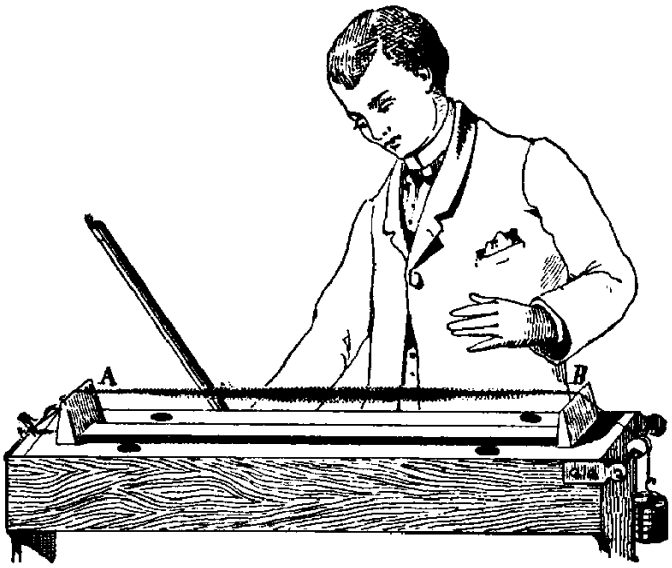
Director Musices User Manual

For Director Musices ver. 3.1.5

Current Release: 2023-04-28



Director Musices User Manual.....*1*

Introduction **3**

Basic operation **4**
Starting the program 4
Loading Music 4
Playing 4
Applying Rules 4

Menu Commands **6**
File menu 6
Edit menu 7
Help menu 8

Window commands **9**
Transport window (top pane) 9
Track window (upper pane) 9
Score window (middle pane) 9
Rule palette window (bottom pane) 10

Performance Rules **11**
Phrasing 11
Harmonic and Melodic Tension 12
Metric patterns and grooves 13
Ensemble Swing 13
Articulation 14
Immanent accents 14
Performed accents 14
Timing 15
Ensemble timing 15
Intonation 15
Piano specific 16
Noise 16
Utility functions 16
References and bibliography 18

Playing Details **20**
Synthesizer Objects 20
Synths available in the DM program track window 20
Recent synths only available in the DM core 21
Old synth definitions only available in DM core 21

Music File Format **22**
Track Variables 23
Input music notation variables 23
Music performance variables 25
Speech notation variables 26
Speech performance variables 26

Writing New Rules **27**
Music Structure 27
Rule top-level definition 27
Serial Rule Macros 27
Serial Access Functions 29
Time shapes 32
Parallel Rule Macros 32
Parallel Access Functions 32

Introduction

Director Musices (DM) is a program that allows you to change the performance of a music score. It contains a set of rules that changes the notes' duration, sound level, articulation, and other parameters. These rules mimic performance principles used by real musicians. The input quantity parameters given to the rules can drastically change a performance according to the intentions of the meta-performer (that is, you). This includes the modeling of different emotional characters such as "happy" or "sad". The rules are resulting from a long-term research project at the Royal Institute of Technology, Stockholm started by Johan Sundberg, Lars Frydén, and Anders Friberg. For further information see our web-site below and the references at the end.

The current implementation is a combination of different languages. It is in an experimental stage since many functions are not working according to general user interface standards. The core rule system is implemented in common lisp and can be run independently in most common lisp environments using a command-line interface. The integrated program GUI including the core and the user interface has been developed in Closure within a java environment by Elin Fischer Friberg.

If you have any questions, comments, or found bugs we would be glad to hear from you and we hope you will enjoy using Director Musices.

Credits

Rule development
Johan Sundberg, Lars Frydén, Anders Friberg, Roberto Bresin. Erica Bisesi, Richard Parncutt, Erwin Schoonderwaldt, Lars Gunnar Bodin, Anders Askenfelt, Patrik Juslin, Torbjörn Gulz, Claes Wettebrandt

Code development
Anders Friberg, Elin Fischer Friberg, Vittorio Columbo, Roberto Bresin

Contact

Anders Friberg
afriberg@kth.se
https://www.kth.se/profile/afriberg

Links

Music performance pages at KTH (the download links, music examples):
http://www.speech.kth.se/music/performance/

Director Musices standalone (3.1.5):
https://github.com/docfry/Director-Musices

Director Musices standalone (3.1.3):
http://odyssomay.github.io/clj-dm/

DM core source code (Common lisp, including recent rules):
https://github.com/docfry/Director-Musices

Director Musices GUI source code (Closure, Common lisp, Java):
https://github.com/odyssomay/clj-dm

Basic operation

Note that the documentation for the DM program GUI is also described partly in http://odyssomay.github.io/clj-dm/doc/start.html

Here follows more general information about the different functions.

Starting the program

There are several ways to start the DM program. The *first alternative* and the easiest one is to double-click the standalone. It includes everything that is needed. The DM lisp core is included from the time when the program was built.

In order to play using an external software synth you need sometimes to start that application first and then select it in DM from the “tools” icon at the upper left corner of the main window.

The *second alternative* is to start DM with an external version of the DM lisp core library. To get the latest DM lisp core code, the following procedure can be used:

1. Download Director Musices.
2. Download the `dm-source` folder to your computer from https://github.com/docfry/Director-Musices
   Alternatively, use Github or an equivalent program to synchronize a local copy of the repository.
3. Macintosh: Start the Terminal and enter the following command:
   `java -jar <your program folder>/director-musices-3.1.5.jar --dm-path <your lisp core folder>/dm-source --watch`
   It should be similar in windows using the command prompt, but this has not been tested.
   With the `--watch` command, the program will automatically reload the files that are changed. This facilitates the development of new rules.

A *third alternative* is to run the DM core without the GUI. It has been adapted to run in LispWorks Personal Edition, which is an integrated environment for lisp development (http://www.lispworks.com/downloads/). It does not include any graphics, but all functionality is present using the command line interface as well as a few file dialogs. It should also be possible to run it in any other common lisp environment, although some minor adjustments may be needed.

Loading Music

Choose "Open score" on the file menu. This assumes the custom format ‘.mus’. It is a text file containing all score information, see below.

Playing

Push the play button in the main window. Technically, it first generates a list that is sorted by time order before the actual playing.

Applying Rules

Select "Open default rule palette" in the initial window or on the File menu and push “Apply & Play”. This will apply all the rules listed in the rule pane with quantities specified by the scrollbars. It is recommended

to listen to each rule in isolation using exaggerated, medium, and negative quantities. In this way, it is possible to get an understanding of how each rule affects the performance. When several rules change the same parameter, the total change is the sum of changes from each rule. The combination of rules acting on the same parameter is additive. That is, if two rules (acting on the same note) increase the sound level by 1 dB, the resulting increase is 2 dB.

It is also possible to load other rule palettes (extension .pal) and define your own. Several rule palettes can be open at the same time for easy comparison of different performances.

When a rule affects more than one performance parameter, additional scaling of these can be done using the rule input parameters in the text window to the right of the rule name. For example,

:amp 0

means that the effect on the sound level will be zero.

:dur 2

means that the effect on the duration (interonset) will be twice the amount normally given by the rule at the specified quantity setting.

Observe that all operations on the music such as rule application and playing are only done on those tracks that are selected in the track pane.

Menu Commands

File menu

`Open Score...`

The main function to open a score file (default extension ".mus"). Initialize also the performance variables. The main window is updated.

`Open test score`

Opens the default score, which is the first part of Mozart's A major sonata.

`Open Performance...`

Opens a file that is containing both the score and all performance variables already set (default extension ".per").

`Import score from MIDI file...`

Reads a type 1 or type 0 midifile (default extension ".mid"). The tracks will be numbered according to the order in the midifile.

It is recommended to use quantized midi files coming from a score editor. If a score editor is used, try to make sure that the note durations are the same as printed on the screen. For example, the Encore program by default makes the play duration of each note to be 90% of it note value. This introduces in Director Musices an extra rest after each note, which in turn makes the rule application unpredictable. Also, Musescore (v. 3.6.2) introduces a small rest. Sibelius (v. 2021.2) works fine if there are not any specific playing instructions (like an imported performed midi file), although triplets seem to have problems.

However, it is possible to open almost any midi file. The rules use in most cases only the duration as input and do not rely on note values. These are needed only for the score display.

If there are several independent voices in one track, they are converted to a single voice, with the exception that simultaneous notes with the same duration are allowed. A type 0 midifile will be split in different tracks according to midi channel information. Note velocities and volume are read and translated to sound level using the default velocity curve. Pitch bend is read and converted to time shapes that will be stretched if the timing is changed. Bank select MSB, Bank select LSB, Pan, and Reverb is read. Most other control changes are ignored.

Note that most MIDI messages like meter and tempo are only set at the beginning of the score.

`Save Score As...`

Saves the score and track variables of the current music. i.e. not the performance variables (default extension ".mus").

`Save Performance As...`

Saves all variables of the current music. i.e. also the performance variables (default extension ".per"). This is the same as the internal representation of the performance at the time of saving. It can be used to save different kinds of pre-computed performances. "Open Performance..." must be used to open the file.

`Save pDM As...`

Will apply most of the rules one by one and store the deviations for each rule, parameter, and note along with the score in a custom format text file with the extension '.pdm'. This pDM file can then be played in

the pd program pDM changing the rules quantities in real-time. It is available on the KTH music performance website, see link above.

`Export Performance to MIDI file...`

Will save the current music in a midifile type 1 with the same midi-codes as when it is played (default extension ".mid"). Note that the selected synth type in the music window will affect the midi-velocity scaling.

`Open rule palette...`

Opens a rule palette file (default extension ".pal"). This opens a window where to interactively work with the rules. The rule names can also be changed directly in the window.

`Open default rule palette`

Opens a rule palette window with default rule settings. This is opened at startup.

`Quit`

Exits the Director Musices program GUI. Make sure that there are no unsaved changes since the program currently does not check for that.

Edit menu

`Tempo...`

Changes the score tempo of the loaded music. All durations are scaled accordingly, implying that the performance is unaffected. Note that it is important to use the right tempo when the rules are applied since many of them are sensitive to absolute durations.

`Octave...`

Octave transposition of the Active tracks.

`Meter...`

Changes the meter variable (mm) on the first note of each track, removes all old bar marks, and attempts to mark the new bars using the rebar function.

`Key...`

Changes the key and modus variable (key, modus) on the first note of each track. The key information is used for example by the Harmonic charge rule. The display of the score is not affected.

`Remove parameter`

Deletes one note parameter in the whole score.

`Remove all phrase marks`

Deletes all 'phrase-start' and 'phrase-end' markers in the score.

`Reset sound level`

Set all 'sl' variables to zero.

`Reset performance`

Reset all performance variables. Corresponds to the initial performance after loading a score file.

`Rebar`

Removes all old barlines (bar) and attempts to mark new barlines using each found meter variable. If the nominal durations of the notes do not add up to the computed bar duration according to Meter and Tempo, a bar is not marked at that position.

`Convert chord list to chord name`

Changes the format of the harmonic analysis from chord list <(“C” “E” “G”)> to chord name <“Cmaj”>

`Distribute chord analysis (not available in 3.1.3)`

A “q” variable found in one track is distributed to all simultaneous notes in the other tracks.

`Distribute phrase analysis`

A “phrase-start” or “phrase-end” variable found in one track is distributed to all simultaneous notes in the other tracks.

`Print all score vars`

Prints in the terminal window all track and note variables sequentially. This is mostly for debugging. The program needs to be started from the terminal.

`Print all score vars round`

Same as above but with all decimals rounded.

`Transpose from major to minor`

Makes a ‘modal’ transposition of all notes using the key and modus variable. For example, in C major all the E, A, and B notes are changed to Eb, Ab, and Bb. Works only for tonally rather simple melodies within the key. This will result in a minor dominant chord.

`Transpose from major to minor`

The corresponding function in the opposite direction.

`DM var settings… (only Windows, not available in 3.1.5)`

Here most of the global parameters can be changed. This is not yet documented so it is mostly useful for the programmers.

Help menu

`Log`

Prints some information about the program processing.

`Command line`

This is a simple common lisp command line interpreter. Any function defined in the DM core library can be called here.

Window commands

Transport window (top pane)

Buttons for play, stop, and selecting MIDI output. The scaling affects the size of the score (currently a bug is making the lines too short when making the score smaller).

It is also possible to play from any position in the score by clicking the grey bar above the score.

Track window (upper pane)

In order from left to right:

`First checkbox`

Selects the track. Only the selected tracks are used for any processing, such as rule application, editing from the menu, and also saving the score.

`Track name`

The name of the track.

`Volume`

This is the MIDI volume command that is sent before the first notes. Note that it only works if the tracks are played on different MIDI channels.

`Pan`

MIDI pan command sent before the first notes. Separate MIDI channels needed.

`Synthesizer object`

It will change the mapping of sound level in dB to MIDI velocity, volume in dB to MIDI volume, pitch bend range, and in some cases the midi program list, see the section below.

`Program`

MIDI program selection.

`MIDI channel`

1-16

`Track delay`

A fixed delay in milliseconds for the track. Can used to compensate for different delays in the external synthesizers.

Score window (middle pane)

Right-clicking on a note brings up a menu with the following items.

`Set phrase-start (4 5 6) etc…`

Phrase marks on three levels can be inserted at that position. For the details, see the phrasing rule section below.

`Edit note…`

All variables for that note can be edited. The same window can be opened by double-clicking on a note in the score.

`Show parameter…`

Any note variable inserted here is shown below the score.

`Show graph…`

Draws a graph below the staff. Currently, there are four different graphs available, Duration (ms), Sound level (dB), Duration difference (%), and Pitch difference (cent).

The duration difference shows the ratio in percent between the performed duration and the nominal duration given by the score. This is the recommended way to display timing changes.

Observe that the shown sound level values are only accurate for the specified synthesizer in the track window. Also, when the velocity sensitivity in the synthesizer can be changed, its needs to be set according to the settings in the selected synthesizer object.

Observe that for the pitch difference, the right pitch bend range has to be set in the synthesizer (usually 2 semitones).

Rule palette window (bottom pane)

`Apply`

Will start from the nominal score and then apply all rules in the list sequentially top-down. The rules are additive; thus, the order is normally not important. However, there are a few exceptions which cannot be defined in this way.

`Apply & Play`

Just a combination for convenience.

`Rule interact:`

If it is selected, the effect on the same note variable on the same note is compressed in the sense that it will be less than the sum of the values from each rule. The number indicates the amount of compression (setting not currently saved in the rule palette file), see [31].

`Sync:`

In a polyphonic score, this is the synchronization rule that is applied. See ensemble timing rules below for details (setting not currently saved in rule palette file).

`Add rule`

New rules can be added to the rule palette by writing the same rule name as specified in the tables below.

Performance Rules

Following is a comprehensive list of the performance rules.

All rules (whenever applicable) have a *quantity parameter k*, which controls the general effect. A *k* value of 1 corresponds approximately to a normal application of the rule. This is, however, dependent on the musical context. Negative *k* values can be used for the reversed effect. For example, the rule Duration-contrast can be used both with negative and positive values depending on the musical intention.

The *key parameters* can be used for additional control of the rules. In cases where the rule changes several performance variables, the key parameters can be used for controlling the amount of change for each performance variable.

The rules are described in detail in different publications, see the references and bibliography below. An overview is given in [28], see also [21]. Many of the rules are discussed in [4]. There is a technical description of many rules in [8]. The references in the tables refer primarily to the technical descriptions.

General key parameters

Most of the rules with more than one affected parameter (e.g. affecting both dr and sl) has a corresponding scaling parameter. These multiply the change of the parameter by the rule with the given value.
For example, ':dur 2' will double the rule impact on duration (the 'dr' parameter in the score).

:dur	scaling factor for IOI (dr)
:amp	scaling factor for sound level (sl)
:droff	scaling factor for articulation (dro)
:vibamp	scaling factor for vibrato extent/amplitude (va)
:vibfreq	scaling factor for vibrato rate/frequency (vf)

Phrasing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Punctuation	dr dro	:dur 1 :duroff 1 :markphlevel7 nil	Automatically locates small tone groups and marks them with a lengthening of the last note and a following micropause [1].	x
Phrase-articulation	dro dr	:dur 1 :duroff 1 :phlevel 5 :subphlevel 6	Micropauses after phrase and subphrase boundaries, and lengthening of the last note in phrases [13]. The phrase boundaries must be marked in the score.	x
Phrase-arch	dr sl	:phlevel 6 :dur 1 :amp 1 :acc 1 :turn 2 :next 1 :2next 1:last 1 :accfn :powerfn :decfn :powerfn :power 2	Each phrase is performed with an arch-like tempo curve: starting slow, faster in the middle, and ritardando towards the end. The sound level is coupled so that a slow tempo corresponds to a low sound level. See [5] and below for an explanation of the key parameters. The phrase boundaries must be marked in the score.	x
Phrase-ritardando	dr sl	:phlevel 4 :amp 2 :acc 0 :turn –2500 :next 1.5 :2next 2 :last 1 :accfn : powerfn :decfn : powerfn :power 3.5	An attempt to make Baroque type of phrasing with only a slight ritardando at the end of each phrase. It applies the Phrase-arch rule with the listed default key parameters. Note that the negative turn parameter specifies ritardando start in ms from the end of the phrase. Not formally evaluated.	x
Final-ritard	dr	:q 3	Ritardando at the end of the piece similar to a stopping runner [3].	x
High-loud	sl		The higher pitch the louder [8].	x

The rules Phrase-articulation, Phrase-arch, and Phrase-ritardando need as input phrase boundaries marked in the score by hand. One way to do this is to right-click on the note where you want to add a phrase mark in the score window. A pop-up menu appears in which you add for example “phrase-start (5 6)”. This means that a phrase starts both at the hierarchical levels 5 and 6, where 4 is the highest level meaning the longest phrase that typically corresponds to a complete melody. Each phrase-start mark must be followed by a phrase-end mark on the same level. In a polyphonic score, it is often convenient to put all the phrase marks on the first note of the measure in the first track. Then the command “Distribute phrase analysis” on the Tools menu can be used to distribute the phrase marks to all tracks. For a simple example, see the default score or “Ekor.mus”. All the phrase marks are shown on top of the score.

Phrase-arch key parameters

:phlevel phrase level corresponding to the marks in the score

:dur scaling factor for IOI (dr)

:amp scaling factor for sound level (sl)

:acc scaling factor for acceleration in the beginning

:turn Positive float number (0.0 to 1.0) relative turning position in the phrase
Positive integer (1 to n) note position from the end of the phrase
Negative integer (-1 to -n) turning position determined in milliseconds from end

:last scaling factor for last note

:next scaling factor for next phrase level ending

:2next scaling factor for second next phrase level ending

:accfn Accelerando shape:

:powerfn A power function used with :power
:runrit Modelled after stopping runners [3], same as Final-ritard, used with :power
:handgest Hand gesture model [24]

Old names for backward compatibility, cannot be used in GUI:

'power-fn-acc A power function used with :power
‘runrit-fn-acc Modelled after stopping runners [3], same as Final-ritard, used with :power
‘hand-gesture-fn-acc Hand gesture model [24]

:decfn Similar for deceleration

:power curvature of the power function or runrit model

Harmonic and Melodic Tension

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Melodic-charge	sl dr va	:amp 1 :dur 1 :vibamp 1	Emphasis on notes remote from the current chord [8].	x
Harmonic-charge	sl dr	:amp 1 :dur 1 :vibfreq 1	Emphasis on chords remote from the current key [8].	x
Chromatic-charge	dr sl		Emphasis on notes closer in pitch. Intended primarily for contemporary atonal music [9].	x

All tonal tension rules need as input the harmonic analysis marked in the score. This is currently done by hand using a similar procedure as for the phrase analysis described above.

Metric patterns and grooves

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Inegales	dr		Makes long-short patterns of consecutive eighth notes [8]. Also called swing eighth notes. Use no-sync in rule window. k=1 gives a ratio of 1.56:1.	x
Ensemble-swing	dr	:solo :bass :drums	Swing eighth notes proportional to tempo. Soloist and drums approx. synchronized at offbeat. Key parameters specify track names (e.g. :solo "Melody") Partly described in [14]. Use no-sync in rule palette window. This rule is replaced with the new rules for ensemble swing below.	x
Offbeat-sl	sl		Increase sound level at offbeats. K=1 gives 3dB increase.	x
Double-duration	dr		Decrease the duration contrast for two notes with the duration relation 2:1 (e.g. a quarter followed by an eighth) [8]. With k=1 it will produce a ratio of 1.68:1.	x
Triple-duration	dr		Decrease the duration contrast for two notes with the duration relation 3:1 (e.g. a dotted eighth followed by a sixteenth). Not evaluated.	x
Note-triplet-duration	dr		Increase contrast between a note and a following triplet of the same total nominal duration. Tentative formulation.	x

Ensemble Swing

These rules are intended to be used in a jazz ensemble consisting of soloist, bass, and drums. The purpose is to change the timing of these instruments according to a typical swing groove. They are described in detail in [32].

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**In DM 3.1.5**
Swing-ratio-tempo-prop-all	dr	:slope :constant	Sets the swing ratio of all offbeat eighth notes in all tracks. k=1 without key parameters gives a ratio corresponding to an average of recorded performances. :slope and :constant can be used for specifying the linear dependence of swing ratio as a function of tempo. Use no-sync in rule palette window. Needs to be applied before the other ensemble swing rules.	x
Swing-beat-delay-tempo-prop-solo	dr	:trackname :slope :constant	All onsets except the identified offbeats are delayed in the specified track according to measurements of soloists. :trackname must be provided. :slope and :constant can be used for specifying the linear dependence of delay as a function of tempo. Use no-sync in rule palette window.	x
Swing-beat-delay-tempo-prop-bass	dr	:trackname :slope :constant	Same as previous but for the bass track with other parameter settings.	x
Swing offbeat accent	sl	:trackname	Apply a dynamic accent to all identified eighth notes occurring on an offbeat. A value of $k = 1$ corresponds to a 3 dB increase in sound level. :trackname needs to be specified.	x
Swing-delay-one-track-ms	dr	:trackname	Optional utility rule that can move a whole track back and forth in time. The input is the delay in milliseconds. For moving the track earlier, a negative value is given. :trackname needs to be specified.	x

Swing-offbeat-delay-one-track-ms	dr	:trackname	This is an optional rule that can be used for moving the offbeat notes back and forth in time. The input is the delay in milliseconds. For ahead in time give a negative value. :trackname needs to be specified.	x

Articulation

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Leap-articulation-dro	dro		Micropauses in leaps [8].	x
Repetition-articulation-dro	dro		Micropauses in tone repetitions [8].	x
Score-legato-art	dro		Tone overlaps for notated legato [23].	x
Staccato-art	dro		Micropauses between tones for notated staccato [23].	x

Immanent accents

The immanent accent models add salience values on each affected note ranging from 1 to 5 for each accent type. Thus, they generate only the note variables listed below and will not affect the performance.

Rule name	**Added note variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Mark-metrical-accent	:beat0 :beat1 :beat2 :beat3 accent-m	(no *k* parameter)	Marks 4 different metrical levels and compute the salience [29].	x
Mark-beat-levels	:beat0 :beat1 :beat2 :beat3	(no *k* parameter)	Utility function that marks 4 different metrical levels [29,30]. Used by mark-metrical-accent.	x
Mark-melodic-accent	accent-c	(no *k* parameter)	Compute the melodic accent salience [29].	x
Mark-harmonic-accent	accent-h	(no *k* parameter)	Tentative computation of harmonic accent salience using the rule harmonic-charge. Note that the harmonic accent described in [29] is not implemented. The chord analysis needs to be present in the file.	x

Performed accents

The performed accent rules apply salience values to the performance by increasing sound level and duration around the accented note. They need to have the saliences marked in the score, for example, generated from the immanent accent rules or manually.

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
melodic-contour-accent	sl dr	:curve :linear :amp 1 :dur 1 :width 1	Applies the melodic accent salience (accent-c) to the performance [31].	x
harmonic-accent	sl dr	:curve :linear :amp 1 :dur 1 :width 1	Applies the harmonic accent salience (accent-h) to the performance [31].	x
metrical-accent	sl dr	:curve :linear :amp 1 :dur 1 :width 1 :marker 'accent-m	Applies the metrical accent salience (accent-m) to the performance [31].	x

Key parameter values
:curve :linear :quadratic :cubic :exponential :cosine :gaussian :hand-gesture
:amp scaling parameter for sound level (sl)
:dur scaling parameter for duration (dr),
:width scaling parameter for width (must be > 0)

Note that the application of the accent rules works for polyphonic scores works but that the melodic contour accent can yield unexpected results for the application of duration variations. The reason is that the duration variations are computed on the sync track, which is a mixture of all tracks, thus, with some jumps in the melody. We recommend using the "simple-mel-sync" option. The other accents work fine.

Also, when several accent rules are combined, it is usually necessary to "compress" the total amount of variation on each note. This is done with the "Rule interact:" option set to a value>1 in the rule window (test 2 or 3).

Timing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Duration-contrast	dr sl	:amp 1 :dur 1	The longer note the longer and louder, and the shorter the shorter and softer. [8]	x
Duration-contrast-art	dro		The shorter note the longer micropause [2, 22].	x
Social-duration-care	dr		Increase duration for extremely short notes [9].	x
Faster-uphill	dr		Decrease duration for notes in uphill motion [8].	x
Leap-tone-duration	dr		Shorten the first note of an ascending leap and lengthen the first note of a descending leap [8].	x

Ensemble timing

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Melodic-sync (no k parameter)	dr		Generates a new track consisting of all tone onsets in all tracks. When there are several simultaneous onsets, the note with the maximum melodic charge is selected. All rules are applied on this sync track and the resulting durations are transferred back to the original tracks [8, 10].	x
Simple-mel-sync (no k parameter)	dr		Generates a new track as above but uses always the top note in chords. Pauses are treated as notes. Is more reliable and faster for larger scores.	x
Bar-sync (no k parameter)	dr		Synchronize the tracks on each barline.	-

Intonation

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
High-sharp	dc		The higher pitch the higher [8]	x
Mixed-intonation	dc		Each note is initially intonated as melodic intonation and then gradually changed to harmonic intonation [10].	-

Harmonic-intonation (no k parameter)	dc		Beat-free intonation of chords relative the root. Needs an harmonic analysis added to the score.	x
Melodic-intonation	dc		Close to pythagorean tuning with e.g. sharp leading tones [8].	x

Piano specific

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Pedaling	Pedal-perf		Performs piano pedaling from the input variable Pedal.	-

Noise

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Noise	dr, sl	:amp 1 :dur 1	Generates random variations of inter-onset durations and soundlevel combining the three noise rules below. Use it without global synchronization to obtain small timing differences between tracks [24].	x
Internal-clock-noise Internal-clock-noise-sync	dr	:sync-type 'simple-mel-sync	Modeling variations in the internal clock with or without synchronization of polyphonic tracks [24].	x
motor-noise	dr		Modeling timing variations in the motor system of a performer [24]. If it is used without synchronization it will make small timing differences between the tracks.	x
motor-noise-amp	sl		Modeling amplitude variations in the motor system of a performer [24].	x

Utility functions

This is a short list of utility functions that can be used in a rule palette.

Rule name	**Affected performance variables**	**Key parameters** and default values	**Short description**	**in DM 3.1.5**
Normalize-sl (no k parameter)	sl		Normalize the sound level over each track in the piece.	x
Maximize-sl (no k parameter)	sl		Translate the sound level so that max sl corresponds to max midi velocity. The result depends on the selected synthesizer.	x
Set-sound-level	sl		Translate the sound level. k=1 gives 3 dB increase.	x
Scale-sound-level-range	sl		Scale the sound level with a multiplication factor. k=1 gives no change. Can be used for changing default values for score dynamics marks.	x
Normalize-dr (no k parameter)	dr		Scale the durations so that the total length is the same as the nominal length.	x
Normalize-dr-bar (no k parameter)	dr		The same as above for each measure.	x
Scale-duration	dr		Scale all durations with the factor given by the k value. Note that k=1 gives no change.	x

Overall-articulation	dro		Applies a constant articulation to all notes followed by another note, thus not on notes before rest. k=1 gives 25% silence.	x

References and bibliography

[1] Friberg, A., Bresin, R., Frydén, L., and Sundberg, J. (1998) "Musical punctuation on the microlevel: Automatic identification and performance of small melodic units*", Journal of New Music Research*, 1998, Vol. 27, No. 3, pp. 271-292

[2] Bresin, R. & Friberg, A. (2000). Emotional Coloring of Computer-Controlled Music Performances. *Computer Music Journal*, 24(4), 44-63.

[3] Friberg, A. and Sundberg, J. (1999) "Does music performance allude to locomotion? A model of final *ritardandi* derived from measurements of stopping runners", *Journal of the Acoustical Society of America,* 105(3), pp 1469-1484

[4] Friberg, A. (1995). *A Quantitative Rule System for Musical Performance*, doctoral dissertation, Royal Institute of Technology, Sweden

[5] Friberg, A. (1995). "Matching the rule parameters of Phrase arch to performances of Träumerei: A preliminary study", in A. Friberg and J. Sundberg (eds*.), Proceedings of the KTH symposium on Grammars for music performance May 27, 1995*, pp. 37-44.

[6] Friberg A, Sundberg J & Frydén L (1994) "Recent musical performance research at KTH", in J. Sundberg (ed.), *Proceedings of the Aarhus symposium on Generative grammars for music performance 1994*, 7-12.

[7] Sundberg, J. (1993). "How can music be expressive?", *Speech Communication* 13, pp. 239-253.

[8] Friberg, A. (1991). Generative Rules for Music Performance: A Formal Description of a Rule System", *Computer Music Journal*, 15 (2), pp. 56-71.

[9] Friberg, A., Frydén, L., Bodin, L.-G., and Sundberg, J. (1991) "Performance Rules for Computer-Controlled Contemporary Keyboard Music", *Computer Music Journal*, 15-2, pp. 49-55.

[10] Sundberg, J., Friberg, A. & Frydén, L. (1989). "Rules for automated performance of ensemble music", *Contemporary Music Review*, 3, 89-109.

[11] Sundberg, J., Friberg, A. & Frydén, L. (1991). "Common Secrets of Musicians and Listeners – An analysis-by-synthesis Study of Musical Performance" in P. Howell, R. West & I. Cross (eds.), *Representing Musical Structure*, London: Academic press.

[12] Carlson, R., Friberg, A., Frydén, L., Granström, B. and Sundberg, J. (1989). "Speech and music performance: parallels and contrasts", *Contemporary Music Review* 4, pp.389-402.

[13] Friberg, A., Sundberg, J. & Frydén, L. (1987). "How to terminate a phrase. An analysis-by-synthesis experiment on the perceptual aspect of music performance", in A. Gabrielsson (ed.), *Action and Perception in Rhythm and Music*, Stockholm: Royal Swedish Academy of Music, Publication No. 55, pp. 49-55.

[14] Friberg, A. and Sundström, A. (2002). Swing ratios and ensemble timing in jazz performance: Evidence for a common rhythmic pattern. *Music Perception*, 19(3), 333-349.

[15] Frydén, L., Sundberg, J. & Askenfelt, A. (1988). "Perception aspects of a rule system for converting melodies from musical notation into sound", *Archives of Acoustics* 13 (3-4), pp. 269-278.

[16] Sundberg, J. (1988). "Computer synthesis of music performance", in J. A. Sloboda (ed.), *Generative Processes in Music*.

[17] Sundberg, J. and Frydén, L. (1984). "Teaching a computer to play melodies musically", *Analytica, Festschrift for Ingemar Bengtsson*, Publications issued by the Royal Swedish Academy of Music, Nr 47, 67-76.

[18] Sundberg, J., Askenfelt, A. and Frydén, L. (1983). "Musical performance: A synthesis-by-rule approach", *Computer Music Journal*, 7, 37-43.

[19] Sundberg, J., Frydén, L. & Askenfelt, A. (1983). "What tells you the player is musical? An analysis-by-synthesis study of music performance", in J. Sundberg, (ed.), *Studies of Music Performance*, Publication issued by the Royal Swedish Academy of Music Nr. 39, Stockholm, pp. 61-75.

[20] Thompson, W. F.,Sundberg, J., Friberg, A., and Frydén, L. (1989). "The Use of Rules for Expression in the Performance of Melodies", *Psychol. of Music* 17, 63-82.

[21] Friberg, A, Colombo, V, Frydén, L and Sundberg, J (2000) “Generating Musical Performances with Director Musices”, *Computer Music Journal*, 24(3), 23-29.

[22] Bresin, R. & Friberg, A. (2000) Emotional Coloring of Computer-Controlled Music Performances. *Computer Music Journal*, 24:4, 44-63

[23] Bresin, R. (2000) *Virtual Virtuosity - Studies in automatic music performance*. Doctoral Dissertation, Speech Music and Hearing, Stockholm: KTH, TRITA-TMH 2000:9, ISSN 1104-5787, ISBN 91-7170-643-7

[24] Juslin, P. N., Friberg, A., & Bresin, R. (2002). Toward a computational model of expression in performance: The GERM model. *Musicae Scientiae special issue 2001-2002*, 63-122.

[25] Sundberg J, Friberg A, and Bresin R (2003) Attempts to reproduce a pianist's expressive timing with Director Musices performance rules. *Journal of New Music Research*, 32(3), 317-326.

[26] Friberg, A. and Battel, G., U. (2002). Structural Communication. In (R. Parncutt & G. E. McPherson, Eds.) *The Science and Psychology of Music Performance: Creative Strategies for Teaching and Learning*. New York: Oxford University Press.

[27] Friberg, A., Bresin, R. and Sundberg, J. (2006). Overview of the KTH rule system for music performance. *Advances in Experimental Psychology special issue on Music Performance*.

[28] Friberg, A. (2006). pDM: an expressive sequencer with real-time control of the KTH music performance rules. *Computer Music Journal*, 30(1), 37-48.

[29] Bisesi, E., Friberg, A. & Parncutt, R. (2019). A Computational Model of Immanent Accent Salience in Tonal Music. *Frontiers in Psychology*, 10(317), 1-19.

[30] Friberg, A., Bisesi, E., Addessi, A. R. & Baroni, M. (2019). Probing the Underlying Principles of Perceived Immanent Accents Using a Modeling Approach. *Frontiers in Psychology*, 10.

[31] Friberg, A. & Bisesi, E. (2014). Using computational models of music performance to model stylistic variations. In Fabian, D.; Timmers, R.; Schubert, E. (Ed.), *Expressiveness in music performance: Empirical approaches across styles and cultures* (pp. 240-259). Oxford University Press.

[32] Friberg, A., Gulz, T. and Wettebrandt, C. (2022). Computer Tools for Modeling Swing in a Jazz Ensemble. Submitted for publication. Preprint is available on request.

Playing Details

The playing function uses the performance variable dr for the total duration of a note until the next note onset and dro for the offtime duration. The playing function also sends all the other performance variables as listed above before note on.

Synthesizer Objects

The performance variables are sent to the selected synthesizer object in the track window. The synthesizer object translates the performance variables to MIDI codes depending on the selected synthesizer. The purpose of this transformation is to have synthesizer-independent parameters in the program that is expressed in physical measures. Most of the synthesizers listed in the track window have calibrated functions for sound level to MIDI velocity and for volume to MIDI volume. This assumes certain settings in the synthesizer since for example velocity scaling often is available as a parameter in synthesizer patches. The pitch bend range in the synthesizer object is usually 2 semitones. The durations will always be played right but if the sound chosen in the synthesizer has long attack and/or decay times, the micropauses will not be properly performed.

For non-scientific purposes, it can be sufficient to try the different synthesizer objects (for example SBlive) and choose the one that has a good balance of both sound level and duration changes.

We can implement new synthesizer objects for your synthesizers if you help us with the measurements. Contact us for further info.

Synths available in the DM program track window

Pinnacle

General midi module from Kurtzweil found on the Pinnacle PC sound board.

SBlive

Sound Blaster live PC sound card.

Roland-A90

The optional sound module in the Roland A90 keyboard controller.

Roland-PMA5, Roland-1010

General midi synthesizers

Kontakt 2 piano, Kontakt 2 wind

Different patches in the Kontakt sampler.

Technics SX-P30

An electronic piano.

Yamaha Clavinova CLP370

An electronic piano.

Yamaha Disklavier E3

A grand piano with midi control.

Yamaha P90

An electronic piano.

Proteus

Sample player from e-mu. Close to the factory settings in Proteus/2 especially the woodwinds. Assumes a pitchbend of ±2 semitones, vel curve 2. Vibrato parameters are not included.

Recent synths only available in the DM core

Yamaha Disklavier upright

The MIDI volume should be set to 100.

Pianoteq7

NY Steinway D prelude, default settings (straight line velocity curve, Dynamics 36 dB)

Old synth definitions only available in DM core

SampleCell

Sample card for Mac. Included in the DM Mac version are some patches in SampleCell that are adapted to Director Musices parameters. They assume that the samples are on the basic CD-ROM shipped with SampleCell. A good way to start is to connect it to SampleCell with the FB01 organ sound loaded. Uses pitch bend ±1 semitone, vibrato extent as midi pressure and vibrato speed as the modulation wheel.

S3000

Sampler from AKAI. Pitchbend ±2 semitones, vel sensitivity +20 (factory setting).

FZ1

Old sampler from Casio.

Fb01, dx21

Old Yamaha synthesizers.

Musse

Singing synthesizer developed at Speech, Music and Hearing, KTH.

Generator

A general purpose software synthesizer from Native instruments. Now called Reactor. Only used for patches developed at the department.

Music File Format

The music is organized into tracks that consist of a list of segments (or notes). All parameters are associated with a segment. A segment can have several pitches, but have only one duration. Examples of the file format can be studied by open any ".mus" file in a text editor.

| means logical *or*

; means comment

The present format is a normal text file formatted in the following way:

```
<track-type>
<track-variable> <value>
<track-variable> <value>
.
.
.
(<var-name> <value> < var-name> <value> ...)     ;each parenthesis corresponds to a segment
(<var-name> <value> < var-name> <value> ...)
.
.
.
<track-type>
<track-variable> <value>
<track-variable> <value>
.
.
.
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
```

In the old format (still supported) a new track starts with the track name followed by the segments:

```
"<track name>"
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
"<track name>"
(<var-name> <value> <var-name> <value> ...)
(<var-name> <value> <var-name> <value> ...)
.
.
.
```

```
track-type is     mono-track | voice-track | multi-track
```

Track Variables

variable **value**

```
:trackname              "<string>"
:midi-channel           1-16
:midi-initial-volume    0 to -64 (dB)
:midi-initial-program   1-127
:synth                  "<synth-name>"    ;synth-name must be one of the
                                          predefined types in DM
```

Input music notation variables

These are the variables used for the representation of the input score. This set of variables is not sufficient for describing a score. Many things are missing such as articulation marks. However, it is easy to extend the input notation with new variables and write rules that convert then into performance variables.

variable **value**

```
n           ("<tone><octave>" . <notevalue>) |
                  (("<tone><octave>" "<tone><octave>" ...) . <notevalue>)
            tone = C | C# | Db | D | D# | Eb | E | Fb | E# |F | F# |
                    Gb | G | G# | Ab | A | A# | Bb | B | Cb | B# |
            octave = 0 ... 9
            notevalue = 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128
            ;the representation of a note
            ;if several pitches, the notes should be in descending order of pitch

dot         1 | 2

tie         t
            ;tie to the next note

tuple       3 | 5 | <tuple-list>
            3 ;triplet  => ndr = ndr * 2 / 3
            5 ;quintuple => ndr = ndr * 4 / 5
           <tuple-list> = (<mul> <div>) ;=> ndr = (ndr * <mul> / <div>)
            ;Must be inserted for each note it applies to.
            ;old symbol: t
```

;Alternative representation of note values used by the MIDI file input reader:
; no need for tuples, dotted, or tied notes.

```
n           ("<tone><octave>" <ratio> <ratio>...)
            | (("<tone><octave>" "<tone><octave>" ...) <ratio> <ratio>...)
            ;each ratio is a notevalue, eg, 1/4, 5/7, 2/3
            ; In this way, any fraction or subdivision of the beat can be represented
            ; the other notevalue variables are not used: dot tie tuple


rest        t
            ;must also have n (() . <notevalue>)

slur        t
            ;slur to next note
```

```
dyn         ppp | pp | p | mp | mf | f | ff | fff
            ; Dynamics

mm          <real number>
            ; Metronome tempo for the division in meter specification/minute.
            ; Needs to be specified on the first note but can appear on any note.
            If absent the default is mm=120.
            ;They are valid until the next mark.

meter       (<mul> <div>)
            mul = 1 ... 16
            div = 1 | 2 | 4 | 8 | 16
            ;ex: meter (4 4) means 4/4
            ;Needs to be specified on the first note but can appear on any note.
            If absent the default div=4.
            ;They are valid until the next mark.

key         "<chord-name>"
            chord-name = "C" | "F# " | …

modus       "maj" | "min"
            ;The modality of the key.

q           <chord-name> | ("<tone>" "<tone>" ...)
            chord-name = "Cmaj" | "F#min" | ...
            tone = see n above
            ;This is the harmonic analysis as either a name or a list of the chord notes beginning
            ;with the bass (or the root)
            ;A Cmaj chord: q ("C" "E" "G")

bar         <integer>
            ;The bar number beginning from 1.
            ;Will be inserted if not present.
            ;Optional

pr          <1..128>
            ;MIDI program number
            ;Optional.

bankmsb     <0..127>
            ;MIDI Bank MSB  select
            ;Optional.

banklsb     <0..127>
            ;MIDI Bank LSB  select
            ;Optional.

pan         <0..127>
            ;MIDI pan
            ;Optional.

reverb      <0..127>
            ;MIDI reverb
            ;Optional.

rit-start   t
```

```
rit-end     t
            ;ritardando start and end
             ;the note marked with rit-end should have an mm mark as well

acc-start   t
acc-end     t
             ;accelerando start and end
            ;the note marked with acc-end should have an mm mark as well

phrase-start (<level> <level> ...)
phrase-end   (<level> <level> ...)
            level = 1...7
            ;The levels are the hierarchical level of the analyzed
            ;phrase structure. 7 is the lowest level representing
            ;short melodic motives. 6 is the next level above.
             ;The format itself permits overlapping phrases.

legato-start      t
legato-end        t
            ;legato start and end

staccato          t
            ;staccato on single notes

accent      1 | 2 | 3
            ;dynamic accent on single notes in the score
            ;1 weak
            ;2 medium - normal accent like sf or >
            ;3 strong

pedal       0 | 1
            ;piano pedaling
            ;a value of 1 means pedal down and 0 pedal up
            ;a value of 0 needs only be inserted when there is no down pedal following
```

Music performance variables

These are the properties generated and altered by the rules and are used for playing and for generating midi files.

variable **value**

```
f0          <integer number>
            ;midi note number

dr          <real number>
            ;Total interonset duration in ms.
            ;This quantity is used for playing and is changed by the rules.

ndr         <real number>
            ;Nominal interonset duration in ms.
            ;Will not be altered by the rules.

dro        <integer number>
```

```
              ;Offtime duration. i.e. offset to the next onset.
              ;Optional.

sl            <real number> | <time-shape object>
              ;Sound level in decibels relatively a constant found in the synth object.
              ;Default value is sl=0.

dc            <integer number> | <time-shape object>
               ;Pitch deviation in cent from equal temperament
               ;Optional

va            <integer number> | <time-shape object>
              ;Vibrato amplitude in cent (+ -).
              ;Optional

vf            <real number> | <time-shape object>
               ;Vibrato frequency in Hertz.
               ;Optional

vol          <real number> | <time-shape object>
               ;volume in dB
              ;range: 0 to -64 dB
               ;will be sent as midi volume
               ;Optional


pedal-perf   0 | 1 | <time-shape object>
               ;piano pedaling with timetable
              ;value 1 means pedal down and 0 pedal up
```

Alternative without time-shape object

```
pedal-down   <real number>
               ;pedal down after delay (ms)
pedal-up     <real number>
               ;pedal up after delay (ms)
```

Speech notation variables

```
text   "<text>"
       ;input original text
```

Speech performance variables

```
ph     "<phoneme>"
       phoneme = a | a: ......
```

Writing New Rules

The following is a partial list of utilities for rule definition. The tools are built within the common lisp environment. Therefore, the syntax is the same as in common lisp (ANSI CL standard) and all common lisp resources are available. The serial note macros are the ones mostly used in the past and are therefore more robust. The recent extensions allowing grouping of segments and notes have not been very much tested yet.

Music Structure

The music is organized into tracks that can be processed either serially or in parallel. Each track consists of a list of notes, each of which has a list of variables. These variables can be the duration, the sound level, the MIDI key number or other properties that have been assigned to the notes. The assignment of such variables is normally realized by the rules using access functions, but the value can also be edited manually if the music is saved in a performance file (or in the score window).

Example of a variable list:

```
(dr 793.0  dro 54 ndr 854.0  sl 10  f0 50  bar 1  phrase-start (4)  key "D"
modus "maj")
```

This means that the note has interonset duration = 793.0 ms, "off-time" duration = 54 ms, nominal interonset duration = 854.0 ms, sound level 1 dB over middle value, MIDI key number = 50, bar #1, A phrase beginning on phrase level 4, and that the key is D major. A more complete description of variables is given in the section Music File Format above.

Rule top-level definition

Rules are defined by the normal lisp defun function using the format specified below. These are the formats recognized by the rule palette window.

```
(defun <rulename> (<k parameter> <additional key parameters>)
   <body>)
```

Defines a rule with the main rule parameter k. The k parameter should be included in all changes of performance variables within the rule. For example, a doubling of k should correspond to a doubling of the *changes* made by the rule. Additional parameters are specified using key parameters.

```
(defun <rulename> (<additional key parameters>)
   <body>)
```

Defines a rule without a k parameter. Additional parameters are specified using key parameters.

Serial Rule Macros

These macros will step through the score in chunks as specified by each macro and are used within the body of a rule definition. For example, each-note will consider all segments up to the next note marker as one chunk.

```
(each-track <body>)
```

Evaluate the <body> for each track in the score. Within the body, access functions are used to access the variables of the tracks.

```
(each-note <body>)
```

Evaluate the <body> for each note and track in the score serially. Within the body, access functions are used to access the variables of the notes.

```
(each-note-if
   <condition 1>
   <condition 2>
   <condition 3>
   .
   .
   .
   (then
     <body>)
```

This is the same as each-note with the addition of conditions. For each note, all conditions must be true to evaluate the body.

```
(each-segment <body>)
(each-segment-if <body>)
```

Evaluate the <body> for each segment and track in the score serially. Within the body, access functions are used to access the variables of the segments. These macros have the same function as each-note and each-note-if if the track is a mono-track. For a voice-track these macros work on a lower level, each segment corresponding to a voice segment or a phoneme.

```
(each-group
  `<group begin condition>
  `<group end condition>
  <body>)
```

Evaluate the <body> for each group of segments as specified by the begin and end conditions. Within the body, access functions are used to access the variables of each group as one chunk.

```
(each-group-if
  `<group begin condition>
  `<group end condition>
  <group condition 1>
  <group condition 2>
  <group condition 3>
   .
   .
   .
   (then
     <body>)
```

This is the same as each-group with the addition of conditions. First, a new track is created consisting of segment group objects (chunks). Then for each chunk, all group conditions must be true to evaluate the body for that chunk.

If an each-track macro is omitted the other macros are iterating over each track as well.

```
(each-note
```

is equivalent to

```
(each-track
```

```
    (each-note
```

Parameters available within macros

`*v*`

Contains the current list of objects

`*i*`

Contains the current object index (zero to length-1)

Serial Access Functions

Within the body of the sequencing macros, the access functions are used for accessing the variables in each chunk. The access functions manipulate normally the data for the whole chunk. They are also used for context definitions. A slowly changing variable (time-shape object) can be added over a whole chunk.

Any variable

```
(this <variable>)
(next <variable>)
(prev <variable>)
```

Return the specified variable of the current, next, or previous chunk. Also available are next2, next3, prev2, prev3.

```
(set-this <variable> <value>)
(set-next <variable> <value>)
(set-prev <variable> <value>)
```

Assigns the specified value to the variable in the current, next and previous chunk. Also available are set-next2, set-next3, set-prev2, set-prev3.

```
(add-this <variable> <value>)
(add-next <variable> <value>)
(add-prev <variable> <value>)
```

adds the specified value to the variable in the current, next and previous chunk. Also available are add-next2, add-next3, add-prev2, add-prev3.

```
(rem-this <variable>)
(rem-next <variable>)
(rem-prev <variable>)
```

removes the specified variable. Also available are rem-next2, rem-next3, rem-prev2, rem-prev3.

Duration

```
(this-dr)
(next-dr)
(prev-dr)
(next2-dr)
(prev2-dr)
(next3-dr)
(prev3-dr)
(set-this-dr <value>)
```

```
(set-next-dr <value>)
(set-prev-dr <value>)
(set-next2-dr <value>)
(set-prev2-dr <value>)
(set-next3-dr <value>)
(set-prev3-dr <value>)

(multiply-this-dr <factor>)
(multiply-next-dr <factor>)
(multiply-prev-dr <factor>)
(multiply-next2-dr <factor>)
(multiply-prev2-dr <factor>)
(multiply-next3-dr <factor>)
(multiply-prev3-dr <factor>)
```

Pitch

If f0 is a list (chord) these functions return only the first tone.

```
(this-f0)
(next-f0)
(prev-f0)
(next2-f0)
(prev2-f0)
(next3-f0)
(prev3-f0)
```

Accessing any note

```
(iget <index>)
(iset <index> <value>)
(iadd <index> <value>)
```

Destructive segment operations

```
(remove-this-segment)
(insert-segment-before-this <new-segment>)
```

Use these functions with caution using simple contexts as they destructively modify the segment-list e.g. (each-segment-if (first?) (remove-this-segment)) will remove ALL segments

Utilities

```
(first?)
(last?)
```

Returns true if it is the first or last chunk, respectively, otherwise returns false. Also first+1 and last-1?.

```
(i?next <index> <variable>)
```

Returns the index of the chunk containing <variable> after the chunk with < index>. If not found ->nil

```
(i?prev <index> <variable>)
```

Returns the index of the chunk with <variable> before chunk with < index>. If not found ->nil

```
(i?last)
```

Returns the index of the last chunk.

```
(exit-track)
```

Go to the end of the track.

```
(i?next-note <index>)
```

Returns the index of the next note excluding rests. Not found -> nil.

```
(i?prev-note <index>)
```

Returns the index of the previous note excluding rests. Not found -> nil.

```
(sharp?)
(flat?)
```

Returns true if the current note is sharp or flat, respectively.

```
(i?drsum <index> <total duration>)
```

returns the index of the chunk up to the total duration included the first note specified by index
returns at least index+1
if it gets to the end - returns last

```
(ndrsum <i-from> <i-to>)
(drsum <i-from> <i-to>)
```

returns the sum of the durations (float) not included i-to

Examples

```
(defun phrase-rule (k)        ;a complete rule for lengthening notes
    (each-note-if             ;before phrase-start markers
        (not (last?))
        (next 'phrase-start)
        (then
           (add-this 'dr (* 40 k))
         )))


(each-track
      (set-this-dr              ;increase the duration for the whole track
          (* (this-dr) 1.2) )) ;with 20 %


(each-note-if                   ;process this note if:
   (< (this 'dr) 500)           ;it is shorter than 500 ms
   (> (this 'f0) (prev 'f0))    ;and if the pitch is higher than previous
   (then
      ...

(each-group                                ;process phrase by phrase
   '(this 'phrase-start)                   ;group beginning
   '(next 'phrase-end)                     ;group end
   (then
     (set-this                             ;increase the sound level with an
       'sl                                 ;envelope over the phrase
       (make-time-shape ...

(each-group                                ;similarly process phrase by phrase
   '(this 'phrase-start)                   ;group beginning
```

```
'(or (last?) (next 'phrase-start))      ;group end, remember not specify
                                        ;context after the last segment
(then
  (print (this-dr))                     ;print the duration of each group
  (set-this 'sl (- *i*)) )))            ;set the sound level to each group
                                        ;index
```

Time shapes

The time shape objects are not yet documented.

Parallel Rule Macros

These macros step in parallel through the tracks one step for each new note in any track.

`(p-each-bar <body>)`
Executes the body each time there is a new bar in the music (not available in current DM version)

`(p-each-note <body>)`
Executes the body each time there is a new note event in the score

Parameters available within macros

For every step in the parallel rule macros, the following variables are set:

`*all-notes*`
A list of all notes (track-object and segment index) contained within the current position (alist)

`*cur-notes*`
all notes that is new for each step (alist)

`*ndr-to-next*`
Duration from the current step (time-slice) to next note

Parallel Access Functions

`(p-this)`
returns an alist of the tracks and variable values in *cur-notes*

`(p-set-this <value>)`
Sets a variable value for each note in *cur-notes*

`(p-this-all <variable>)`
returns an alist of the track and variable value for each track and note in *all-notes*

```
(v-iget (<track> . <index>) <variable>)
```
get the variable value for the specified note

```
(v-iset (<track> . <index>) <variable> <value>)
```
set the variable value for the specified note

```
(v-this <track>  <variable>)
```
get the variable value in the current note in the specified track

```
(v-this-all <track>  <variable>)
```
get the variable value from current or last current in the specified track

```
(v-this-plist <track>)
```
get the variable list in the current note in the specified track

```
(v-set-this <track> <variable> <value>)
(v-add-this <track> <variable> <value>)
(v-set-next <track> <variable> <value>)
(v-set-prev <track> <variable> <value>)
```
Set the variable value in the current, next or previous note in the specified track

Parallel Utility Functions

```
(p-first?)
(p-last?)
```

```
(v-i?next (<track>  . <index>) <variable>)
```
Returns a pair with the note index and variable after note i. If not found ->nil

```
(v-i?next-or-last (<track>  . <index>) <variable>)
```
Returns a pair with the note index and variable after note i. If not found -> last note

```
(v-i?last <track>)
```
Returns the number of the last note

```
(i?-barnr <track> barnr)
```
return note number from track and bar number else nil

```
(p-min pair-alist)
```
return minimum

if same: take only one

```
(p-min-all pair-alist)
```
if same rounded return a list of those

```
(p-max pair-alist)

(v-to-ni <track>)
```
