

Table of contents

1	Introduction.....	2
2	Program flow.....	3
	2.1if, elseif, else, endif.....	3
	2.2select, case, default, endsel.....	3
	2.3do, loop.....	4
	2.4do, until.....	4
	2.5while, wend.....	4
	2.6for, to, step, next.....	5
	2.7foreach, in, next.....	5
	2.8break.....	5
3	Math.....	6
	3.1Commands.....	6
	3.2Functions.....	6
4	Strings.....	8
	4.1Functions.....	8
5	Tables.....	9
	5.1Commands.....	9
	5.2Functions.....	9
6	Date and time.....	10
	6.1Commands.....	10
	6.2Functions.....	10
7	Files.....	11
	7.1Commands.....	11
	7.2Functions.....	11
8	Window.....	12
	8.1Commands.....	12
	8.2Functions.....	13
9	Images and drawing.....	14
	9.1Commands.....	14
	9.2Functions.....	16
10	Text output, input and fonts.....	17
	10.1Commands.....	17
	10.2Functions.....	18
11	Audio.....	19
	11.1Commands.....	19
	11.2Functions.....	20
12	Functions.....	20
	12.1Static definition.....	20
	12.2Anonymous definition.....	21
	12.3return.....	21
	12.4this.....	22
	12.5Global variables.....	22
13	Key codes.....	23

1 Introduction

Earlier versions of naalaa was strongly typed and had no coercion at all. In n7, on the other hand, variables may change type at any time, and implicit type conversions are made whenever needed. The n7 value types are numbers and strings, and its reference types are tables (arrays) and functions.

N6:

```
foo = 7
bar# = 3.14
pickle$ = "Hello"
wln bar + float(foo)
wln pickle + ", " + str(foo)
```

N7:

```
foo = 7
bar = 3.14
pickle = "Hello"
wln bar + foo
wln pickle + ", " + foo
```

Output:

```
10.14
Hello, 7
```

Arrays have been replaced with tables in n7. A table consists of key and value pairs. The keys can be numbers or strings, and the values can be numbers, strings, tables or functions. Migrating from n6 arrays to n7 tables shouldn't be much of a problem:

N6:

```
foo[10]
for i = 0 to 9
    foo[i] = rnd(100)
next
```

N7:

```
foo = []
for i = 0 to 9
    foo[i] = rnd(100)
next
```

N7:

```
foo = dim(10)
for i = 0 to 9  foo[i] = rnd(100)
```

N7:

```
foo = []
for i = 0 to 9  foo[sizeof(foo)] = rnd(100)
```

2 Program flow

2.1 if, elseif, else, endif

```
if  $e_1$ 
  <statements1>
[elseif  $e_2$ 
  <statements2>
..
[elseif  $e_n$ 
  <statementsn>
] .. ]
[else
  <statementsdefault>
]
endif
```

If expression e_x evaluates to anything but 0, the statements $statements_x$ are executed, whereafter a jump is made to the first statement following *endif*. If all expressions, $e_1 .. e_n$, evaluates to 0 and *else* is not omitted, the statements $statements_{default}$ are executed.

2.2 select, case, default, endsel

```
select  $e$ 
  case  $c_{1,1}$ [,  $c_{1,2} ..$  [,  $c_{1,n}$ ] .. ]
  <statements1>
  [case  $c_{2,1}$ [,  $c_{2,2} ..$  [,  $c_{2,n}$ ] .. ]
  <statements2>
  ..
  [case  $c_{m,1}$ [,  $c_{m,2} ..$  [,  $c_{m,n}$ ] .. ]
  <statementsm>
] .. ]
```

```
[default
  <statementsdefault>
]
endsel
```

If expression e matches any of $c_{x,1} .. c_{x,n}$, the statements $statements_x$ are executed, whereafter a jump is made to the first statement following *endsel*. If no match is found and *default* is not omitted, the statements $statements_{default}$ are executed.

2.3 do, loop

```
do
  <statements>
loop
```

Execute statements *statements* until *break* is called.

2.4 do, until

```
do
  <statements>
until  $e$ 
```

Execute statements *statements* until expression e evaluates to anything but 0.

2.5 while, wend

```
while  $e$ 
  <statements>
wend
```

Execute statements *statements* as long as expression e doesn't evaluate to 0.

2.6 for, to, step, next

```
for (variable)id = (number)startValue to (number)endValue [step (number)stepSize]  
  <statements>  
next
```

Before each iteration step, variable *id* is assigned a value, starting with *startValue*. When the statements *statements* have been executed, $|stepSize|$ or $-|stepSize|$ is added to *id*, depending on if *endValue* is higher or lower than *startValue*. A step size of 1 is used if *stepSize* is omitted. The iteration stops when $id > endValue$ if $endValue > startValue$, or when $id < endValue$ if $startValue > endValue$.

Note that *startValue*, *endValue* and the optional *stepSize* are evaluated only once – before the first iteration step.

2.7 foreach, in, next

```
foreach [(variable)k,] (variable)v in (table)t  
  <statements>  
next
```

Execute statements *statements* once for every entry in table *t*. In each iteration step, *v* is assigned the value of a table entry and, if not omitted, *k* is assigned the corresponding key. There is no guaranteed order in which the table entries are visited, not even for arrays.

Note that the table is locked during a foreach loop, meaning that any attempt to free any of its entries will cause a runtime error. You may add entries to the table, but such entries won't appear in the current iteration.

2.8 break

```
break
```

Leave a *do*-, *while*-, *for*- or *foreach*-loop.

3 Math

3.1 Commands

randomize (number)seed

Set seed value used by the *rnd* function.

3.2 Functions

(number)**min**((number)x, (number)y)

Returns the lowest number of *x* and *y*.

(number)**max**((number)x, (number)y)

Returns the highest number of *x* and *y*.

(number)**abs**((number)x)

Returns the absolute value of *x*. The following syntax can also be used:

$$y = |x|$$

(number)**floor**((number)x)

Returns the highest integer value that is lower than or equal to *x*.

(number)**ceil**((number)x)

Returns the lowest integer value that is higher than or equal to *x*.

(number)**round**((number)x)

Returns the integer value that is closest to *x*.

(number)**pow**((number)x, (number)y)

Returns x^y . The following syntax can also be used:

$$z = x^y$$

(number)**sqr**((number)x)

Returns the square root of *x*.

(number)cos((number)i)

Returns the cosine of the angle a , expressed in radians.

(number)sin((number)a)

Returns the sine of the angle a , expressed in radians.

(number)tan((number)a)

Returns the tangent of the angle a , expressed in radians.

(number)acos((number)x)

Returns the arc cosine of x in radians.

(number)asin((number)x)

Returns the arc sine of x in radians.

(number)atan((number)x)

Returns the arc tangent of x in radians.

(number)atan2((number)y, (number)x)

Returns the arc tangent in radians of y/x .

(number)rad((number)a)

Returns a , an angle expressed in degrees, converted to radians.

(number)deg((number)a)

Returns a , an angle expressed in radians, converted to degrees.

(number)rnd()

Returns a random number in the range $[0 .. 1]$.

(number)rnd((number)n)

Returns a random integer in the range $[0 .. n - 1]$

(number)rnd((number)minValue, (number)maxValue)

Returns a random integer in the range $[minValue .. maxValue]$

4 Strings

4.1 Functions

(number)**len**((string)*s*)

Returns the number of characters in *s*.

(string)**lower**((string)*s*)

Returns *s* converted to lowercase.

(string)**upper**((string)*s*)

Returns *s* converted to uppercase.

(string)**left**((string)*s*, (number)*n*)

Returns the *n* leftmost characters of *s*.

(string)**right**((string)*s*, (number)*n*)

Returns the opposite part of the string compared to *left*.

(string)**mid**((string)*s*, (number)*n*[, (number)*m*])

Returns *m* characters, starting at character index *n*, of *s*. If *m* is omitted, only one character is returned.

(array)**split**((string)*s*, (string)*sub*)

Returns an array with all substrings of *s* split at every occurrence of *sub* (*sub* is excluded from the resulting strings).

(number)**instr**((string)*s*, (string)*sub*[, (number)*n*])

Returns the character index in *s* of the first occurrence of *sub*. The search starts at character index *n*, which is 0 if omitted. -1 is returned if no occurrence was found.

(string)**replace**((string)*s*, (string)*sub*, (string)*rep*[, (number)*n*])

Returns a string where all occurrences of *sub* in *s* are replaced with *rep* if *n* is omitted. If *n* is not omitted, the search starts at the character index *n* and only the first occurrence is replaced.

(string)**chr**((number)*c*)

Returns the character represented by the ASCII code *c*.

(number)**asc**((string)*s*)

Returns the ASCII code of the first character in *s*.

5 Tables

5.1 Commands

free key (table)*t*, (string/number)*k*

Removes the entry with key *k* from table *t* if it exists. If the key is numeric, any sequent numeric keys will be decreased.

free val (table)*t*, *v*

Removes every entry with *v* as value from table *t*. If entries with numeric keys are removed from a sequence, any sequent keys will be re-indexed.

free *v*

Removes a variable *v* from its parent, which can be a table, program memory or a function's local memory.

```
free foo.x
```

gives the same result as:

```
free foo, "x"
```

No form of re-indexing occurs if you use *free* with a numeric index:

```
free foo[1]
```

clear (table)*t*

Clears table *t*.

5.2 Functions

(table)**dim**((number)*size*₁ [, (number)*size*₂ .. [, (number)*size*_{*n*} ..])

Returns an array with one or more dimensions of the specified sizes, *size*₁ .. *size*_{*n*}.

(table)fill(*v*, (number)*size*₁[, (number)*size*₂ .. [, (number)*size*_{*n*} ..])

Returns an array with one or more dimensions of the specified sizes, *size*₁ .. *size*_{*n*}, with every element set to a deep copy of *v*.

(number)sizeof((table)*t*)

Returns the number of elements in table *t*.

(table)copy((table)*t*)

Returns a deep copy of table *t*.

6 Date and time

6.1 Commands

wait (number)*ms*

Wait for *ms* milliseconds.

fwait (number)*fps*

Wait until at least 1000/*fps* milliseconds have passed since the previous call to *fwait*. You can use this at the end of a game loop to cap the number of frames per second to *fps*.

6.2 Functions

(number)clock()

Returns time in milliseconds since program execution started.

(number)time([(number)*year*[, (number)*month*[, (number)*day*[, (number)*hour*[, (number)*minute*[, (number)*second*]]]]])

Returns time in seconds since epoch for specified date and time. If all arguments are omitted, the current date and time is used.

(table)datetime([(number)*t*])

Returns a table with information about the time *t*, specified in seconds since epoch. If *t* is omitted, the current date and time is used. The fields of the table are:

year Ex: 2021

month	1..12, where 1 is January
day	Day of month, [1..31]
hour	[0..23]
minute	[0..59]
second	[0..59]
wday	[1..7], where 1 is Monday
yday	[1..365], where 1 is January 1

(number)fwait((number)fps)

Wait until atleast $1000/fps$ milliseconds have passed since the previous call to *fwait* and return 1 if the function actually *had* to wait. In a game loop, you can use the return value to determine if you should skip drawing during the next frame.

7 Files

7.1 Commands

open file (number)*id*, (string)*filename*

Open file *filename* for reading through identifier *id*.

create file (number)*id*, (string)*filename*

Create file *filename* for writing through identifier *id*.

free file (number)*id*

Close file *id*.

write file (number)*id*, (string)*s*

Write *s* to file *id*.

wln file (number)*id*, (string)*s*

Write *s* followed by a line break to file *id*.

7.2 Functions

(number)openfile((string)*filename*)

Same as the command *open file* but auto-generates and returns an identifier.

(number)createfile((string)*filename*)

Same as the command *create file* but auto-generates and returns an identifier.

(number)file((number)*id*)

Returns 1 if file *id* has been opened/created or 0 if it hasn't.

(string)hread((number)*id*)

Reads characters from file *id* until any form of whitespace or end of file is reached and returns the characters as a string. If no characters could be read an unset variable is returned.

(string)frln((number)*id*)

Reads characters from file *id* until a new line or end of file is reached and returns the characters as a string. If no characters could be read an unset variable is returned.

(number)hreadc((number)*id*)

Reads a character from file *id* and returns it as a number (use *chr* to convert it to a string). If no character could be read an unset variable is returned.

(string)openfiledialog([(string)*extension*])

Shows a system dialog for opening a file. Set the optional *extension* to a file extension (such as "txt") to only list files of that type. If the user selects a file, its full path is returned, else an empty string is returned.

(string)savefiledialog([(string)*extension*])

Same as *openfiledialog*, but shows a system dialog for saving a file.

(number)exists((string)*filename*)

Returns 1 if a file with the name *filename* exists or 0 if it doesn't.

8 Window

8.1 Commands

set window (string)*title*, (number)*w*, (number)*h*[, (number)*fullScreen*[, (number)*scaleFactor*]]

Create a window with the width *w*, height *h* and title *title*. If the optional *fullScreen* is anything but 0, the window will be stretched to cover the entire screen. *scaleFactor* can be set to an integer value higher than 1 to scale the window if *fullScreen* is 0.

Note that multiple windows are not supported. If you call *set window* more than once, the previous window will be replaced. Also, in earlier versions of naalaa, a default window was created when the program started. But in n7, no window is created until you call *set window*. The program is automatically terminated if the user closes the window.

set redraw (number)*v*

Disable automatic redraw if *v* is 0, else enable it.

By default, the window content is updated every time you use a command that draws something, such as *wln* or *draw image*. When this behavior is disabled, you manually have to call *redraw* to update the window content.

redraw

Update the window content.

set mouse (number)*value*

Set mouse cursor visibility to *value*, where 0 means invisible and anything else visible.

set mouse (number)*x*, (number)*y*

Set mouse position to (*x*, *y*).

8.2 Functions

(number)**window**((string)*title*)

Returns 1 if there is any running n7 program with the specified title.

(number)**keydown**((number)*keyCode*[, (number)*unflag*])

Returns 1 if key *keyCode* is being pressed. If the optional *unflag* is anything but 0, the function won't return 1 for the key until it has been released and pressed again. See *Key codes* for a lists all available key codes.

(number)**inkey**()

Returns the ASCII code of any printable character that was processed during the last *wait* call. The input is queued, and the function returns 0 when the queue is empty. You can use *inkey* to implement custom text input. For game controls, use *keydown*.

(number)mousex()

Returns the window x coordinate of the mouse cursor.

(number)mousey()

Returns the window y coordinate of the mouse cursor.

(number)mousebutton((number)button[, unflag])

Returns a non-zero number if mouse button *button* is being pressed. If the optional *unflag* is anything but 0, the function will return 0 for the button until it has been released and pressed again. The only valid *button* values are 0 for the left mouse button, 1 for the right button and 2 for the mousewheel.

9 Images and drawing

Window and image coordinates are always specified from the top left corner.

9.1 Commands

set color (number)r, (number)g, (number)b[, (number)a]

Set color in RGBA format, where *r*, *g*, *b* and *a* are the red, green, blue and alpha intensities, all in the range [0 .. 255]. If omitted, *a* is assumed to be 255. This affects all future draw commands.

set color (table)c

Set color from an array *c*, [*r*, *g*, *b*] or [*r*, *g*, *b*, *a*], where *r*, *g*, *b* and *a* are the red, green, blue and alpha intensities in the range [0 .. 255]. If the array only has three elements, 255 is used for alpha.

set additive (number)value

Turn additive draw mode on if *value* is anything but 0, else off.

set clip rect (number)x, (number)y, (number)w, (number)h

Set the top left corner of the clipping rectangle to (*x*, *y*) and its width and height to *w*, *h*. This affects all future draw commands.

clear clip rect

Restore the clipping rectangle to the entire area of the destination image (usually the window's back buffer).

cls

Clear the window content.

draw pixel (number)x, (number)y

Draw a pixel at position (x, y) .

set pixel (number)x, (number)y

Set the pixel at position (x, y) . The difference between *set pixel* and *draw pixel* is that *set pixel* SETS the alpha component of the pixel, while *draw pixel* uses the alpha for blending.

draw line (number)x1, (number)y1, (number)x2, (number)y2

Draw a line between $(x1, y1)$ and $(x2, y2)$.

draw rect (number)x, (number)y, (number)w, (number)h[, (number)fill]

Draw a rectangle with its top left corner at (x, y) and the width and height w, h . If *fill* is set to anything but 0 (default), the rectangle is filled.

draw ellipse (number)x, (number)y, (number)rx, (number)ry[, (number)fill]

Draw an ellipse with its center at (x, y) and x and y radiuses rx and ry . If *fill* is set to anything but 0 (default), the ellipse is filled.

draw polygon (array)points[, (number)fill[, (number)pointCount]]

Draw a polygon based on the coordinates stored in the array *points*, with the format $[x_0, y_0, x_1, y_1 \dots x_n, y_n]$. If *fill* is set to anything but 0 (default), the polygon is filled. Use the optional *pointCount* if you want to draw a polygon based on only the first *pointCount* coordinate pairs in the array.

load image (number)id, (string)filename[, (number)cols, (number)rows]

Load image *id* from file *filename*. Optionally divides the image into *cols* columns and *rows* rows (see *set image grid*).

set image grid (number)id, (number)cols, (number)rows

Divide the image *id* into *cols* columns and *rows* rows. This doesn't change the image in any way, but allows you to use a special version of *draw image* to draw a specific cell of the grid. The cells are indexed from left to right and top to bottom starting with 0.

```
set image grid myImage, 4, 2
```

, would divide an image into the cells:

0	1	2	3
4	5	6	7

set image colorkey (number)*id*, (number)*r*, (number)*g*, (number)*b*

Make color *r*, *g*, *b* completely transparent in image *id*.

create image (number)*id*, (number)*w*, (number)*h*

Create image *id* and set its width and height to *w* and *h*.

free image *id*

Remove the image *id* from memory. All images are automatically freed when the program ends.

set image (number)*id*

Set the destination image for all draw commands (including text output) to *id*. Use the constant *primary* as *id* to direct drawing back to the window.

draw image (number)*id*, (number)*x*, (number)*y*

Draw image *id* at position (*x*, *y*).

draw image (number)*id*, (number)*x*, (number)*y*, (number)*c*

Draw cell (see set image grid) *c* of image *id* at position (*x*, *y*).

draw image (number)*id*, (number)*x*, (number)*y*, (number)*srcX*, (number)*srcY*, (number)*w*, (number)*h*

Draw a rectangular part of image *id*, defined by the position (*srcX*, *srcY*) and the width and height *w* and *h*, at position (*x*, *y*).

draw vraster (number)*id*, (number)*x*, (number)*y1*, (number)*y2*, (number)*u1*, (number)*v1*, (number)*u2*, (number)*v2*

Draw a vertical line from (*x*, *y1*) to (*x*, *y2*) using image *id* as texture. The texture coordinates (*u1*, *v1*, *u2* and *v2*) should be in the range [0..1].

The current drawing color (set with *set color*) affects *draw vraster* and *draw hraster* (below) differently than the other drawing commands. The RGBA color is applied as a "fog" effect, where an alpha value of 0 means no fog (texture is drawn with its original color) and 255 means full fog (texture color is completely replaced with fog color).

draw hraster (number)*id*, (number)*y*, (number)*x1*, (number)*x2*, (number)*u1*, (number)*v1*, (number)*u2*, (number)*v2*

Draw a horizontal line from (*x1*, *y*) to (*x2*, *y*) using image *id* as texture. The texture coordinates (*u1*, *v1*, *u2* and *v2*) should be in the range [0..1].

9.2 Functions

(number)**loadimage**(*filename*[, (number)*cols*, (number)*rows*)

Same as the command *load image* but auto-generates and returns an identifier.

(number)**createimage**((number)*w*, (number)*h*)

Same as the command *create image* but auto-generates and returns an identifier.

(number)**image**((number)*id*)

Returns 1 if the image *id* exists (has been loaded or created) or 0 if it doesn't.

(number)**width**([(number)*id*])

Returns the width, or cell width if an image grid has been set up, of image *id* or the destination image if *id* is omitted .

(number)**height**([(number)*id*])

Returns the height, or cell height if an image grid has been set up, of image *id* or the destination image if *id* is omitted.

(array)**pixel**((number)*x*, (number)*y*)

Returns the color at position (*x*, *y*) as an array [*r*, *g*, *b*, *a*] with RGBA intensities in the range [0 .. 255].

10 Text output, input and fonts

You can only use fonts, caret positioning and text justification when writing text to a window with the commands *wln*, *write* and *center*. If you use *wln* or *write* without having created a window, the output will be directed to the console. When a window has been created you can still output text to the console with *pln* (useful for debugging).

10.1 Commands

println (string)*s*

Write string *s* to the console and move to a new line.

set caret (number)*x*, (number)*y*

Set position of text output/input to (*x*, *y*). The actual caret is only visible during text input with the *rln* function.

set justification left/right/center/(number)*v*

Set justification of text output/input to left, right or centered either using the keywords *left*, *right* and *center* or a value *v*, where 0 represents centered, any negative number is left and any positive number right.

wln (string)*s*

Write string *s* and move to a new line.

write (string)*s*

Write string *s* without moving to a new line.

center (string)*s*

Write string *s* centered and move to a new line (same as calling *wln* after *set justification center*).

create font (number)*id*, (string)*name*, (number)*size*[, (number)*bold*[, (number)*italic*[, (number)*underline*[, (number)*smooth*]]]]]

Create a *bitmap* font with the identifier *id* from a system font named *name* (ex. "arial"). Use *size* to set the font's height in pixels. If *bold*, *italic* and/or *underline* is anything but 0 the font will be bold, italic and/or underlined. If *smooth* is anything but 0, an antialiasing filter is applied to the created bitmap font.

save font (number)*id*, (string)*filename*

Save created/loaded bitmap font *id* as *filename.txt* and *filename.png*.

load font (number)*id*, (string)*filename*

Load bitmap font *id* from *filename.txt* and *filename.png*.

set font (number)*id*

Set font used by *wln*, *write*, *center* and *rln* to *id*.

10.2 Functions

(string/number) **rln**([*maxChars*[, (number)*type*]])

Waits for and returns keyboard input from the user. If *maxChars* is higher than 0, the number of characters is limited to its value. If *type* is set to *TYPE_NUMBER*, only numeric input is allowed and a number is returned. By default *maxChars* is 0 (unlimited number of characters) and *type* is *TYPE_STRING*. Currently, the only valid *type* values are *TYPE_NUMBER* and *TYPE_STRING*.

If no window has been created, input is read from the console. In this case *maxChars* has no effect.

(number)**createfont**((string)*name*, (number)*size*[, (number)*bold*[, (number)*italic*[, (number)*underline*[, (number)*smooth*]]]])

Same as the command *create font* but auto-generates and returns an identifier.

(number)**loadfont**((string)*filename*)

Same as the command *load font* but auto-generates and returns an identifier.

(number)**font**((number)*id*)

Returns 1 if font *id* exists (has been created or loaded) or 0 if it doesn't.

(number)**fwidth**([(number)*id*,] (string)*s*)

Returns the width of string *s* if printed with font *id* or the currently used font if *id* is omitted.

(number)**fheight**([(number)*id*])

Returns the height of font *id* or the currently used font if *id* is omitted.

11 Zones

Zones can be used for button logic.

11.1 Commands

create zone (number)*id*, (number)*x*, (number)*y*, (number)*w*, (number)*h*

Create zone *id* and set its position to (*x*, *y*) and its width and height to *w* and *h*.

free zone (number)*id*

Free zone *id*. All images are automatically freed when the program ends.

11.2 Functions

(number)createzone((number)x, (number)y, (number)w, (number)h)

Same as the command *create zone* but auto-generates and returns an identifier.

(number)zone()

Returns the id of the last zone that had a valid click. Each click is returned only once.

(number)zone((number)id)

Returns the status of zone *id*. These are the different status values:

- 0 The mouse cursor is not over the zone
- 1 The mouse cursor is over the zone but the left mouse button is not pressed
- 2 The mouse cursor is over the zone and the left mouse button is pressed

Note that if a click has been *initiated* for a zone (mouse button down while being over it), no other zone will have status 1 until the mouse button has been released. This is simply how buttons usually work.

(number)zone((number)x, (number)y)

Returns the id of a zone at the position (*x*, *y*).

(number)zonex((number)id)

Returns the x coordinate of zone *id*.

(number)zoney((number)id)

Returns the y coordinate of zone *id*.

(number)zonew((number)id)

Returns the width of zone *id*.

(number)zoneh((number)id)

Returns the height of zone *id*.

12 Audio

Currently only wav files in PCM format are supported for sound and music.

12.1 Commands

load sound (number)*id*, (string)*filename*

Load sound *id* from file *filename*.

free sound (number)*id*

Remove the sound *id* from memory. All sounds are automatically freed when the program ends.

play sound (number)*id*[, (number)*volume*[, (number)*panning*]]

Play sound *id* with volume *volume*, [0..1], and panning *panning*, [-1..1], where -1 is left and 1 is right. The default values for *volume* and *panning* are 1 and 0.5.

load music (number)*id*, (string)*filename*

Load music *id* from file *filename*.

free music (number)*id*

Remove the music *id* from memory. All music is automatically freed when the program ends.

play music (number)*id*[, (number)*loop*]

Play music *id*. If *loop* is not omitted and set to anything but 0, the music will loop.

stop music (number)*id*

Stop playing music *id*.

set music volume (number)*id*, (number)*volume*

Set the volume of music *id* to *volume*, [0..1]. The default *volume* value for a piece of music is 1.

12.2 Functions

(number)**loadsound**((string)*filename*)

Same as the command *load sound* but auto-generates and returns an identifier.

```
(number)sound((number)id)
```

Returns 1 if the sound *id* exists (has been loaded) or 0 if it doesn't.

```
(number)loadmusic((string)filename)
```

Same as the command *load music* but auto-generates and returns an identifier.

```
(number)music((number)id)
```

Returns 1 if the music *id* exists (has been loaded) or 0 if it doesn't.

13 Functions

13.1 Static definition

```
function functionName([parameter1[, parameter2 .. [, parametern] .. ]])
```

```
<statements>
```

```
endfunc
```

Create a static function with the name *functionName* and an optional list of parameters. Call a function through its name followed by a list of comma-separated arguments, matching the parameter list in the function definition, within parentheses. You have to use parentheses even if the function expects no arguments.

When you call a function, its statements are executed. When the statements have been executed, or a *return* statement is reached (see *return*), program execution continues from where the function was called.

Variables defined inside a function (including the parameters) only exist while the function is executing. Unless a variable in the main program has been declared as *visible* or *constant* (see *Global variables*) a function can't see or access it.

```
' Define a function with one parameter.
function MyFunction(aParameter)
    pln "The parameter is " + aParameter
endfunc
' Call MyFunction
MyFunction("Foobar")
```

13.2 Anonymous definition

```
(function)function([parameter1[, parameter2..[, parametern] .. ]])  
<statements>  
endfunc
```

Returns an anonymous function with an optional list of parameters.

```
' Assign an anonymous function to a variable named foo.  
foo = function(aParameter)  
    pln "The parameter is " + aParameter  
endfunc  
' Call the function through foo.  
foo("Foobar")
```

13.3 return

```
return [v]
```

Leave the function and optionally return a value that may be captured by the caller.

```
' Define a function with two parameters.  
function Multiply(x, y)  
    ' Return product.  
    return x*y  
endfunc  
' Capture result in a variable named result and print it.  
result = Multiply(10, 3)  
pln result  
' Print the result of another call directly.  
pln Multiply(5, 5)
```

13.4 this

If a function is called through a table field, you can access the table inside the function using *this*.

```
' Create a table and add a variable named meaningOfLife to it.
```

```
foo = []
foo.meaningOfLife = 42
' Add a function named Print.
foo.Print = function()
    ' Use this to access the variable meaningOfLife.
    pln "The meaning of life is " + this.meaningOfLife
endfunc
' Call foo's function.
foo.Print()
```

13.5 Global variables

visible $var_1 [= e_1][, var_2 [= e_2] .. [, var_n [= e_n]] ..]$

constant $const_1 = e_1[, const_2 = e_2 .. [, const_n = e_n] ..]$

14 Key codes

These key codes should be used with the keydown function, since the actual values are platform dependent:

KEY_TAB
KEY_RETURN
KEY_SHIFT
KEY_CONTROL
KEY_ESCAPE
KEY_SPACE
KEY_PAGE_UP
KEY_PAGE_DOWN
KEY_END
KEY_HOME
KEY_LEFT
KEY_UP
KEY_RIGHT
KEY_DOWN
KEY_INSERT
KEY_DELETE
KEY_0
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6
KEY_7
KEY_8
KEY_9
KEY_A
KEY_B
KEY_C
KEY_D
KEY_E
KEY_F

KEY_G
KEY_H
KEY_I
KEY_J
KEY_K
KEY_L
KEY_M
KEY_N
KEY_O
KEY_P
KEY_Q
KEY_R
KEY_S
KEY_T
KEY_U
KEY_V
KEY_W
KEY_X
KEY_Y
KEY_Z
KEY_MULTIPLY
KEY_ADD
KEY_SEPARATOR
KEY_SUBTRACT
KEY_DIVIDE
KEY_F1
KEY_F2
KEY_F3
KEY_F4
KEY_F5
KEY_F6
KEY_F7
KEY_F8
KEY_F9
KEY_F10

KEY_F11

KEY_F12