RECENT DEVELOPMENTS IN ENP-SCORE-NOTATION

Mika Kuuskankare
DocMus
Sibelius Academy
mkuuskan@siba.fi

Mikael Laurson
CMT
Sibelius Academy
laurson@siba.fi

ABSTRACT

In this paper we describe the recent developments on a text-based score representation format called ENP-score-notation. ENP-score-notation is used to describe scores in Expressive Notation Package (ENP). We introduce the basic concepts behind ENP-score-notation and discuss several new features in detail along with some notational examples. We also examine some syntactic changes and point out the differences between the new and the old syntax. Furthermore, we propose a new interface to PWGL, including several new boxes, which allows to use ENP-score-notation to construct musical objects in a PWGL patch.

1. BACKGROUND

Many different text-based formats have been developed for representing musical information. Some typical examples are Common Music Notation [10], GUIDO [3], LilyPond [9], and MusicXML [2]. There are also some Lisp-based representations that are able to preserve only a subset of the musical information, such as the rhythm. These include the RTM-notation of both PatchWork [6] and OpenMusic [1].

There are several advantages in text-based representation of a musical score: (1) the layout of notational objects is separated from the representation, (2) the musical typesetting is improved when the typesetting program improves, and (3) textual formats are human readable.

ENP-score-notation [8] is a text-based format to represent ENP scores. ENP [4] is a LISP-based music notation program. It currently runs on MacIntosh OS X platform. The main purpose of ENP is to represent musical data for compositional and music analytical applications. ENP-score-notation is based on the hierarchical score structure of ENP. ENP-score-notation offers a large subset of definable attributes. The syntax is designed so that it also allows to add new features easily. The current subset, however, should be large enough for most practical purposes. The Lisp list format allows a straightforward way to generate and manipulate musical data. ENP-score-notation can also be used in PWGL to represent musical objects. PWGL [7] is a multi-purpose visual programming environment specialized in music and sound related applications. PWGL uses ENP as its notational front end.

The rest of the paper is organized as follows. Firstly, we briefly present the overall data structure of an ENP score. Secondly, we discuss ENP-score-notation in general. Next, we point out some syntactic changes in ENP-score-notation. Finally, we introduce a set of new PWGL boxes, relating to ENP-score-notation, along with some example patches.

2. ENP SCORE STRUCTURE IN BRIEF

ENP scores are internally represented as hierarchical tree structures. For example, a score consists of a list of parts and parts, in turn, consist of list of voices, etc. In general, any higher-level structures are constructed by collecting lower-level structures as lists. Figure 1 gives an overview of the score structure.

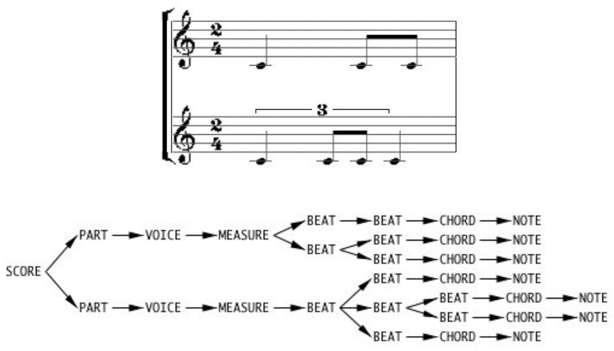


Figure 1. An example of an ENP score (above) and its object hierarchy displayed as a tree structure (below).

On the beat-level ENP score representation is based on the concept of PW-beat which can be represented as a Lisp list. This list, called a `beat-list`, has two main components: `beat-count` and `rtm-list`. `beat-count` is a positive number, usually an integer. `rtm-list`, in turn, contains a list of numbers, called `rtm-values`. The type of an individual `rtm-value` has the following meanings: (1) positive integer represents a note, (2) negative integer represents a rest, and (3) floating-point number indicates a note that is tied to the previous one.

Figure 2 gives some typical examples of the `beat-list` notation along with their notational counterparts. More detailed explanation of PW-beat can be found in [6]. ENP score structure, in turn, is discussed in greater detail in [8].

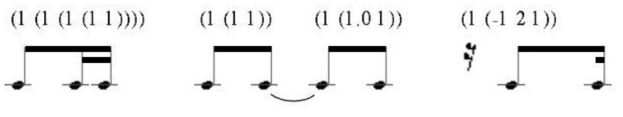


Figure 2. Some simple rhythms demonstrating the use of `rtm-values` in the `beat-list`.

3. ENP-SCORE-NOTATION

When we use a text-based format to describe the complex structural information contained by a musical score there are several things to be considered. The syntax has to be: (1) compact so that we are able to handle large scores efficiently, (2) understandable to a human reader so that it could be used directly as input to build scores, (3) easily translatable into the score representation (and vice versa), and (4) extensible in order to meet future extensions and improvements.

3.1. ENP-score-notation Syntax

The structure of ENP-score-notation [8] reflects the hierarchical structure of the score. The syntax is similar to the LISP list syntax and also resembles CLOS in the way the optional keyword arguments are given. The basic syntax of ENP-score-notation describes only the rhythmic skeleton along with the overall structure of the score (i.e., parts, voices, measures, etc.). All other information (including pitches) is defined as optional keyword arguments. For example, in its simplest form, a triplet is notated as `(1 (1 1 1))`[1]. The pitch information[2] can be added by using a keyword `:notes`, e.g., `(1 ((1 :notes (60 64 67)) (1 :notes (60 64 67)) (1 :notes (60 64 67))))`. The `:notes` keyword refers to an actual slot in an ENP chord object.

There are several advantages in this approach: (1) Since all the attribute names refer to actual slots in CLOS objects, features can be added in most cases without any additional coding. (2) Syntax is uniform and simple: everything else, besides rhythm, is defined by adding optional attributes. (3) Unless a particular attribute is given it is set to some reasonable default. (4) The hierarchical relations of notational objects can be determined by examining the textual representation. (5) The format maps exactly to the score hierarchy and is therefore easily transformable back to an ENP score.

Below we give a simplified but complete example of the structure of ENP-score-notation along with the resulting score (Figure 3):

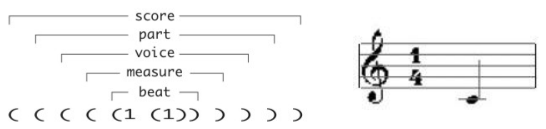


Figure 3. A simplified ENP-score-notation example (left) and the resulting score (right).

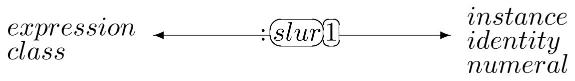
One of the problems with strict hierarchical representation is how to express entities that do not obey the hierarchy. ENP-expressions are a typical group of objects that break the otherwise hierarchical representation described above. In the next subsection we briefly describe how ENP-expressions are represented in ENP-score-notation.

[1]Even if not explicitly indicated the resulting triplet would in this case also contain some pitch information. The default pitch, unless stated otherwise, is c^1 (= middle c).

[2]Pitches in ENP are generally represented as midi values, 60 being c^1, i.e., middle C. However, ENP can also represent microtones. They are notated by adding the relevant decimal fraction to the midi value. Thus, 60.5 would be c^1 raised by 1/4 (half a semitone).

3.2. Syntax of ENP-expressions

Currently ENP-expressions [4, 5] can be attached to either single chords or notes (single-expressions) or to groups of chords or notes (group-expressions). Expressions can be incorporated into ENP-score-notation by using a keyword :expressions followed by a list of expressions and possible initialization options. Expressions are identified by using an appropriate keyword indicating the class of the expression. Among valid expression keywords are, for example, `:group, :slur, :bpf, :accent, :staccato`, etc. In case of group expressions the keywords usually contain an additional numeral that indicates the instance identity of the expression. The instance identity numeral is used to distinguish between different instances of group-expressions (in case of overlapping slurs, for example). Thus, `:slur1` indicates an ENP expression of class 'slur' with the identity value '1'. All subsequent references to `:slur1` point to the same instance. Furthermore, when an expression is introduced for the first time, any number of initialization options can be supplied. This information consists of keyword and value pairs defining any additional properties of the expression. The components of an expression keyword are as follows:



4. CHANGES IN ENP-SCORE-NOTATION SYNTAX

In this Section we enumerate some syntactic changes in ENP-score-notation and also present some new functionalities, such as the possibility to describe non-mensural notation.

4.1. Changes in Beat Types

Formerly, the type of the resulting beat was indicated by inserting an extra symbol as the last element in the beat-list. For example, a grace-beat was created by using the symbol `'grace-beat`, e.g., `(1 (1) 'grace-beat)`. The new syntax, however, unifies the use of these kinds of optional attributes. When using the current version of ENP-score-notation the grace-beat is defined as follows:

(1 (1) :class :grace-beat). Notice the inclusion of a keyword :class. The beat list can also have several other optional attributes. The following values for the :kind keyword are currently supported: :accelerando, :ritardando, and :feathered. The distinction between :class and :kind keywords also allow to mix these attributes, e.g., to create a grace-beat with accelerando beaming.

Figures 4 demonstrates the use of the :kind keyword and Figure 5, in turn, gives an example of a grace-beat created with the keyword :class.

```
(((((1 (1 1 1 1 1 1) :KIND :ACCELERANDO)))))
```

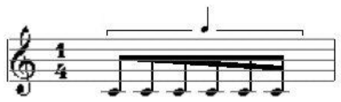


Figure 4. A beat with accelerando beaming.

```
(((((1 ((1 (1 1) :CLASS :GRACE-BEAT) (-1)))))))
```

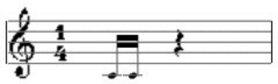


Figure 5. A grace-beat

4.2. Non-mensural notation

The current version of ENP-score-notation allows to represent non-mensural notation. The syntax is otherwise identical to that of mensural-notation, but instead of using proportional beat-counts and rtm-values, non-mensural notation uses absolute start-times and relative offset-times. Thus a mensurally notated chord, (1 ((1 :notes (60 64 67)))), could potentially be translated to non-mensural notation as (0.0 :notes (60 64 67) :duration 1.0), where the first floating point number represents the start-time of the chord.[3] Each note can also have an independent offset-time relative to the start-time of the chord. Next we give a complete example of non-mensural notation and the resulting score (Figure 6):

```
(((((0.2 :NOTES ((60 :OFFSET-TIME -0.15)
                 (66 :OFFSET-TIME 0.0)
                 (71 :OFFSET-TIME 0.15))
         :EXPRESSIONS ((:crescendo/1
         :initial-dynamics :p :end-dynamics :f)))
    (0.5 :NOTES (77) :EXPRESSIONS (:crescendo/1))))))
```

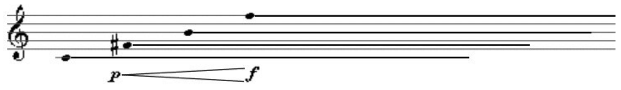


Figure 6. A non-mensural score described with ENP-score-notation.

[3]We assume that, in the mensural case, the time-signature is 4/4 and tempo is 60 beats/minute. Thus, the duration of one quarter-note is 1 second.

4.3. Changes in ENP-expression Syntax

When using the old ENP-score-notation syntax the instance identity numeral is attached directly to the end of the expression keyword, i.e., :bpf1. This creates a potential conflict when used with expressions whose names contain a numeral, such as the string number expressions (:string1, :string2, etc.). The new syntax allows to separate the expression name and the instance identity numeral more precisely by using a slash (/). Now different instances of, for example, string number expressions can be denoted as :string1/1 and :string1/2. This approach clarifies ENP-score-notation syntax. The old syntax, however, is still valid and can be used alongside the new one.

4.4. Score-BPF notation

The Score-BPF [5] is multipurpose graphical object that can represent breakpoint functions as a part of a musical texture. breakpoint functions, in turn, are piece-wise linear functions. It can contain an arbitrary number of breakpoint functions which, in turn, contain a list of points with x and y values. The x values are timing values, in seconds, relative to the start-time of a Score-BPF in the score.

The new version of ENP-score-notation supports Score-BPF expressions. Currently there are two ways to define a point in a breakpoint function:

(1) By using absolute values. In this case a point can simply be indicated by a list of two numbers, e.g., (0.0 1.0). The first number in the list defines the position of the point in time relative to the start-time of the Score-BPF. The second value is a y value that should fall inside the user definable height of the Score-BPF.

(2) By using values that are proportional to the length (in time) and height of the Score-BPF. The point is written as a list of two numbers ranging form 0.0 to 1.0. In this case it is required to include an optional keyword argument, :scaled t, indicating that the incoming values are to be scaled inside the Score-BPF (Figure 7).

Moreover, the user can define a number of others attributes, such as type, name and height of the Score-BPF, the name and color of the breakpoint function, etc.

5. PWGL INTERFACE TO ENP-SCORE-NOTATION

In this Section we present the interface between ENP-score-notation and PWGL. Due to its list based representation ENP-score-notation provides a natural way to construct musical structures in a PWGL patch. We introduce next three new boxes that allow the user to generate and manipulate ENP objects in a PWGL patch. Two of the boxes, enp-constructor and

```
(((((1 ((1 :notes (72)
           :expressions ((:bpf1 :points (((0.0 0.5)
                                          (1.0 0.75))
                                 :scaled t
                                 :name "acc.")))))
        (1 :notes (74) :expressions (:bpf1))
        (1 :notes (76) :expressions (:bpf1))
        (1 :notes (77) :expressions (:bpf1))
        (1 :notes (78) :expressions (:bpf1))))))))
```

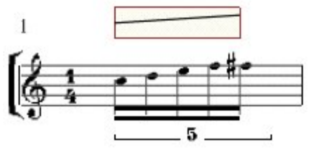


Figure 7. A Score-BPF expression (:bpf1) inserted in the score with the help of ENP-score-notation. Notice also the use of the extra arguments `:name` and `:scaled`.

`enp-object-composer`, are used to convert ENP-score-notation to ENP objects. The third complementary box, `enp-score-notation`, is in turn used to convert an ENP score to ENP-score-notation.

5.1. enp-constructor -box

`enp-constructor` -box is used to create a single kind of ENP object. The main-box consists of two required input-boxes: (1) Selector-input-box which is used to select the type of the resulting object. The box dynamically changes the number and type of required input-boxes as the user changes this value. (2) One or two value-input-boxes that accept as input either ENP-score-notation expression or ENP objects. (Figure 8)

Any additional properties can be defined by extending the `enp-constructor` -box. Extending reveals one special input-box pair at the time. The collection of possible input-box pairs depends on the type of the resulting object. The input-box pairs, in turn, contain two input-boxes arranged in a row. The first input-box, *selector-input-box*, is used to select an attribute keyword. A change in the selector-input-box performs a side effect to the second input-box, called *state-input-box*. The state-input-box dynamically changes its type according to the value of the selector-input-box. If the value of the selector-input-box is, for example, `:start-time`, the state-input-box is of type float-input-box. On the other hand, if the value of the selector-input-box is `:expressions`, the state-input-box is an expression menu-input-box containing all the possible ENP-expressions. Figure 8 shows an `enp-constructor` -box for a beat object. The box is extended to add a class attribute (i.e., :class and :grace-beat). The lowest row of the box shows the corresponding input-box pair.

5.2. enp-object-composer -box

`enp-object-composer` -box can be used to convert any lower level object to any of the higher level objects (see the score hierarchy in Section 2). Thus,

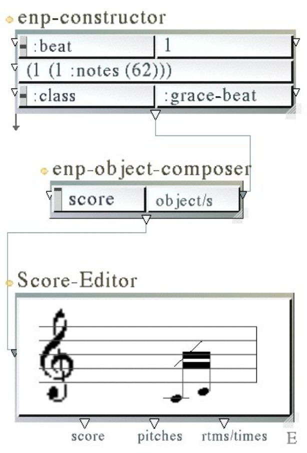


Figure 8. `enp-constructor` -box is used to create a grace-beat.

a note could potentially be converted to a score, etc. `enp-object-composer` -box has several useful properties: (1) it allows fast visualization of any musical object (e.g., convert a note directly to a score), (2) it can handle either single objects or lists of objects, (3) it can be used as a shorthand to create any higher level objects out of any combination of lower level objects (types can be freely mixed), and (4) the input format is similar to the hierarchical score representation and can be used to further modify the structure of the resulting object. Figure 8 shows one application to `enp-object-composer` -box. In this case a grace-beat is converted directly into a score.

Let us next examine a more complex patch given in Figure 9 (see Appendix). The main idea behind this patch is to demonstrate how to create different score structures from the same input data just by adding one extra parenthesis each time. The original input in this case is a list of two notes. In the first case (a) this input creates a score with two parts each containing one note. In (b) one extra parenthesis is added around the two notes. This, in turn, creates a score with one part containing two voices. Each voice again contains one note (the stems of the notes are drawn in different directions). In (c) two extra parenthesis are added around the the notes and the resulting score contains a part with two measures. In the last case (d) there is a total of four parenthesis around the notes. The resulting score contains one part with a measure containing two notes. This procedure can be continued to create even deeper structures.

5.3. enp-score-notation -box

`enp-score-notation` -box is used to convert a score to ENP-score-notation expression. Unless stated otherwise, this box collects all the optional attributes for each object in the score.[4] It is, however, possible to

[4]There is a relating mechanism in ENP called automatic source code generator. The purpose of the automatic source code generator is to build a Lisp representation of a CLOS object. It can then be used to re-create

have control over which attributes are collected. The `enp-score-notation` -box contains a menu-input-box and a dialog-input-box for this purpose (see Figure 10). The menu-input-box can be used to choose between `:exclude` and `:include` keywords. The dialog-input-box, in turn, can be used to select the set of attributes from a dialog containing all the possible attributes. According to the value of the menu-input-box these attributes are then either included in or excluded from the result.

Figure 10, in the Appendix, gives an example of the extreme effects of `:exclude` and `:include`. The topmost score is the starting point containing a short passage of music with some expression markings and special note-heads. The `enp-score-notation` -box on the left (a) has the `:exclude` parameter set to `NIL` meaning that nothing is excluded from the result. This creates an exact copy of the original score (b). The `enp-score-notation` -box on the right (c) has the `:include` parameter set to `NIL`. In this case none of the optional attributes is included. Only the rhythmic skeleton is collected as shown in (d).

6. CONCLUSIONS

This paper presented some recent developments in ENP-score-notation. Several new features were introduced. Also some syntactic changes were proposed that further clarified or extended the functionality of ENP-score-notation. Furthermore, a set of dedicated PWGL boxes were introduced. These boxes provide a bridge between ENP-score-notation and PWGL and allow an algorithmic control over ENP-score-notation.

There is still some room for improvement. The two ways to describe breakpoint functions, presented in this paper, are quite primitive. There is no apparent relation between the points and the notational objects (except for the start and end points). The syntax should be expanded so that it would allow the notational objects to be referenced by a symbol or an index.

The current inclusion functionality of `enp-score-notation` -box is somewhat ambiguous. As explained in the footnote 4 the inclusion of a certain attribute is dependent not only on the choices made by the user but also the underlying decisions made by the system. There should be a mechanism to force certain attributes to be present in the ENP-score-notation expression (e.g., force-include feature). This would ensure that the needed attributes would always be collected. The information could then be used to construct other objects that explicitly rely on the existence of such data. Instead, the current implementation requires that the user has knowledge about the default values of the various attributes.

an exact copy of the object. However, in order to optimize the Lisp representation, the automatic source code generator also compares the properties of the CLOS objects to their default values and determines if the value of the particular property needs to be included. This mechanism is quite sophisticated and can also handle arbitrary deep hierarchical structures.

The same kind of strategy is used in the case of ENP-score-notation. The `enp-score-notation` -box actually collects only those attributes that are different from their default value. For example, let us say that there is a note whose duration is 1.0 seconds. This is also the default duration of a note object. In this case the `:duration` keyword and the corresponding value are not included in the ENP-score-notation expression.

7. ACKNOWLEDGMENTS

The work of Mikael Laurson has been financed by the Academy of Finland.

8. REFERENCES

[1] Gerard Assayag, Camillo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3):59–72, Fall 1999.

[2] M. Good and G. Actor. Using MusicXML for File Interchange. In *Third International Conference on WEB Delivering of Music*, page 153, Los Alamitos, CA, September 2003. IEEE Press.

[3] H. H. Hoos, K. A. Hamel, K. Renz, and J. Kilian. The GUIDO Music Notation Format - A Novel Approach for Adequately Representing Score-level Music. In *Proceedings of International Computer Music Conference*, pages 451–454, San Francisco, 1998.

[4] Mika Kuuskankare and Mikael Laurson. ENP2.0 A Music Notation Program Implemented in Common Lisp and OpenGL. In *Proceedings of International Computer Music Conference*, pages 463–466, Gothenburg, Sweden, September 2002.

[5] Mika Kuuskankare and Mikael Laurson. ENP-Expressions, Score-BPF as a Case Study. In *Proceedings of International Computer Music Conference*, pages 103–106, Singapore, 2003.

[6] Mikael Laurson. *PATCHWORK: A Visual Programming Language and some Musical Applications*. Studia musica no.6, Sibelius Academy, Helsinki, 1996.

[7] Mikael Laurson and Mika Kuuskankare. PWGL: A Novel Visual Language based on Common Lisp, CLOS and OpenGL. In *Proceedings of International Computer Music Conference*, pages 142–145, Gothenburg, Sweden, September 2002.

[8] Mikael Laurson and Mika Kuuskankare. From RTM-notation to ENP-score-notation. In *Journées d'Informatique Musicale*, Montbéliard, France, 2003.

[9] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *XIV Colloquium on Musical Informatics (XIV CIM 2003)*, Firenze, Italy, May 2003.

[10] Bill Schottstaedt. Comon Music Notation. In *Beyond MIDI, The Handbook of Musical Codes*. MIT Press, Cambridge, Massachusetts, 1997.

Appendix

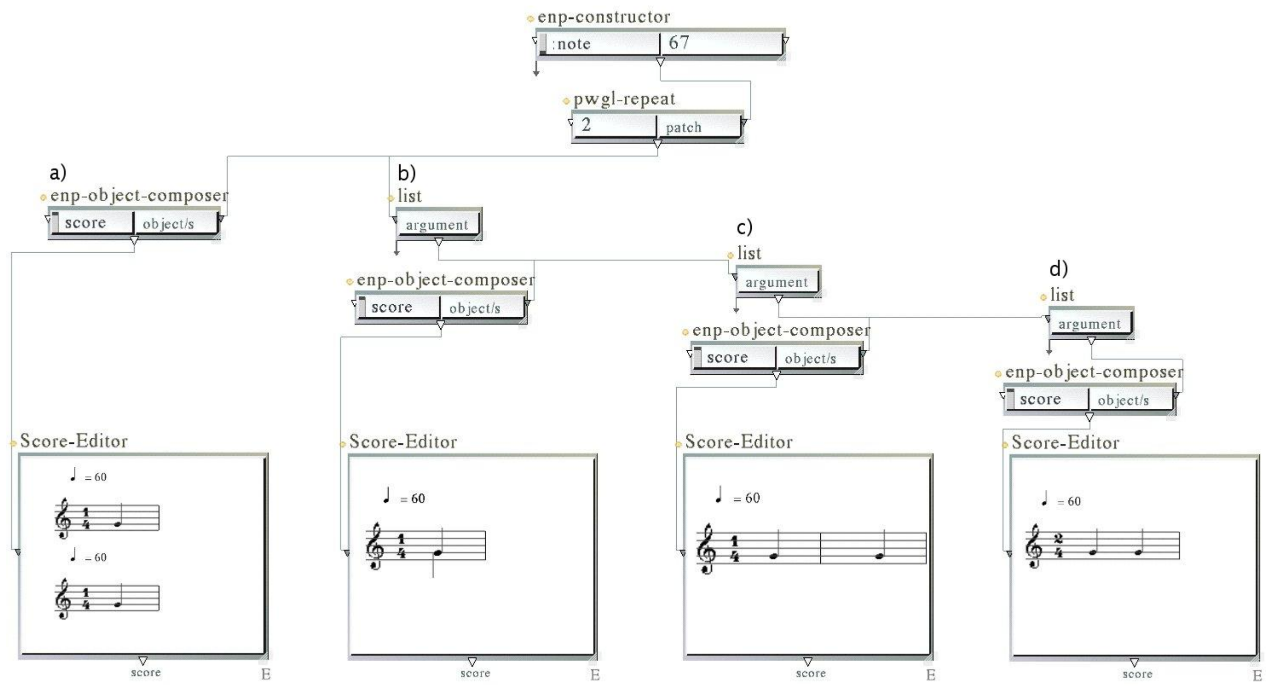


Figure 9. An Example showing the behavior of the `enp-constructor` -box. By manipulating the input data, different kinds of score structures can be created.

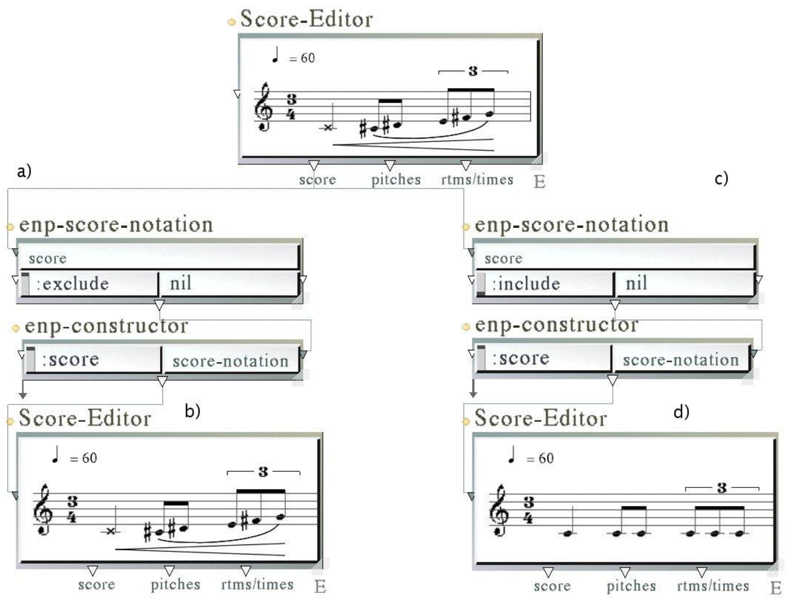


Figure 10. An example showing the two extremes of `enp-score-notation` -box. On the left: an exact copy is made. On the right: only the rhythmic skeleton is retained.