem ".ly"-*File* einen gleichnamigen pdf-*File*:



Von hier kann dieser mit einem `open`-Befehl abgeholt, geöffnet und betrachtet werden.

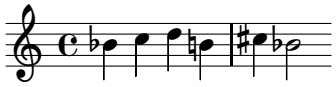


Er zeigt sich genau wie folgend:
w. z. b. w.[1]

[1] **w**as **z**u **b**eweisen **w**ar

Die beinah kleinste Partitur der Welt

Michael Enzenhofer



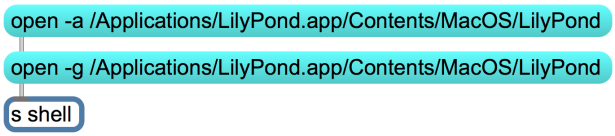
Music engraving by LilyPond 2.16.2—www.lilypond.org

Ergänzend möchte ich noch anführen, dass der `open`-Befehl noch nützlich sein könnte, um ein Programm zu starten oder um ein Dokument mit einem Programm zu starten.

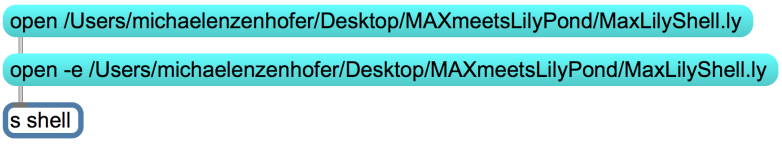
In unserem Fall eventuell LilyPond selbst oder den ".ly"-*File* mit LilyPond.

Allgemein gilt der `open`-Befehl mit dem *Pfad*.
Mit der *Option* `-a` wird das Programm geöffnet und in den *Vordergrund* geholt.
Mit *Option* `-g` wird das Programm geöffnet, bleibt jedoch *Hintergrund*.

Im folgenden Beispiel führt die *Pfad*-Angabe zum LilyPond-Programm (!) und öffnet dieses.



Ein ".ly" File kann theoretisch mit `open` auch ohne *Pfad*-Angabe

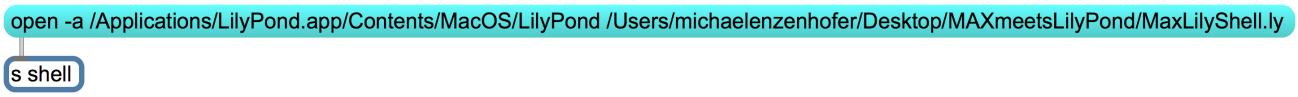


geöffnet werden, es muss jedoch der *File* so eingestellt sein (im *Information*-Fenster – bei "Öffnen mit:"), dass er sich auch mit LilyPond öffnen würde. Das muss nicht unbedingt die Voreinstellung sein, weil es sich im Grunde um einen gewöhnlichen Text-*File* handelt.

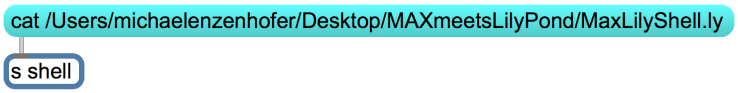
`open` mit der *Option* `-e` öffnet den angegebenen *File* mit dem "TextEdit"-Programm (bei OSX) und dies funktioniert erwartungsgemäß auch mit unserem ".ly"-*File*.

Sicherer ist es, nach dem `open`-Befehl vor den zu öffnenden *File-Pfad* – durch

ein Leerzeichen getrennt (!) – den Programm-*Pfad* – von dem Programm welches den *File* öffnen soll – zu stellen:



Der Inhalt eines Text-*Files* – und somit auch der Inhalt unseres ".ly"-*Files* – kann auch mit einem `cat`-Befehl direkt bei der *Standardausgabe* (*stdout*) ausgelesen und beispielsweise im *text*-OBJECT wieder abgeholt werden.



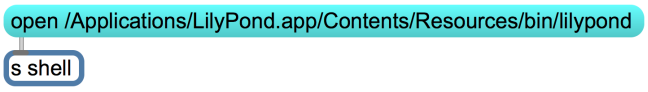
`cat` steht dabei für "con**cat**enate and print files".

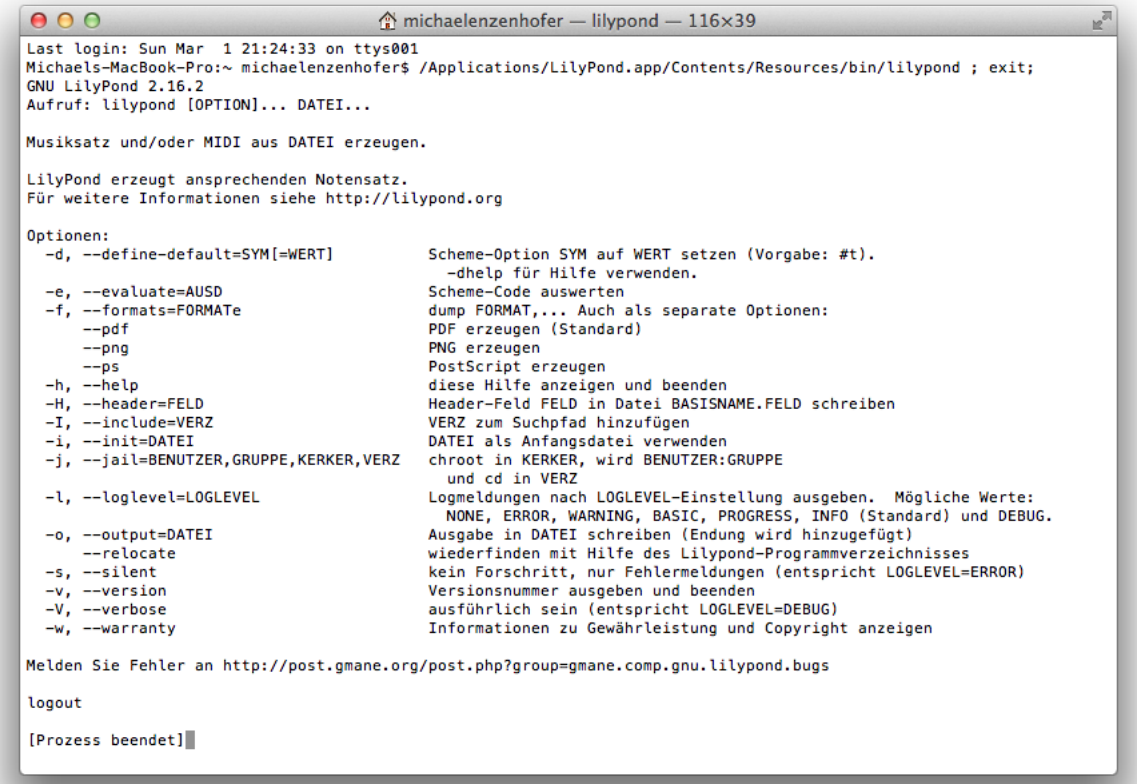
Hier möchte ich noch darauf hinweisen, dass SHELL-*Commands* einen Text-*File* als Ganzes zu manipulieren nicht nur imstande, – sondern prädestiniert sind dazu.
Mögliche Manipulationen wären für unser Vorhaben nicht uninteressant, dazu wäre allerdings mindestens eine weitere *Masterarbeit* notwendig.

Die wesentlichsten „Zauberwörter“ wären:

- `grep`
- `sed`
- `awk`

Ein `open`-Befehl an den `lilypond`-Befehl-*File-Pfad* ;-) gesetzt, öffnet ein *Terminal*-Fenster mit Hinweisen zu den *Optionen* des `lilypond`-Befehls.





Drucken aus der Shell

Ein Text-*File* lässt sich mit dem *shell*-OBJECT auch automatisiert ausdrucken. Das heißt: Mit unserem „Knopfdruck“ zur Generierung des LILYPOND-".ly"-*Files* mit anschließender Umwandlung in einen pdf-*File*, kann auch gleich (sicherheitshalber ein paar Millisekunden später) – mit oder ohne das Ergebnis zu betrachten – ein Druckauftrag an den Drucker gesendet werden.

Auf diese Weise könnte man „spielend“ mit dem dicksten Musiklexikon der Welt in das "Buch der Rekorde" kommen.
Man könnte beispielsweise sämtliche *Permutationen* von zwölf Tönen notieren.

Das wären

$12! = 479\,001\,600$ Notenzeilen.

Wenn wir auf jeder Seite 12 Zeilen unterbrächten, ergäbe das

$479\,001\,600/12 = 39\,916\,800$ Seiten.

Das Ganze platzsparend mit zwei "Seiten pro Blatt" und "Beidseitig" gedruckt ergibt

$9\,979\,200$ Noten-Blätter.

Die Dicke eines 80g-Papiers liegt bei durchschnittlich $0.1mm$.

Das heißt, zehn Blätter sind $1mm$ dick und $9\,979\,200$ Blätter $997\,920\,mm$, das ergibt

$997.920\,m$ Papier.

Ein $1km$ dickes Musik-Buch – davon habe ich schon immer geträumt.

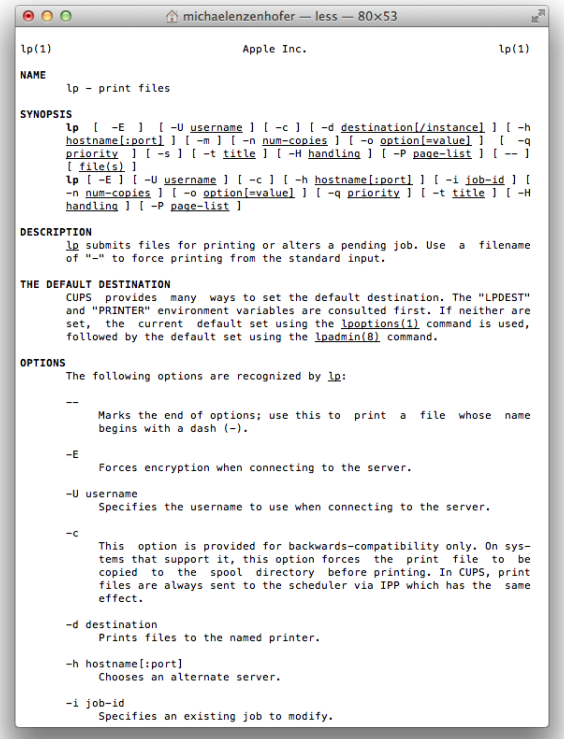
Zur Veranschaulichung, wie unglaublich grenzenlos allein die melodische Vielfalt in der Musik ist, würde mich das Experiment interessieren. Hoffentlich kommt niemand auf die Idee, es umzusetzen. Theoretisch müsste es funktionieren – und wenn man ein paar von mir vorexerzierte Prozeduren einschlägt – auch auf „Knopfdruck“.

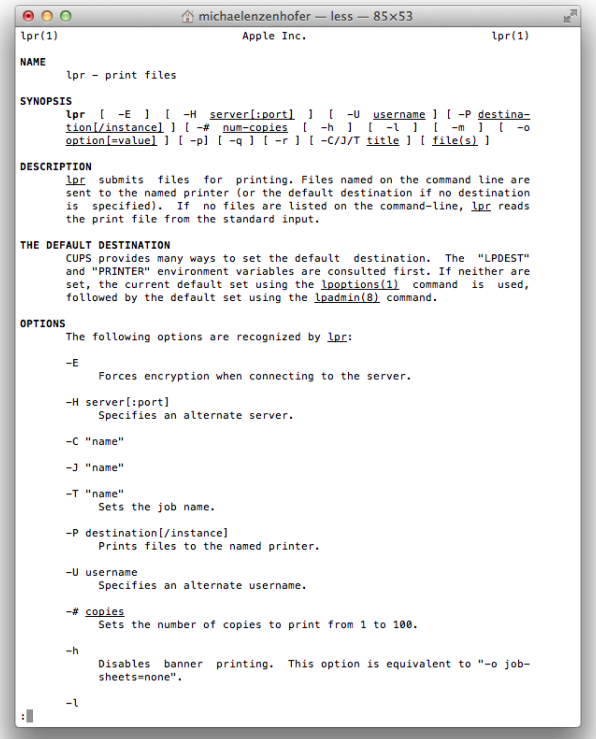
Zum Drucken aus der SHELL bieten sich zwei Befehle gleichermaßen geeignet an.

- `lp`
- `lpr`

Wenn im *Terminal* ein `man`-Befehl (`man` für `man`ual) vor einen Befehlsnamen gesetzt wird, öffnet sich eine sogenannte *man-Page*. Diese gibt Einsicht über die Optionen eines Befehls mit eventuellen Beispielen.

Mit `man lp` öffnet sich das links dargestellte Fenster und mit `man lpr` das rechte.



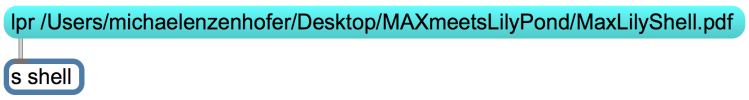


Die *man-Page*s lassen sich noch weiter-*scrollen* und damit die vollständige Übersicht über die *Optionen* einsehen.
Wesentliche *Optionen* zu `lp` möchte ich noch herausheben:

- -d *destination* — «Prints files to the named printer.»
- `-m` — «Sends an email when the job is completed.»
- `-n` *copies* — «Sets the number of copies to print from 1 to 100.»
- `-P` *page-list* — «Specifies which pages to print in the document. The list can contain a list of numbers and ranges . . .»
- `-o` *media=size* — «Sets the page size to size. Most printers support at least the size names "a4", "letter", and "legal".»

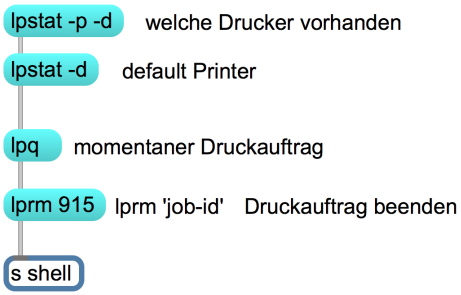
- `-o landscape`
- `-o sides=one-sided`
- `-o sides=two-sided-long-edge`
- `-o sides=two-sided-short-edge`
- `-o fitplot` «Scales the print file to fit on the page.»
- `-o number-up=16` «Prints multiple document pages on each output page.»

Folgender `lpr`-Befehl funktioniert jedenfalls auch wunderbar.



`lpr` mit dem *Pfad* zum Dokument – beziehungsweise unserem pdf-*File* – druckt die Seite.

Mit den Befehlen `lpstat` und den passenden *Optionen* können Drucker auch ausgewählt werden.
Mit `lpq` werden die *Identifikationsnummern* (*id*'s) der Druck-*Jobs* abgefragt und können mit `lprm` und anschließender *id-Nummer* wieder beendet werden.



Mit „einem Wort“:
den Druck-Aufträgen aus MAX mithilfe des *shell*-OBJECTS steht nichts im Wege.

10 Zusammenfassung

Heute fragt man sich, wer der eigentliche Genius einer Komposition ist. Der Komponist oder sein Werkzeug oder etwa der Hersteller des Werkzeugs.

Software wie MAX und LilyPond sind jedenfalls geniale Werkzeuge, die dem Komponisten heute zur Verfügung stehen – MAX zur *algorithmischen Komposition* und LilyPond zur *algorithmischen Notation.*

Die kreative Verbindung von Werkzeugen zeitigt oft – und auch hier – Ergebnisse, die von neuer Qualität bestimmt sind.

MAX als zentrale Musik-Programmierumgebung mit Unix-Kommandos und der Verknüpfung mit LilyPond erlaubt uns, über die „Partitur auf Knopfdruck“ nachdenken.

In vorliegender Arbeit wird ein Überblick über allgemeine Funktionsweisen sowie themenspezifische traditionelle und aktuelle Entwicklungen mit diesen Software-Produkten gegeben.
In der Folge wird anhand *minimaler* Beispiele versucht, Schritt für Schritt einen möglichen Weg zum eigentlichen Ziel zu bahnen.

Meine Entwicklungen – im Besonderen in MAX - werden hier als Download zur freien Verfügung gestellt.
http://michael-enzenhofer.jimdo.com/resultate-ergebnisse/partitur-auf-knopfdruck/
(März 2015).

Ein schöner *Ausblick* wäre, wenn nun viele Freunde der *Algorithmischen Komposition* meinen vorgezeichneten Weg einschlagen würden. Einerseits, um ihn weiter zu entwickeln, andererseits, um ihre gewonnenen Erkenntnisse wiederum interessierten Komponisten zur Verfügung zu stellen.

Meine Hoffnung ist, dass der allgemeine Stellenwert der *computergestützten algorithmischen Komposition* in Zukunft deutlich höher wird, als bisher, und dass *Algorithmen* entwickelt und der Musik-Welt zur Verfügung gestellt werden.

Komponisten sollten nicht verheimlichen müssen, dass in ihren Kompositionen Mathematik im „Spiel“ ist und welche *Algorithmen* sie zur Realisierung einsetzen.

Der „Zauber“, der ihren Kompositionen innewohnt, wird durch den offenen Umgang damit nicht verloren gehen, sondern dadurch verstärkt werden –
so meine Überzeugung.

Bibliografie

BERNDT, Tobias:
LaTeX *— Der typographische Einstieg.*
Addison Wesley (München 2008).

BRAUN, Hans-Joachim:
Komponieren und Konstruieren lehren.
In: **LÜTGE**, Christoph und **MEYER**, Torsten L.(Hrsg.):
Musik – Technik – Philosophie.
Verlag Karl Alber (Freiburg/München 2005) S. 112–114.

COPE, David:
The Algorithmic Composer.
A-R Editions (Madison/Wisconsin 2000).

GRAHAM, Paul:
On Lisp *— Advanced Techniques for Common Lisp.*
Prentice Hall (New Jersey 1994).

HINDEMITH, Paul:
Unterweisung im Tonsatz *— Theoretischer Teil.*
Schott (Mainz 1940).

HEER, Andreas:
Panther für Profis.
Midas (Zürich 2004).

HERNDLER, Florian und **NEUNER**, Florian:
Der unfassbare Klang — *Notationskonzepte heute.*
Klever (Wien 2014).

HOFSTADTER, Douglas:
Gödel Escher Bach — *ein Endloses Geflochtenes Band.*
dtv/Klett-Cotta (München 1991).

MACCORMICK, John:
9 Algorithms that changed the future.
Princeton University Press (Princeton 2012).

MAHNKOPF, Claus-Steffen:
Technik und moderne Musik.
In: **LÜTGE**, Christoph und **MEYER**, Torsten L.(Hrsg.):
Musik – Technik – Philosophie.
Verlag Karl Alber (Freiburg/München 2005) S. 147–166.

NIERHAUS, Gerhard:
Algorithmic Composition — *Paradigms of Automated Music Generation.*
SpringerWienNewYork (Wien 2009).

RAUHE, Hermann und **FLENDER**, Reinhard:
Schlüssel zur Musik.
Knaur (Ulm 1990).

SCHNEBEL, Dieter:
Techno-logische Probleme zeitgenössischen Komponierens.
In: **LÜTGE**, Christoph und **MEYER**, Torsten L.(Hrsg.):
Musik – Technik – Philosophie.
Verlag Karl Alber (Freiburg/München 2005) S. 184–193.

SCHILLINGER, Joseph:
The Schillinger System of Musical Composition.
Carl Fischer (New York 1946).

SCHILLINGER, Joseph:
The Mathematical Basis of the Art.
Philosophical Library (New York 1948).

Weblinks

ftp://arts.ucsc.edu/pub/ems/Lobjects (Dezember 2014).
http://abcnotation.com (Februar(2015).
http://www.algorithmic.net (August 2014).
http://www.all-day-breakfast.com/cannam/linux-musician/lilypond.html
(Februar 2015).

http://www.bachproject.net (Jänner 2015).
http://blog.zdf.de/hyperland/2012/10/kreative-maschinen-wie-algorithmen-komponieren/
(August 2014).

https://ccrma.stanford.edu/software/cmn/cmn/cmn.html (Februar 2015).
http://www.computermusicnotation.com (Jänner 2015).
http://cps.bonneville.nl (Jänner 2014).
https://cycling74.com (Dezember 2014).

http://www.danieleghisi.com/biography/ (Jänner 2015).
http://www.didkovsky.com (Jänner 2015).

https://www.finalemusic.com (Jänner 2015).
http://frescobaldi.org(Februar 2015).

http://georghajdu.de (Jänner 2015).

http://hanwen.home.xs4all.nl (Februar 2015).

http://icking-music-archive.org (Februar 2015).
http://www.ic.unicamp.br/~meidanis/courses/mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (Jänner 2015).
http://jmax.sourceforge.net (Jänner 2015).
http://joyofsource.com/about.en.html (Februar 2015).
http://jtra.cz/stuff/lisp/sclr/index.html (Jänner 2015).

http://letoverlambda.com (Jänner 2015).
http://lilypond.org (Februar 2015).
http://lilypond.org/doc/v2.18/Documentation/notation.pdf (Februar 2015).
http://lilypond.org/doc/v2.18/Documentation/snippets.pdf (Februar 2015).
http://lilypond.org/doc/v2.18/Documentation/usage.pdf (Februar 2015).
http://www.lilypond.org/doc/v2.18/Documentation/usage/command_002dline-usage (Februar 2015).
http://www.lispworks.com/documentation/HyperSpec/Front/Contents.htm (Jänner 2015).

http://michael-enzenhofer.jimdo.com/resultate-ergebnisse/partitur-auf-knopfdruck/ (März 2015).
http://music.columbia.edu/~brad/maxlispj/ (Jänner 2015).
http://www.mutopiaproject.org (Februar 2015).

http://www.native-instruments.com/de/products/komplete/synths/reaktor-5/ (Jänner 2015).

http://ooolilypond.sourceforge.net (Februar 2015).
http://www.openoffice.org/de/ (Februar 2015).

http://www.plogue.com (Jänner 2015).
http://puredata.info (Jänner 2015).

http://repmus.ircam.fr/openmusic/home (Jänner 2015).

http://www.schemers.org (Februar 2015).
http://science.jkilian.de/salieri/GUIDO/ (Februar 2015).
http://scoremus.com (Februar 2015).
http://www.scribus.net (Februar 2015).
http://www.sibelius.at (Jänner 2015).
http://stackoverflow.com/questions/2087693/how-can-i-get-all-possible-permutations-of-a-list-with-common-lisp

(Jänner 2015).
http://support.apple.com/de-at/ht1528 (Februar 2015).

http://www.ugcs.caltech.edu/~rona/tlisp/tlsamples.html (Jänner 2015).

http://vvvv.org (Jänner 2015).

http://www.weblily.net (Februar 2015).
http://www.welt.de/wissenschaft/article111915874/Die-Frau-die-das-erste-Computerprogramm-schrieb.html (August 2014).
http://de.wikipedia.org/wiki/Algorithmus (August 2014).
http://de.wikipedia.org/wiki/Algorithmische_Komposition (August 2014).
http://de.wikipedia.org/wiki/Everything_is_a_file (Februar 2015).
http://de.wikipedia.org/wiki/IRCAM (Dezember 2014).
http://de.wikipedia.org/wiki/LilyPond (Februar 2015).
http://de.wikipedia.org/wiki/Max/MSP (Dezember 2014).
http://de.wikipedia.org/wiki/Portal:Linux (Februar 2015).
http://de.wikipedia.org/wiki/Unix (Februar 2015).

http://www01.zkm.de/algorithmische-revolution/ (März 2015).

Hiermit erkläre ich eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe. Alle Stellen oder Passagen der vorliegenden Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für die Reproduktion von Noten, grafische Darstellungen und andere analoge oder digitale Materialien.
Ich räume der Anton Bruckner Privatuniversität das Recht ein, ein von mir verfasstes Abstract meiner Arbeit auf der Homepage der ABPU zur Einsichtnahme zur Verfügung zu stellen.