# SCORE OBJECTS IN OM#

**Jean Bresson**
Ableton, Berlin
STMS lab: Ircam, CNRS, Sorbonne Université, Paris[1]
jean.bresson@ircam.fr

## ABSTRACT

This paper is an overview of the new score objects and editors available in the OM# visual programming and computer-assisted composition environment.

## 1. INTRODUCTION

OM# [1] [2] is a visual programming and computer-assisted music composition environment derived from OpenMusic [2]. As a computer-assisted composition environment, its main purpose is to provide composers with programming tools allowing them to implement models for the generation, transformation, representation or synthesis of musical material. The visual language is a comprehensive and general-purpose graphical interface on Common Lisp, using the "patching" metaphor to assemble function calls and data structures in functional graphs.

The possibility to manipulate and visualize musical data structures within visual programs, using music notation in particular, is a key element to make such environment an effective compositional framework. Musical data containers and editors can be used as input, output, and for the storage, display and manipulation of intermediate material and results in compositional processes. They enable a specific workflow that contributes to set computer-assisted composition (CAC) beyond so-called "algorithmic composition" systems [3].

OM# was inially developed in the context of research projects aiming at extending the possibilities of OpenMusic in the domains of interaction, time structures and sound spatialization [4, 5, 6]. The early prototypes of the visual language – as presented for instance in [1] – did not yet include any support for scores and "traditional" music notation. We are now a few years later, and a fairly complete score object framework is available (see Figure 1). [3] This paper gives an overview of this framework.

[1] This work was partially carried out while the author was at Ircam STMS laboratory.

[2] https://cac-t-u-s.github.io/om-sharp/

[3] The project was also renamed OM# in the meantime since these early prototypes.

*Copyright: ©2021 Jean Bresson. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.*

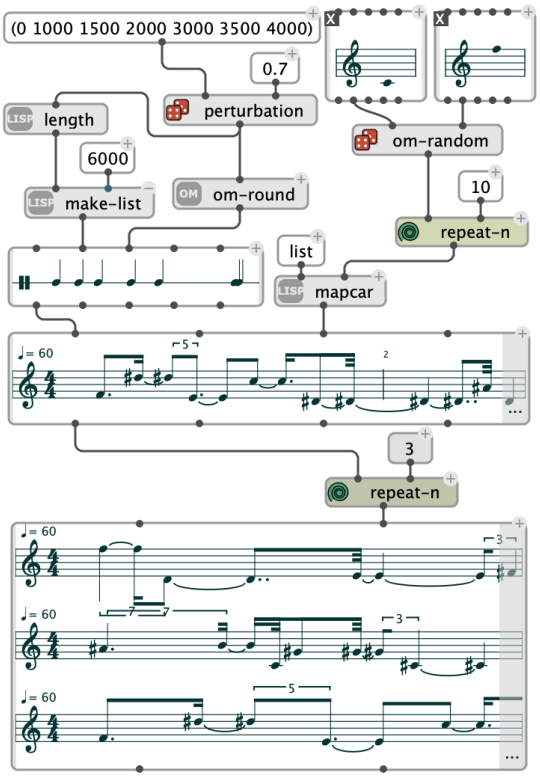


**Figure 1**. Score objects in an OM# visual program.

## 2. SCORE OBJECTS – BASICS

The updated score object framework in OM# inherits the core functionality from OpenMusic/Patchwork [7], and is structured hierarchically as follows:

A NOTE is defined by a pitch, a velocity and a duration, as well as complementary information about MIDI channel and port numbers.

A CHORD is a set of one or several NOTEs.

CHORD-SEQs and VOICEs are sequences of CHORDs, where time-positions are represented respectively as absolute onsets or as rhythmic proportions (see Section 3).

MULTI-SEQ and POLY are polyphonic objects which contain superimposed CHORD-SEQs or VOICEs, respectively.

Figure 2 shows an overview of these different objects. [4]

[4] From the user interface point of view, CHORD, CHORD-SEQ and VOICE objects' read/write accessors gather pitches, velocities, durations, etc. as separate lists of values, offering an orthogonal approach to this underlying hierarchical structure.

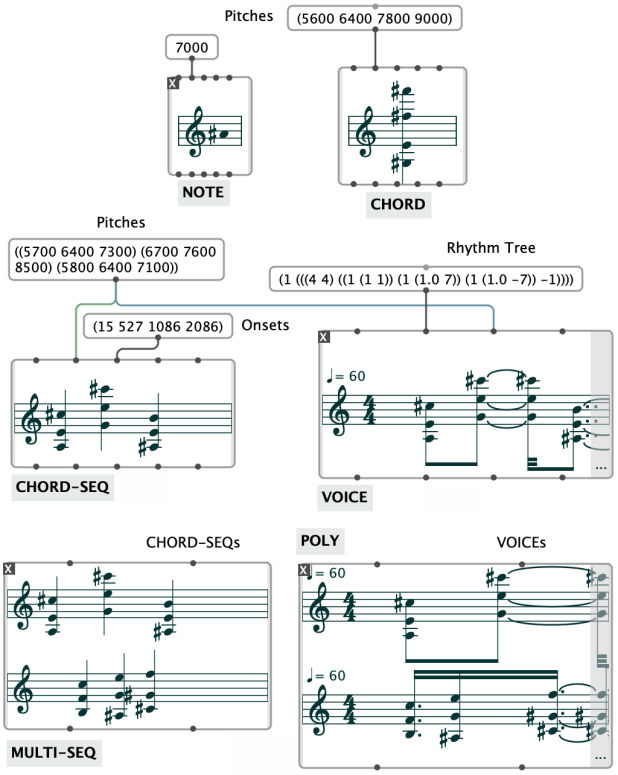


**Figure 2**. OM# main score objects.

## 3. RHYTHMIC STRUCTURES AND NOTATION

Each score object has an explicit or implicit *onset*, determining its time positioning relative to its container object. CHORDs are the actual basic element (also called "timed-item") in OM#'s time representation framework [8].

In order to give account for rhythmic notation, the VOICE object represents time structures using an additional layer of MEASUREs and GROUPs, overlaid on top of the sequence of chords (a GROUP contains nested sub-GROUPs, CHORDs, or RESTs).

The temporal/horizontal spacing algorithm handles graphical alignment between simultaneous events and symbols in polyphonic scores, taking into account the constraints of rhythmic structures (where horizontal space is not necessarily proportional to durations), beaming, artificial spacing introduced by bars, keys, alterations and other symbols. It offers a few different scaling options, and an alternative "proportional" representation of rhythmic structures (i.e. spacing proportionally to the actual duration of musical events) (see Figure 3).

From the user perspective, the rhythmic structure is expressed using a Rhythmic Tree (RT): a recursively nested (*duration* (*subdivisions*)) pair (where each element in *subdivisions* is another RT) representing relative durations and grouping in a compact textual format [9]. Rests are encoded by negative durations, and tied notes by float values. The RT is then internally converted to onsets of the chord sequence (considering a given tempo), and determines the rhythmic layer of measures and nested GROUPs, displayed with adequate beaming, tuplets, note heads and dots depending on the given metrics.

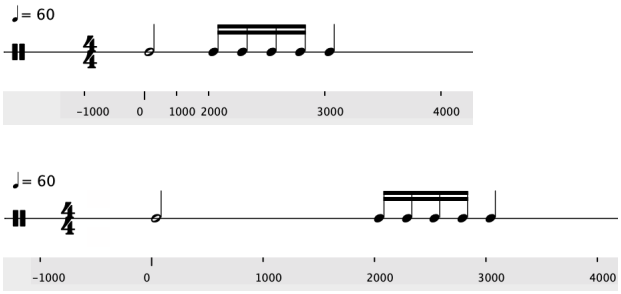


**Figure 3**. Options for time-spacing of a VOICE: rhythmic (top) vs. proportional (bottom).

OM# supports *grace notes*: a subdivision of 0 in a rhythmic tree group is interpreted as a note shifted by a small offset before or after the closest non-null subdivision in that group (depending on the relative position in the list, and on possible other grace notes aligned in a sequence before or after the "main" note of the group) – see Figure 4. Grace notes are displayed and editable in CHORD, VOICE and POLY editors.

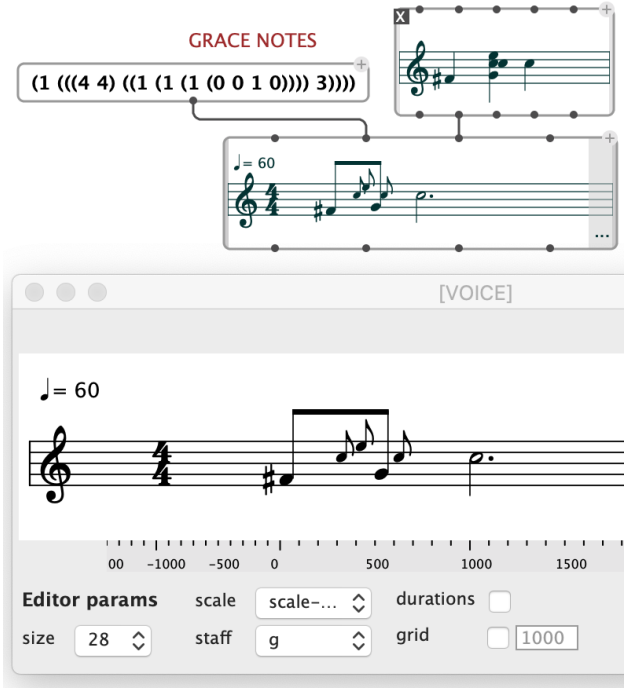


**Figure 4**. Grace notes: Distribute chord pitches before and after the beat. The group of subdivisions (0 0 1 0) in the rhythm tree indicates that the first two notes are offset before, and the last one after the beat position of the chord.

## 4. DISPLAY AND EDITING

OM# score rendering follows the SMuFL (Standard Music Font Layout) specification [5] [6] [10], and so theoretically can adapt to any SMuFL-compliant musical font. [7]

[5] https://www.smufl.org/
[6] https://w3c.github.io/smufl/gitbook/
[7] OM# currently uses the *Bravura* font.

Smooth and continuous zooming in/out gesture and rendering is allowed by a precise positioning of musical glyphs on the staves following this layout specification.

Various options for displaying musical parameters like duration, velocity, MIDI channel and port using musical symbols, numeric values, note heads, color and opacity allow complementing the basic score information (see Figure 5). Picking and editing these parameters is enabled at the level of the NOTE, CHORD, and VOICEs.

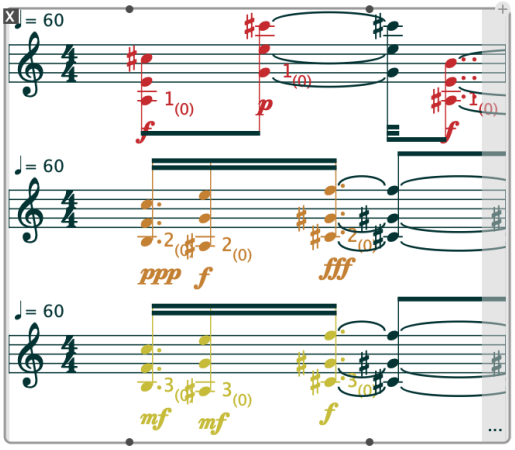


**Figure 5**. Coloured and extended score display including velocities and MIDI ports/channels.

## 5. EXTRAS

EXTRA objects are also inherited from the OpenMusic score framework: they represent additional elements that can be added to the score, although not used for actual (MIDI) rendering. The EXTRA objects currently available include texts, symbols (glyphs from the musical font), altered note heads (e.g. square heads, etc. – also with any glyph from the font), velocity symbols (marking the velocity for some specific chords or notes in the score), as well as labelled markers for score segmentation and annotation (see Section 6). They can be attached to any element of the score objects' internal hierarchy (NOTEs, CHORDs, GROUPs, etc.)

In OM#, EXTRA objects can be set and manipulated as lists (or lists of lists, etc.) directly through optional inputs of the score object boxes (see Figure 6).

## 6. GROUPING AND SEGMENTATION

The *time markers* in score editors can be used for extracting, processing, reordering or applying arbitrary functions to delimited score segments, implementing a simple and versatile version of the "segmentation framework" introduced in [11]. Figure 7 illustrates an application of the *map-segments* function used along with *omquantify* to perform the piecewise rhythmic quantification [8] of a segmented CHORD-SEQ object.

[8] Conversion to VOICE by a translation of sequences of durations into Rhytmic Trees.

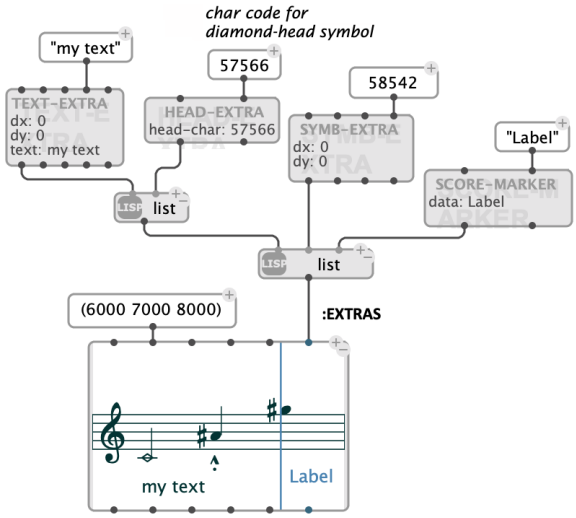


**Figure 6**. EXTRAs: setting additional score components. The first chord has an attached text and special note-heads, the second chord has another attached symbol, and the third chord has a labelled marker.

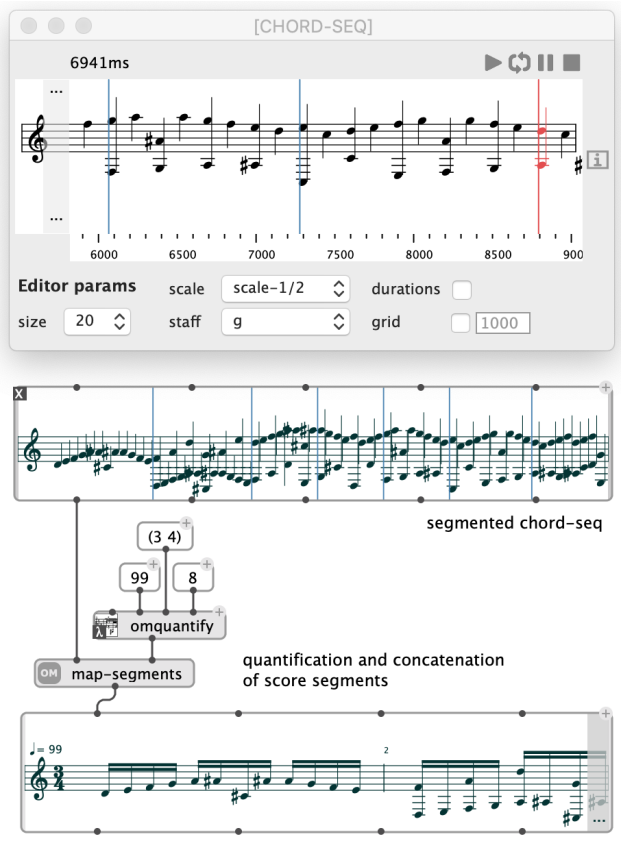


**Figure 7**. Segmentation and piecewise rhythmic quantification using score markers and *map-segments*. Markers can be added algorithmoically as "score extras" or manually in the editor. Score segments delimited with markers are quantified one by one, and the results concatenated in *map-segments*.

The score editor features additional utilities for grouping score elements: selected groups identified by unique IDs can be displayed or processed either internally (in the edi-

tor) or externally (in visual programs). They can also constitute the basis for score analysis models: a basic pitch-class set analysis comes inbuilt in the chord-seq editor as a simple example, using the N-CERCLE object representation [12] for selected pitch sets (see Figure 8). [9]

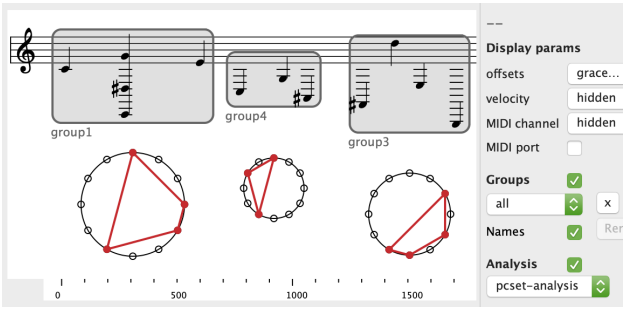


**Figure 8**. Grouping and analysis inside the CHORD-SEQ editor: pitch-class set analysis.

## 7. PROGRAMMABLE EDITOR PARAMETERS

OM# score object boxes include optional inputs (and outputs) making it possible to set (and respectively, to read) attributes which do not belong to the contained / generated score object, but to the box and editor, and determine how the score will be rendered in it. Such attributes include the staves (*G*, *F*, *GF*, etc.), the scale (diatonic, 1/4th tones, etc.), or the musical font size.

Figure 9 shows these parameters manipulated in an OM# visual program.

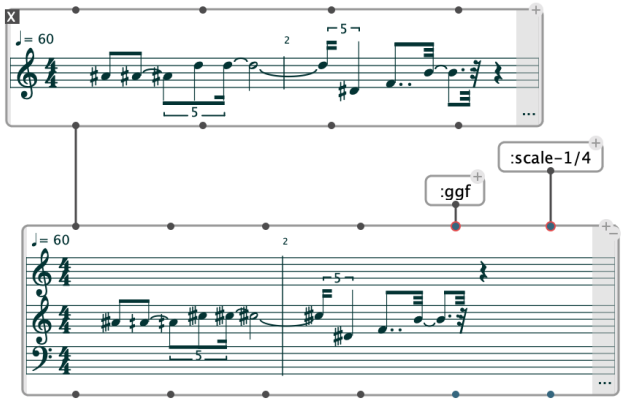


**Figure 9**. Setting score display parameters in OM# visual programs: staff configuration and (quarter-tone) scale.

## 8. SCORES AS REACTIVE INTERFACES

OM# features an embedded "reactive" extension, enabling programs to run and update the score and other data containers and editors as a response to user interaction and external incoming events [13].

Score objects in this context, in addition to dynamically displaying updated states and result of visual programs, can act as interactive controllers, propagating user inputs in downstream data processing.

The NOTE object box is implemented as a *slider* UI box (also called *Interface Box*) so that clicking and dragging on it dynamically updates the pitch of the stored NOTE value (according to the mouse position on the staves), as well as any downstream-connected parts of the visual programs, if adequate box connections are set reactive (see Figure 10).

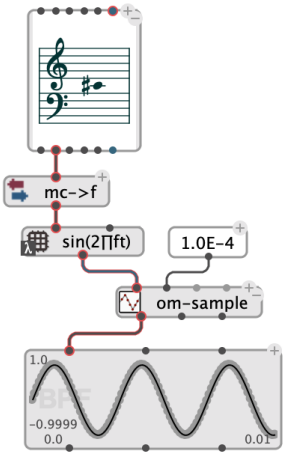


**Figure 10**. Using the NOTE box as input controller (*slider*) to interactively parametrize and fine-tune a visual program (here, a sine-wave generation algorithm). Reactive inlets/outlets and patch-cords are highlighted in red.

The interaction with score objects enabled with this reactive model is close to the one experienced by users in Max [14] (in particluar using the *bach* framework [15]), with the main differences that: (1) Reactive data-flow is simulated – while evens and notifications are "pushed" in the data-flow graph, the execution implements a pull-based model; and (2) Graphs are only *locally* reactive (where connections between boxes are explicitly set so), which allows the user to control the computation load and frequency in response to changes, and to mix reactive data flow with pull-based evaluation of the visual programs.

## 9. CONCLUSION

The score representation and editing features presented in this paper contribute to make of OM# an operational and effective framework for computer-assisted music composition today.

The more tangible improvements of this framework are at the level of display and interaction with score elements, facilitated by a renewed rendering framework. The inclusion of editor parameters in visual programming is also a new concept permitted in OM#, and the representation of grace notes is another notable increment.

However, some of these features are still incomplete and constitute the object of future work: at the time of this writing, rhythmic structure editing is still limited (as compared to OpenMusic, for instance), as well as support for tempo changes and variations or micro-tonality beyond 1/4 and 1/8th tones.

[9] Grouping and segmentation features presented in this section are available since OM# v1.3.

## 10. REFERENCES


[1] J. Bresson, D. Bouche, T. Carpentier, D. Schwarz, and J. Garcia, "Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic," in *Proceedings of the International Computer Music Conference (ICMC)*, Shanghai, China, 2017.

[2] J. Bresson, C. Agon, and G. Assayag, "OpenMusic. Visual Programming Environment for Music Composition, Analysis and Research," in *ACM MultiMedia'11 (OpenSource Software Competition)*, Scottsdale, USA, 2011.

[3] G. Assayag, "Computer Assisted Composition Today," in *1st Symposium on Music and Computers*, Corfu, Greece, 1998.

[4] D. Bouche, J. Nika, A. Chechile, and J. Bresson, "Computer-aided Composition of Musical Processes," *Journal of New Music Research*, vol. 46, no. 1, 2017.

[5] J. Bresson, J. MacCallum, and A. Freed, "o.OM: Structured-Functional Communication between Computer Music Systems using OSC and Odot," in *FARM: Workshop on Functional Art, Music, Modeling & Design – Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP)*, Nara, Japan, 2016.

[6] J. Garcia, T. Carpentier, and J. Bresson, "Interactive-Compositional Authoring of Sound Spatialization," *Journal of New Music Research*, vol. 46, no. 1, 2017.

[7] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue, "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic," *Computer Music Journal*, vol. 23, no. 3, 1999.

[8] J. Garcia, D. Bouche, and J. Bresson, "Timed Sequences: A Framework for Computer-Aided Composition with Temporal Structures," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*, A Coruña, Spain, 2017.

[9] C. Agon, K. Haddad, and G. Assayag, "Representation and Rendering of Rhythmic Structures," in *Second International Conference on Web Delivering of Music*, Darmstadt, Germany, 2002.

[10] D. Spreadbury and R. Piéchaud, "Standard Music Font Layout (SMuFL)," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR)*, Paris, France, 2015.

[11] J. Bresson and C. Pérez-Sancho, "New Framework for Score Segmentation and Analysis in OpenMusic," in *Proceedings of the Sound and Music Computing conference (SMC)*, Copenhagen, Denmark, 2012.

[12] M. Andreatta and C. Agon, "Implementing Algebraic Methods in OpenMusic," in *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

[13] J. Bresson and J.-L. Giavitto, "A Reactive Extension of the OpenMusic Visual Programming Language," *Journal of Visual Languages and Computing*, vol. 25, no. 4, 2014.

[14] M. Puckette, "Combining Event and Signal Processing in the MAX Graphical Programming Environment," *Computer Music Journal*, vol. 15, no. 3, 1991.

[15] A. Agostini and D. Ghisi, "A Max Library for Musical Notation and Computer-Aided Composition," *Computer Music Journal*, vol. 39, no. 2, 2015.