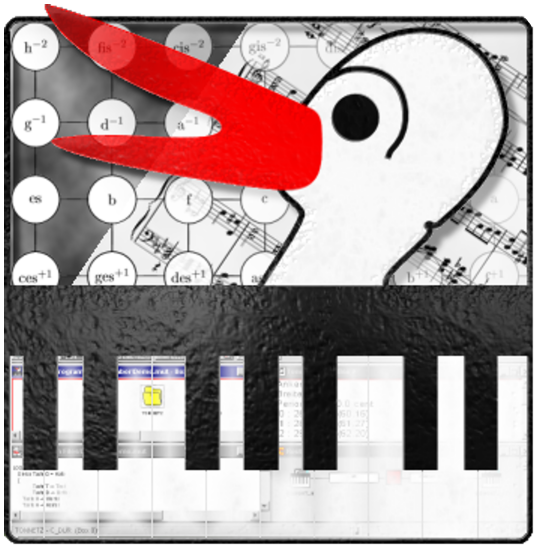
Reference manual

Mutabor

Ein computergesteuertes Musikinstrument zum Experimentieren mit Stimmungslogiken und Mikrotönen

Volker Abel, Peter Reiss,
Rüdiger Krauße und Tobias Schlemmer

Programmversion 3.0x (2018)



Contents

1 Language reference

On the one hand this reference manual shall be a short is reference book, on the other hand, it provides the basic description of the logic language of MUTABOR dar. It is not suitable for the Logical language to learn, for this we direct the reader to the Operation manual, but it provides exact information for dubious questions. The reference manual is designed to determine the functionality of the logic language. When a user of MUTABOR encounters that a programmed logic doesn't work the way he wants it to, then this reference manual will be the decision-making body, against which he can check whether the program logic is wrong, or whether MUTABOR not work correctly. This section describes the logic language MUTABOR in version 3.0.

2 Lexical conventions

A program consists of a text, that is saved in a file. It will be compiled in one single phase.

2.1 Input symbols

There are five classees of input symbols: Names, reserved words, numbers, operators and other delimiters. Spaces, tabulators, vertical tabulators, line feeds, page feeds as well as comments, usually denoted as white space, will be ignored, with the exception that they are necessery to distinguish direct neigbouring names, numbers or reserved words.

If the input been already dissambled up to a certain character, the next symbol will be the most possible string of characters, which can be interpreted as an input symbol.

2.2 Comments

Comments an entailed in douple quotes (''). Comments cannot be nested, but can span multiple lines.

2.3 Names

A name is a sequence of letters and numerals; the first character must be alphabetic. This includes the underscore (_) and the apostrophe (') .[1] Large and small letters are treated the same. Names can be at any length. To distinguish names of the entire name is used, and not just the first few letters, as it is commen in other programming laguages.

In the books of MUTABOR the terms *name* and *identifier* are used in parallel and with equal meaning. Furthermore, in some syntax graphs also terms of the type *interval name*, *tone name*, ... are used. These terms are considered syntactically as simple names. Such a name describes that a particular type of Objects are meant. In principle, this does not belong to the syntax of the logic language, but will be checked later semantically. However, it increases the understanding of the examples when seen directly, what kind of name comes into question.

[1] The underscore is one of the letters, so you can write meaningful names legibly, e. g., `LOGIC third_tone_mutating` is better than `LOGIC thirdtonemutating`, and the apostrophe counts as a letter so that tone with a simple or multiple apostrophies can provide the name, e. g `TONE e'''`

2.4 Reserved words

The following words are reserved and can be used only in their predefined meaning:

ELSE	SHIFTED	PATTERN	INTERVAL
LOGIC	MIDIIN	MIDICHANNEL	MIDIOUT
KEY	TONE	TONESYSTEM	RETUNING
ROOT			

Likewise, the folloging German key words can be used. They are also reserved and must not be used as names.

ANSONSTEN	FORM	HARMONIE	INTERVALL
LOGIK	MIDIIN	MIDIKANAL	MIDIOUT
TASTE	TON	TONSYSTEM	UMSTIMMUNG
WURZEL			

All other words can be used as names or identifiers.

2.5 Numbers

Numbers can be integers or reals written with a decimal point. A notation using decimal exponents is not allowed. Furthermore a number can start with the symbol # and contain further hexadecimal digits. This notation means, that the number will be evaluated as number with basis 16. In the hexadecimal notation no decimal point is allowed, consequently, only integers can be noted in this system.

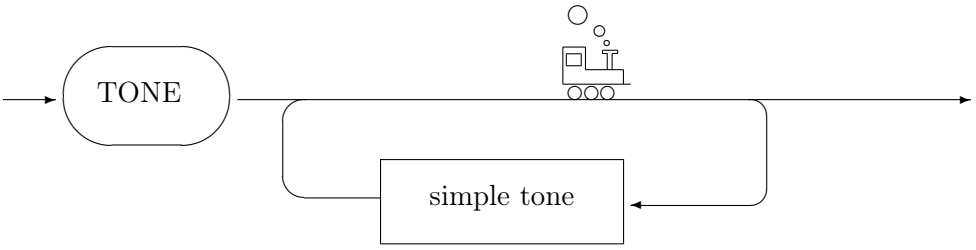
Similar to names, numbers can be denoted in syntax diagrams like of *factor* or *anchor key* in order to emphasize their semantic meaning. Syntactically these are usual numbers.

At certain places only integers are allowed (e. g. as anchor key, width of a tone system,. . .) and at other places also decimal fractions are allowed (tone frequency, interval size,. . .).

3 Syntax description

In the description of the syntax in this book the notions of grammar will be drawen in *in rectangular boxes* and reserved words and characters *in boxes with rounded corners*. The syntax can be derived by going along the arrows in the syntax diagram. Alternatives can bee seen as branching points and repeating parts occure when the diagram contains cycles. An Example:

tone declaration:



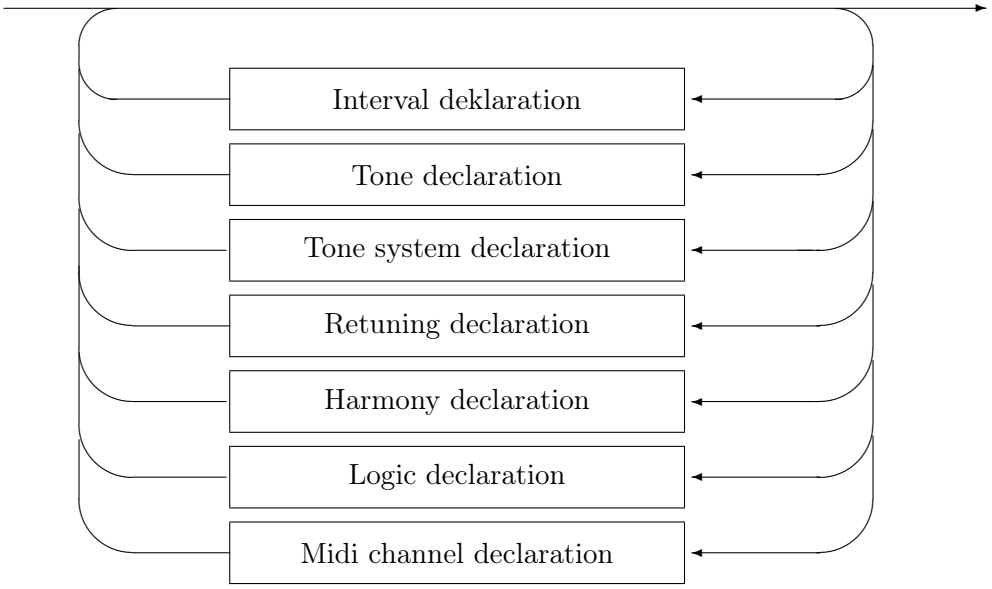
The word *TONE* must be written exactyly as is, while the construct *simple tone* has to be further dissolved. The loop has the effect, that multiple constructs of the form *single tone* can follow each other.

4 The logik program

The logic is the top concept and corresponds to the complete program. The logic program is an arbitrary sequence of declarations. Especially must be emphasized that even an empty file is a permissible program. In this case no logic is present and the 12 tone equal temperament will be used.

Intervals, tones, tone systems, retunings, harmonies, logics und MIDI channels can be declared. The different elements can be declared in an arbitrary order. In particular, elements can be referenced before they are used.

Logic program:



5 Interval

An interval declaration defines individual intervals and the corresponding values. Interval names must not be defined twice, but can have the same name as objects of another kind. The value of an interval ist its frequency proportion. It can be specified in one of the three ways:

- as number ratio,
- as root expression or
- as combination of other intervals.

Here `Number1 : Number2` denotes the ratio

$$\frac{\texttt{Number1}}{\texttt{Number2}},$$

and `A ROOT B` is the `A`th root of `B`:

$$\sqrt[\texttt{A}]{\texttt{B}}.$$

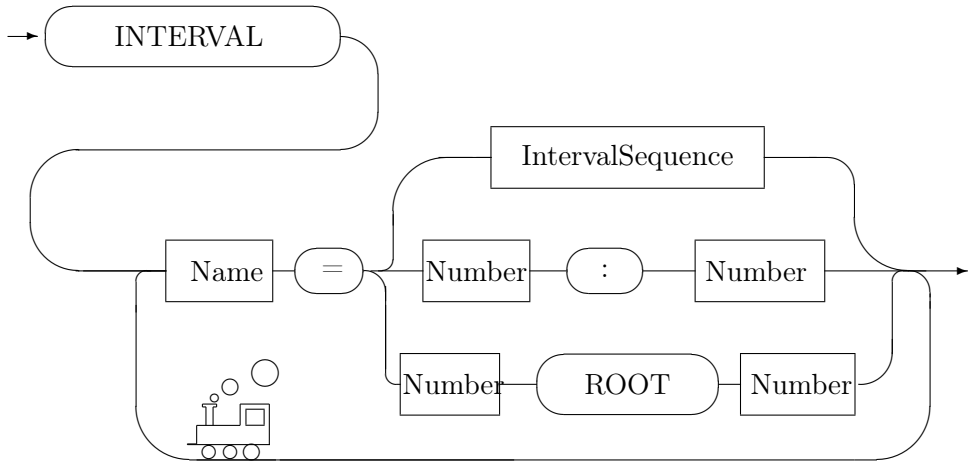
Whenever intervals are combined from other intervals, so these intervals must not have circular dependencies, but the values of the intervals must be computable in a consistent form. Each interval that is used to form another one must be declared in of the same logic program. An interval can optionally be preceeded by a factor. Then the value of this interval will be counted multiple times. Since the intervals are frequency ratios, the value of the value of the combined interval is determined multiplicative. For example, the interval

```
INTERVAL I = 3 octave - 5 third + 17 cent
```

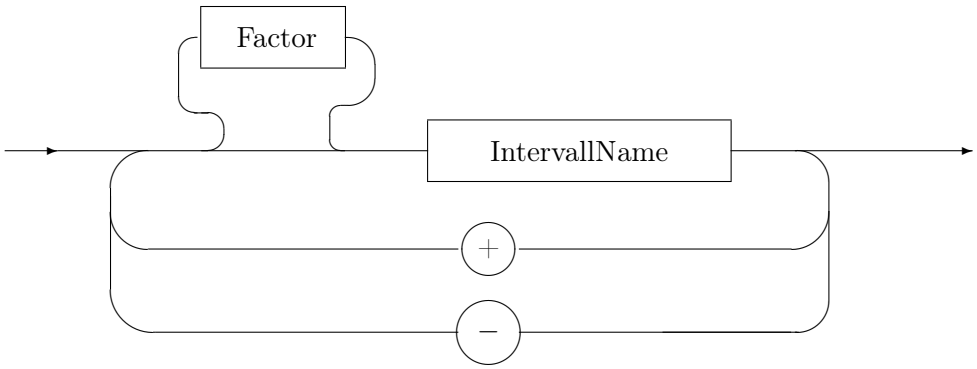
will be calculated according to the following scheme:

$$\texttt{I} = \texttt{octave}^{3} * \texttt{third}^{-5} * \texttt{cent}^{17}.$$

Interval declaration:



IntervallSequence:



6 Tone

A tone declaration defines individual tones and the corresponding values. Tone names must not defined twice, but they can have the same name as object of another kind. The value of the tone is a number, which corresponds to the frequency of the pitch of the tone. It can be a simple number or it can be calculated accordung to a reference tone, which can be moved by given intervals. In the first case the frequency is denoted directly. In the other cases, after the sign of equality a name of a reference tone follows together with a sequence of intervals, which will be multiplied to the reference tone until the requested tone will be reached. In principle, each tone must be defined in the logic program, if it shall be used as a reference tone. Furthermore the definitions of all tones must not contain circular dependencies.

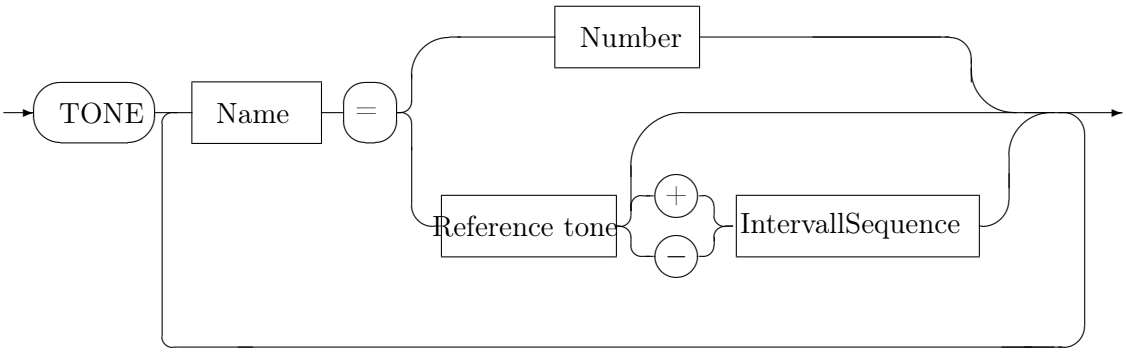
After it has been calculated each tone has a fixed frequency. Correspondingly, the tone name is synonyme for an absolute frequency, whose value will be obtained multiplicatively. Example:

```
TONE  bes = c - 2 fifth + octave
```

will be calculated as

$$\text{frequency}(\texttt{b}) = \text{frequency}(\texttt{c}) * \texttt{fifth}^{-2} * \texttt{octave}.$$

Tone declaration:



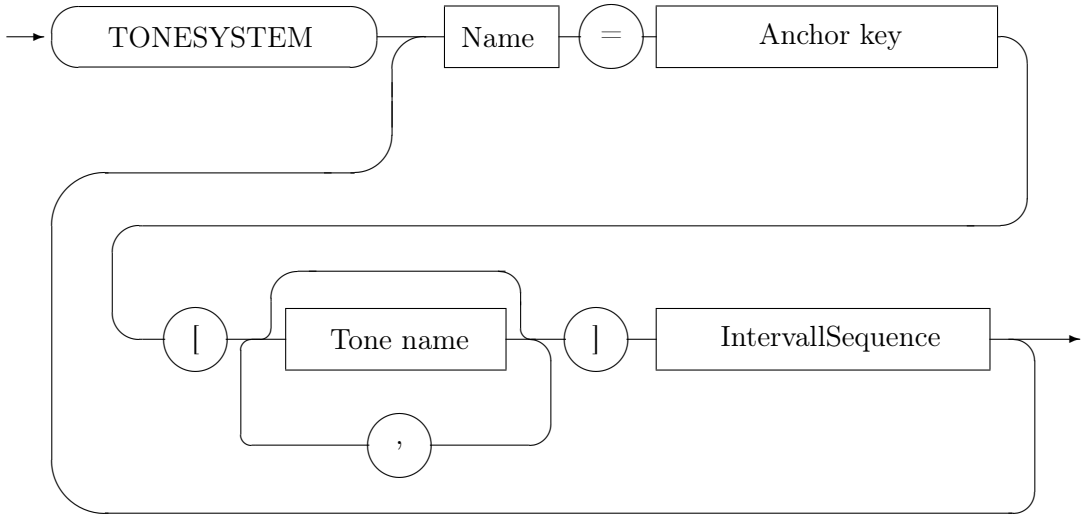
7 Tone system

A tone system declaration defines individual tone systems and the corresponding vlues. Names of tone systems must not be defined twice. However, they can have the same name as objects of another kind. A tone system consists of an anchor key, a set of tones, which is called fundamental scale, and an period interval. The period interval can be combined from other intervals. The fundamental scale consists of a sequence of tones, wich must be eclosed by brackets **[]** and which have to be separated by commas. It corresponds to a range on the piano keyboard. The correspondence of the tone arises from the anchor key. This key coresponds to the MIDI number of the first tone of the fundamental scale.[1] Starting at this position, the tones will be assigned to the piano keys rightwards. The key right of the anchor key corresponds to the second tone of the fundamental scale, etc. . . . After all tones have been assigned to the fundamental scale, the whole fundamental scale is repeted above the assigned keys transposed by the period interval until the border of the claviature is reached.

In the declaration of the fundamental scale several tones cab be omitted, but not the separating commas. The ommited tones cause the effect, that the corresponding keys will produce no sound. Omitted tones will counted as nomal tons in the mapping $pianokey \rightarrow tone$ as if they were present.

[1] The tone c' corresponds to the MIDI number 60.

Tone system declaration

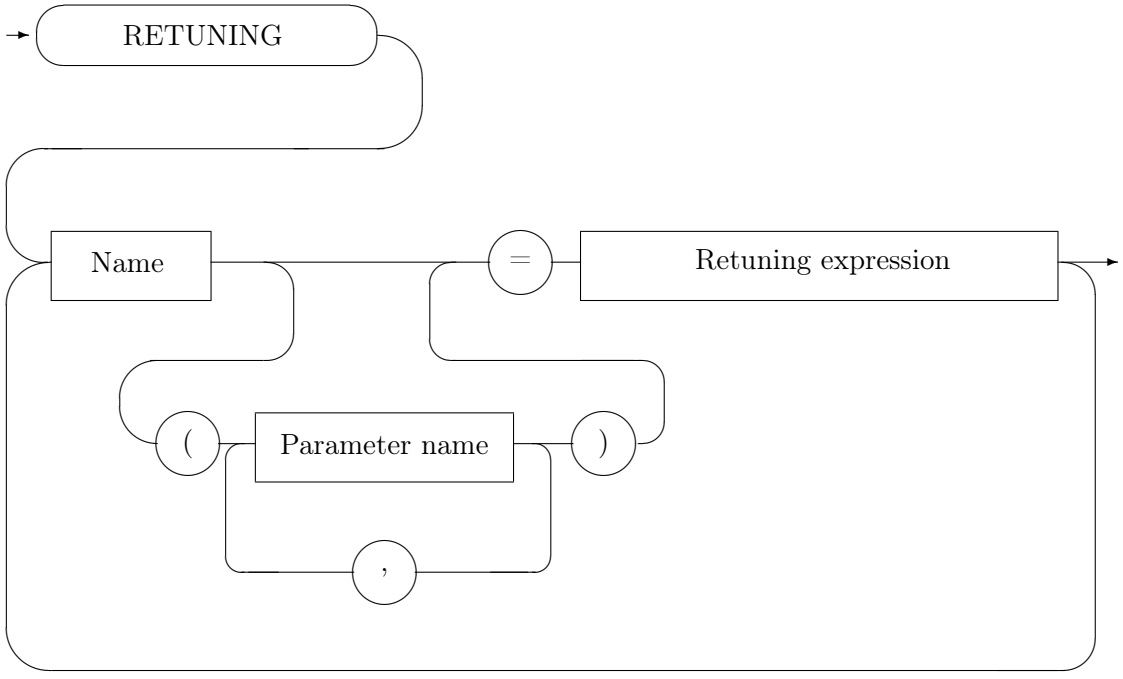


8 Retuning

A retuning declaration defines individual retunings and the corresponding values. Retuning names must not be defined twice, but they can have the same name as objects of another kind. A retuning contains rules, how to change the current tuning. The value that tchanges can be absolute as well as relative to the current state. For an absolute change a number or a symbolic parameter is provided, which denotes the new value regardles of the old one. For a relative change the actual value is used as a reference value and changed by a certain value. The symbol means semantically "*the old value*", which shall be referenced.

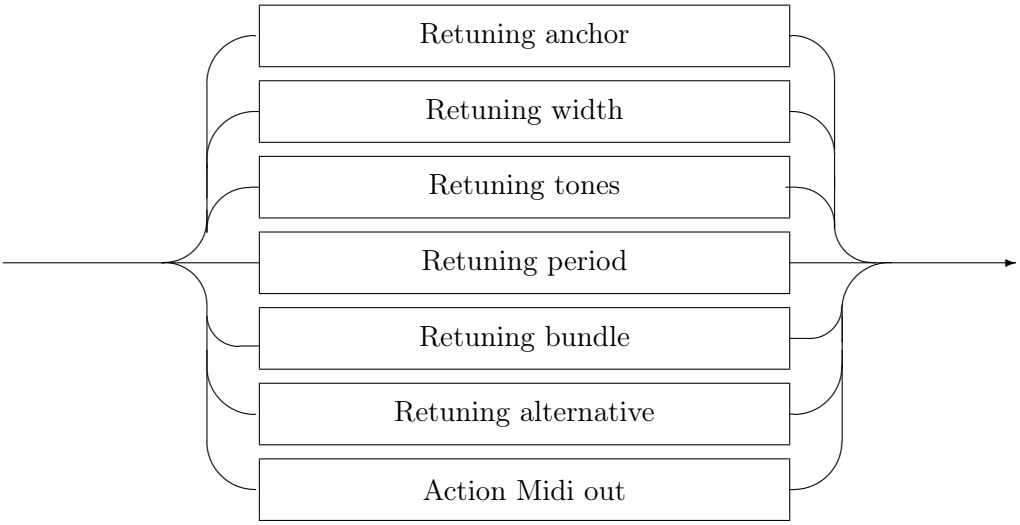
If the retuning shall depend on one or more parameters, a symolic name mustbe provided for each parameter. This praameter name can be used later in the retuning expression.

Retuning declaration:



There are seven kinds of retuning expressions, which change different properties of the actual tuning:

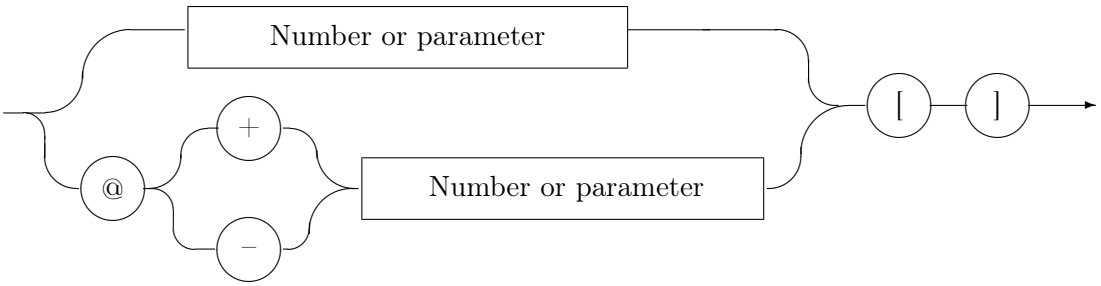
Retuning expression:



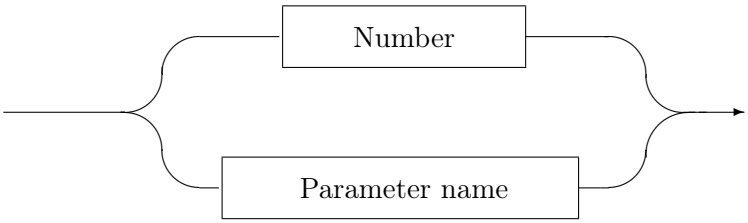
Retuning anchor corresponds to a movement of the fundamental scale. In such a tuning the new anchor key is defined. This new anchor key remains at its frequency and oll other tones are derived in such a way that the complete interval structure will be retained.

In general from a change of the anchor key also a change of the tones results.

Retuning anchor:



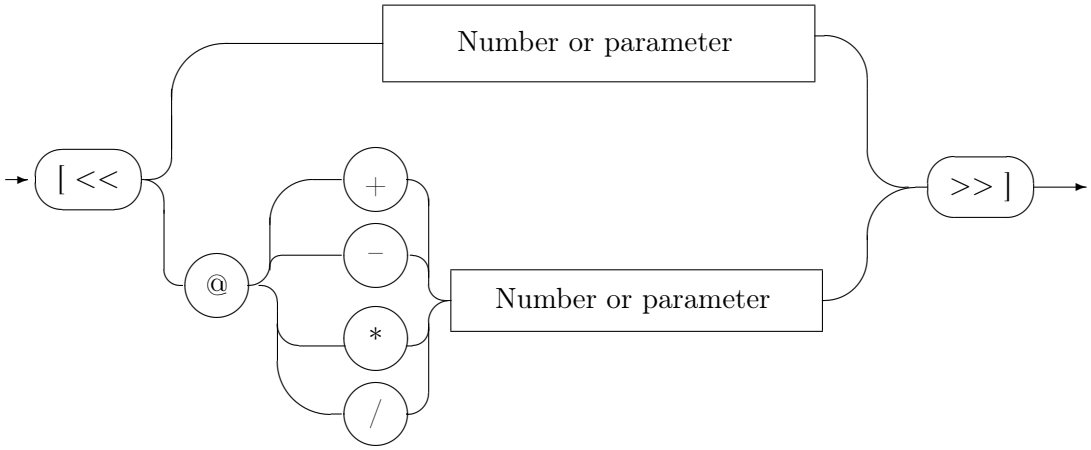
Number or parameter:



Retuning width corresponds to a change in the width of the fundamental scale. The first tone of the fundamental scale remaines at its current value, and all tones right of it remain also at their current state. Then they are counted, how many of them will be needed, to reach the new width of the fundamental scale. These form the new fundamental scale. The interval from the anchor key to the first key after the fundamental scale is treated as the new period interval.

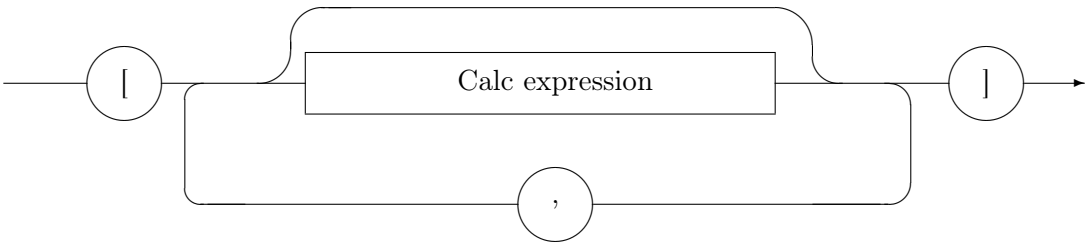
In general, a change of the width leads also to a change of the period interval. A change of the period interval will be ignored, if it leeds to a value smaller than 1. A shortening of the width will be ignered too, if it leads to a width smaller than 1 or to a width greater than 60.

Retuning width:

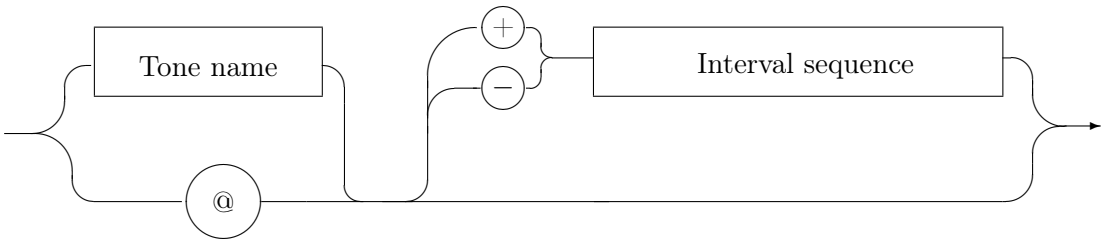


Retuning tones corresponds to a change of individual tones in the fundamental scale. For each tone a calculation expression is provided, which denotes how the tone changes. These expressions are devided by commas and correspond to the tones in the same way, as they are denoted by tone systems. In case of a constant tone the symbol must be used, sine *nothing* means that the tone will be muted. If the number of expressions is smaller than the width of the fundamental scale the remaining tones are not changed. If the width of the fundamental scale is smaller than the number of expressions, the remaining expressions have no effect.

Retuning tones

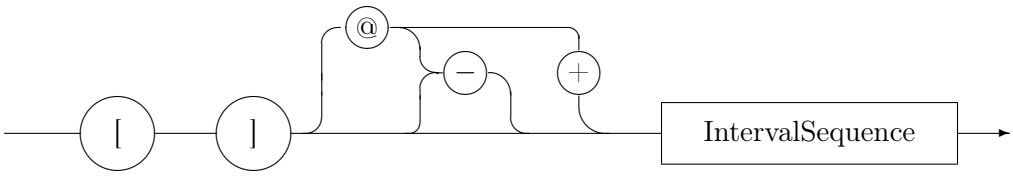


Calc expression:



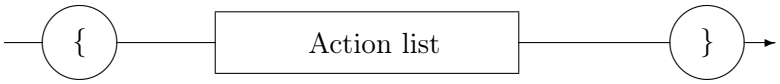
Retuning period corresponds to a change of the period interval. Here, the value of the period interval is changed such that the tones of the fundamental scale remain constant, but all other tones are recalculated, as they arise from the fundamental scale and the period interval.

Retuning period:

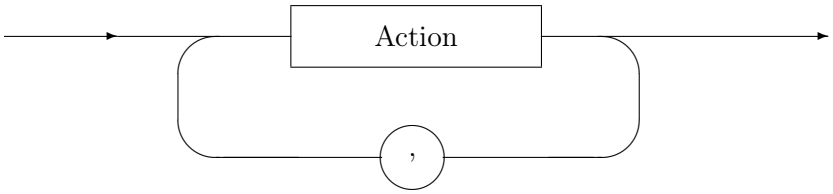


Retuning bundle A retuning bundle contains a sequence of actions. The different actions can call other retunings, tone systems or logics, or they can send MIDI messages. Also here, no circular dependencies are allowed. The individual actions will be executed in the given order. The parameters of a retuning bundle can be passed to the individual calls, as long the action is a retuning an it accepts sufficiently many parameters. The given parameters will be assigned in the same order to the parameters of the called retuning as they are provided.

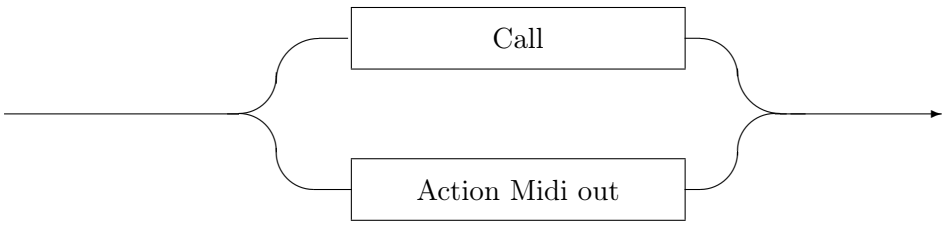
Retuning bundle:



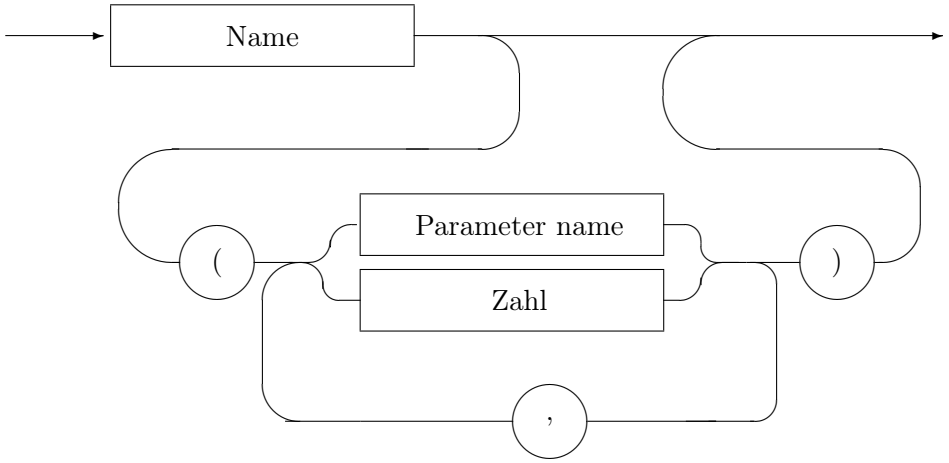
Aktion list:



Action:

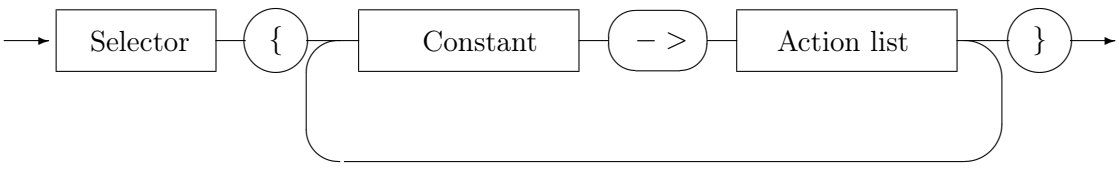


Call:



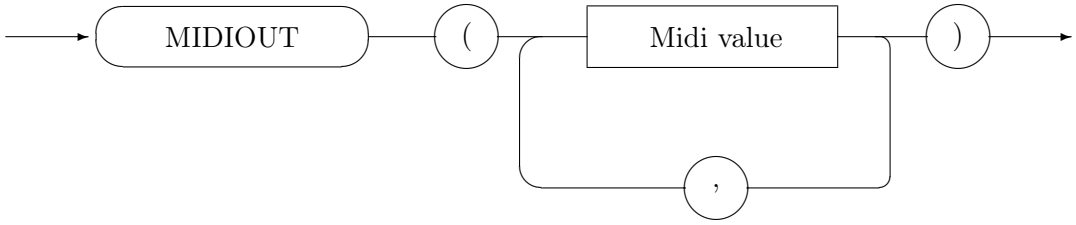
Retuning alternative A retuning alternative allows the selection of one alternative from a set of choices in dependency of a parameter of the retuning (the selector). Each alternative consists of a constant, which determines the case, and a sequence of action, which will be executed if the choice selects this alternative. The individual actions of an alternative are delimited by commas (`,`), while complete alternatives are not seperated by any special delimiter. The constants of the alternatives must be unique integers. At the end of an alternative a special constant `else` can be used. If the selector coinsides with a constant the corresponding alternative is executed. If it is not equals to any constant and the constant `else` is given the alternative of the `else` branch is used. If the selector is different from any constant and the special constant `else` is not given, no alternative will be executed. The call of the retuning has no effect in this case.

Retuning alternative:



Action MIDI out a MIDI output is started with the word `MIDIOUT` and is a sequence of numbers which are seperated by commas and entailed in parentheses. For number the hexadecima writing (#*digits*) is allowed. The numbers bust be in the range from 0 to 255, as they must fit in a bye each. There is no control if the MIDI message is correct or meaningful.

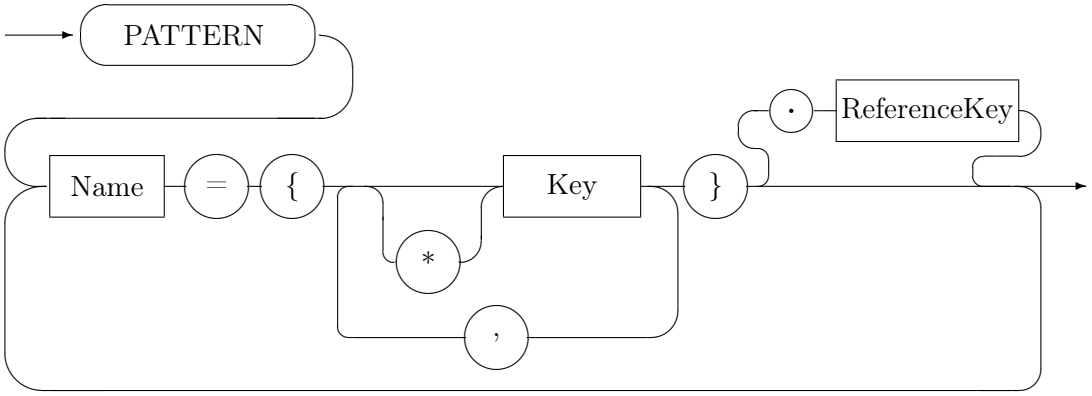
Action Midi out



9 Harmony

A harmony declaration defines individual harmonies and the corresponding values. Harmony names must not be used twice, but they can have the same name as objects of another kind. The value of a harmony is a sequence of key numbers, which identify individual keys within the fundamental scale. Keys on the keyboard, which are outside of the fundamental scale are projected into the fundamental scale and form the projection scale, which has the same width as the fundamental scale, but contains the information which (projected) keys (chromas) are currently held down. A harmony will be recognised, if in the projection scale exactly the given keys are held down. Hereby some keys can be ignored. The corresponding key numbers must be preceeded by an asterisk (*). Optionally, a reference key can be used. It's value doesn't change the pattern for harmnoy recognition, but defines a relative transposition of the reference chroma in the parameter `DISTANCE`. If no reference key is given, its value will be assumed as 0, which means no transposition of the reference chroma.

Harmony Declaration:

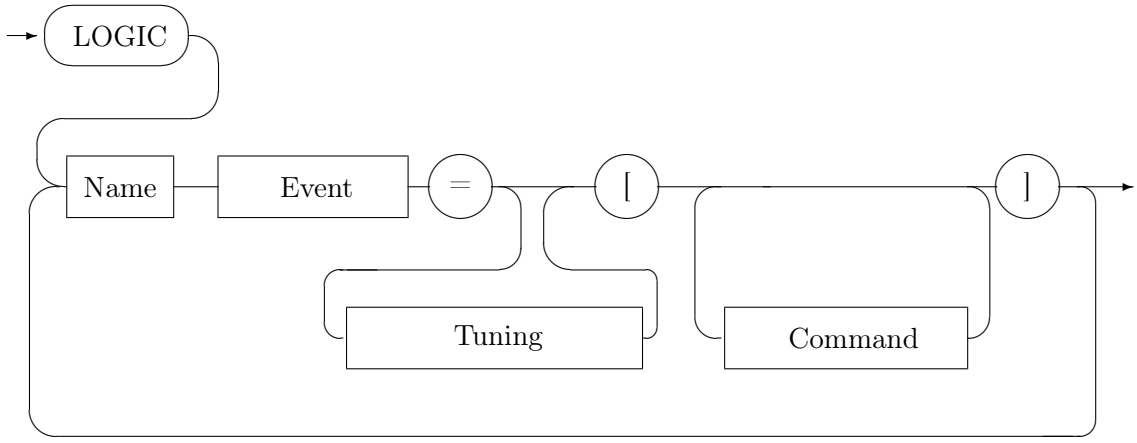


10 Logic

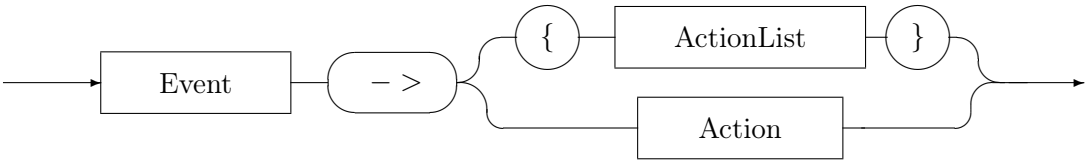
A logic declaration defines individual tuning logics and the corresponding values. Logic names must not be defined twice, but can be the same as names of objects of another kind. Logics are activatied by a trigger and consist of an initial tuning and set of instructions. An initial tuning can be a tone system or a retuning. It can be omitted, in this case the prior tuning will be retained. Similar, the set of instructions can be empty. In this case the actual tuning will be a static tuning that doesn't change automatically. An instruction consists of a trigger and a list of actions. An action can be a call to a retuning, a tone system or a logic, or it can send a MIDI message.

In principle, two sets of triggers exist: The triggers, which activate one of the logics, and the triggers, which select an instruction in the currently active logic. Ther former are called *global triggers* and the latter ones *local triggers*. When an event occurs first the set of global triggers is checked, afterwards the set of local triggers is checked, if the event fits to one of them. In each of these two sets the triggers are tested in the same order as they were declared in the tuning logic. If a global trigger is found, the corresponding logic will be activated. If a local trigger is found, the corresponding action will be called inside the logic. Inside the set of the global triggers the special trigger `ELSE` is not allowed.

Logic declaration



Command:



10.1 Trigger

Each of the 26 latin (ASCII) letters, a harmony, a harmonic form, a MIDI event or the special trigger `ELSE` can be chosen as trigger.

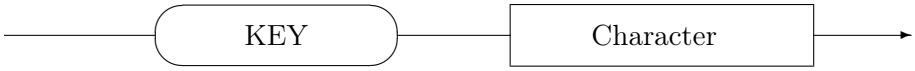
Event:

10.1.1 Keys

This trigger consists of the word `KEY`, followed by one of the 26 latin letters. Whenever inside one of the MUTABOR window the given key is pressed on the computer keyboard, this trigger is matched as long as the window has no special meaning for it. The trigger corresponds always to the MUTABOR box that corresponds to the last active window with an associated box. The first ten boxes can be selected using the number keys Ctrl + 0 to Ctrl + 9 . The instruments/MIDI channels 1–9

can be chosen by the keys 1 to 9 . Thus, only instruments/MIDI channels 1–9 as defined below can be controled by key triggers.

Key event:



10.1.2 Harmonies

Harmony or pattern trigger can recognize either a harmony (pattern) or a harmonic form (shifted pattern). The harmony is called by the name of its pattern and the contained key numbers are treated with respect to the fundamental scale. All pressed keys are projected into the fundamental scale and the resulting key pattern is compared with the harmony. If they coicide the harmony is treated as recognized. Optional keys must be marked by an asterisk (*). They will not be conidered for comparison.

In order to further restrict the recognition of certain harmonys a pre-key can be given. It describes which chroma the lowest pitch must have to be recognized as the given harmony. Equally a post-key can be given which denotes the chroma of the higest tone. If pre-key and post-key are given, both must be matched. The chroma corresponds to the width of the actual fundamental scale.[1] If pre-key and post-key have a chroma that is not contained within the current fundamental scale, this harmony cannot be played as it is contradictory. Such harmonies are „*impossible*“ trigger and will issue a warning during compilation. Practically, they have no effect, since they cannot be true.

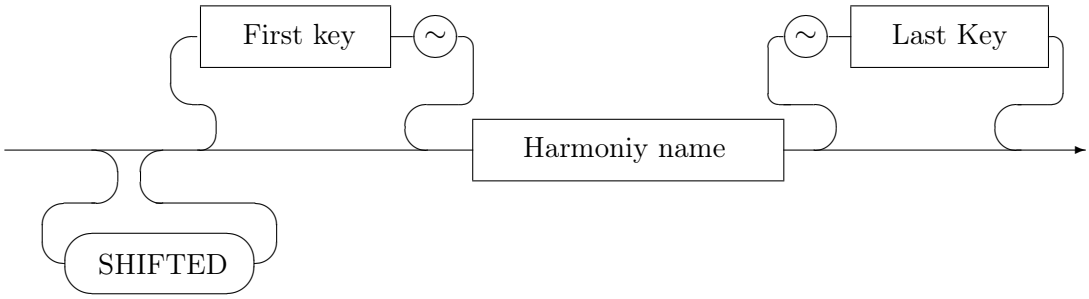
For a harmonic form for the pattern will be checked, if the harmony is true, relatively to each tone of the fundamental scale. If such a shifted harmony is recognised, the trigger is treated as matched and the value `DISTANCE` is set according to the analysis of the harmony.

If for the harmony a reference key has been given, then the harmony analysis will be executed relative to the reference key.

A special form of MIDI triggers is denoted by the keyword `ELSE`. This branch will be executed if the harmony analysis fails to recognise the current pattern. This is **not** influenced by the raw MIDI analysis, which follows afterwards.

[1] This width is not fixed to 12, but can be freely choosen inside wide margins.

Harmony event

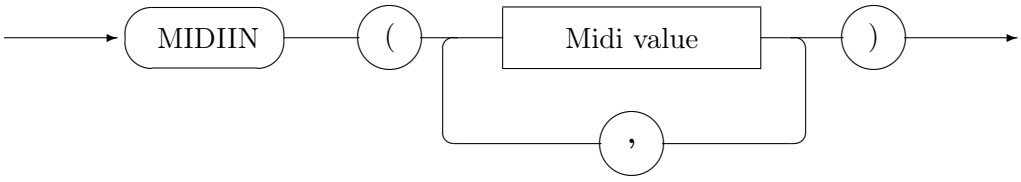


10.1.3 MIDI trigger

MIDI events are started by the word *MIDIIN* and are sequences of numbers contained in parentheses. For the numbers the hexadecimal writing can be used (#*hex-digits*). The first number must be in the range from 128 to 255, since it must be a status byte. All following numbers must be in the range from 0 to 127, which is the range of data bytes. In the status byte the lower four bits (channel number) must be zeros. The channel number will be filled in according to the chosen instrument/MIDI channel such that the MIDI event corresponds to that instrument/MIDI channel.

MIDI bytes are compared messagewise. This leads to the restriction, that the longest MIDI trigger is a MIDI message. Longer MIDI triggers can be realised by chaing logics with the corresponding triggers.

Midi event



10.2 Actions

Actions are triggered inside a logic and can call either a tone system or a retuning, change to another logic or send a MIDI message. On the call of a retuning exactly the same number of parameters must be given in the same order as the called retuning needs. The given parameters are assigned to the parameter names in the same order as they are given in the definition. The parameters can be numbers as well as the value *DISTANCE*. The value DISTANCE corresponds to the result of the last analysis of harmonic forms, where the value of DISTANCE will be retained until the next analysis of harmonic forms.

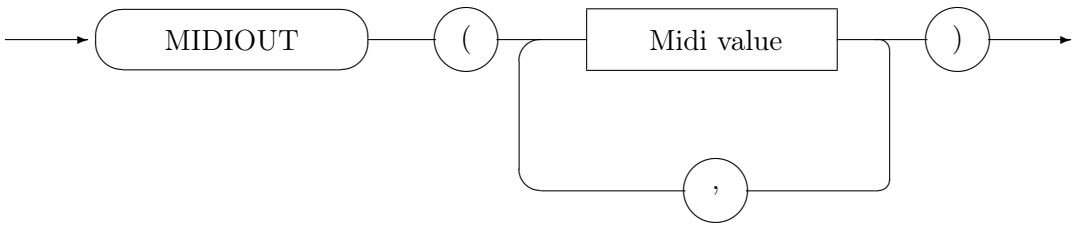
A logic can execute a single action or an action list, which is denoted in braces. In the case of an action list all actions are executed in the given order.

In action lists there is an additional possibility to trigger a new analysis of the current harmony using the current logic and the current tone system. This is done using the keywerd *HARMONY_ANALYSIS*. Such a harmony analysis is allowed only if the tone system or the logic has changed since the last analysis. Otherwise MUTABOR might crash if an action calls itself.

10.2.1 MIDI output

A MIDI output is started with the word *MIDIOUT* and is a sequence of numbers delimited by parentheses. For the numbers the hexadecimal notation (#*hex digits*) can be used. The numbers must be in the range from 0 (#00) to 255 (#FF) as they must be single bytes. There is not check, whether the resulting message is a correct MIDI message.[2]

Action Midi out



[2]Currently (2011-07-03) in each message at most 4 bytes are possible.

11 MIDI channels

The midi channel declaration has been superseded by the routing system introduced in MUTABOR 3.0. The following descriptions is retained for compatibility.

A MIDI channel declaration defines the correspondence, which MIDI channel shall be recognised by MUTABOR, and how much MIDI output channels are assigned to the different input channels. This is imported for the multi 16 processing. The MIDI in channels are called "*instruments*", since for each separate MIDI in channel the corresponding logic program is used exclusively. Up to 16 instruments can play at the same time in different logics.[1] If all 16 instruments/MIDI channels are used and MUTABOR is instructed to use the multi 16 processing, so each instrument can play only a single tone, since only 16 MIDI channels are available as output channels.

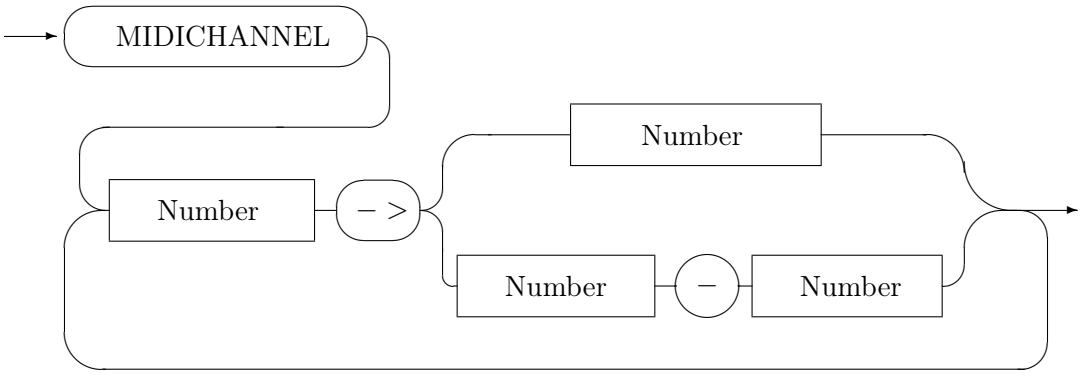
For the multi 16 processing each input channel is assigned a range of output channels. The size of this range corresponds to the number of tones, that can be played simultaneously using this input channel. The channel information of the received status byte defines, for which input channel the message is used. Each input channel can use the complete logic program and works independently from the other input channels. During harmonic analysis or the analysis of a MIDI input message the channel of the message will honoured. On the screen the user can switch between the input channels 1 to 9 by pressing the keys 1 to 9 . A key trigger will be considered with respect to the currently shown input channel. If no MIDI channels have been declared, ind if no channel assignment has been described in the routing panel, the declaration

```
MIDICHANNEL  1  ->  1 - 16
```

will be used automatically.

[1] Since MUTABOR 3.0 this limit can be exceeded by the usage of the routing system.

Midi channel declaration:



11.1 Glossary

key The piano keys are given as distances in half tones to the anchor key. The anchor key has number 0, the next key right of it number 1, the following 2 and so on.

selector The selector of a retuning alternative is the parameter, which chooses the alternative. Generally, this is a parameter (i. e. the parameter name) of the retuning and represents an integer.

letter Here, each of the 26 Latin ASCII letters can be given (no accents or umlauts). If you press the corresponding key on the computer keyboard, the corresponding event will be triggered. The letters are treated case insensitive.

anchor key The anchor key is the piano key, at which the fundamental scale starts and is denoted as MIDI number of the key. The middle c (c') has number 60, the other keys are numbered increasing, and decreasing, respectively, i. e. c#' has number 61, d' number 62 and so on.

constants The constants are integers, which denote at which value of the selector the logic will switch into the following retuning.

names Many calls and assignments are done in MUTABOR using names. In the declaration of each object you must give it a name, which can be used later as reference in other declarations

numbers Numbers can be denoted in MUTABOR by integers or decimals with decimal point.

MIDI values MIDI values are integers between 0 and 2555, which are considered by MIDI devices as commands and data. Additionally, here a hexadecimal notation is allowed, which must be preceded by a leading hash sing (#), e. g. #9C, #A0, #78, #E0, ...

Index