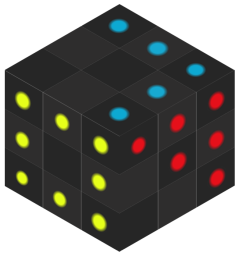

Rubik's®
FUTURO CUBE
Motion controlled gaming cube, lots of challenges in your pocket

PAWN - embedded scripting language

# Rubik's Futuro Cube – SDK
# document version 1.4 OPEN BETA



www.futurocube.com

# 1 Licenses

## 1.1 Seven Towns Ltd.



## 1.2 Pawn

PAWN is free and it is published under the Apache License 2.0, plus an exception clause to explicitly permit static linking of the library to commercial applications. More information about PAWN, language syntax and other info can be found at:

- http://www.compuphase.com/pawn/pawn.htm

## 1.3 Exception clause to the Apache License version 2.0

As a special exception to the Apache License 2.0 (and referring to the definitions in Section 1 of that license), you may link, statically or dynamically, the "Work" to other modules to produce an executable file containing portions of the "Work", and distribute that executable file in "Object" form under the terms of your choice, without any of the additional requirements listed in Section 4 of the Apache License 2.0. This exception applies only to redistributions in "Object" form (not "Source" form) and only if no modifications have been made to the "Work".

# Contents

# 2 Introduction

Welcome to the world of programming Rubik's Futuro Cube. We hope you enjoy it as much as we were enjoying creating this development kit. Rubik's Futuro Cube can be programmed by simple, typeless 32-bit language called "PAWN". This language has C-like syntax and even if simple is very powerful and fast. To get little overview what is possible on basic level and how the language looks like, please have a look at available examples at our SDK page

- http://www.futurocube.com/sdk.htm

More information about PAWN, language syntax and other info can be found at

- http://www.compuphase.com/pawn/pawn.htm

For language syntax guide look at

- http://www.compuphase.com/pawn/Pawn_Language_Guide.pdf

You will also need to download Rubik's Futuro Cube Suite (RFC Suite), available for Windows and Mac OS. Windows users additionally have to correctly install USB driver. Please follow instructions at

- http://www.futurocube.com/support.htm

SDK will need some testing and we will immediately react on reported bugs. Do not hesitate to write us either using contact form on our web or email us directly: info@futurocube.com

# 3 Hello world

In order to access shell commands and compiling ability, once RFC Suite is started, switch in menu to SDK mode. Rubik's Futuro Suite contains standard PAWN compiler and allows to compile scripts automatically with preset recommended stack and optimization settings. (can be adjusted by user). Easiest way to start is to place library "futurocube.inc" at the same directory, where is your script and than just select the script in Suite and try to compile it. It automatically searches for libraries in the same folder. Rubik's Futuro Suite does not contain any script editor. Any text editor can be used.

As many first introductions starts with "hello world" program, let us do the same:

```
#include <futurocube>

main()
{
 SetColor(cORANGE)
 DrawSquare(GetCursor())
 PrintCanvas()
 printf("hello world\r\n")
}
```

# 4 Walkers, squares, indexes

Each square on the cube surface is defined by specific INDEX and specific SIDE and SQUARE number. These values are connected together by simple math:

```
side=index/9
square=index%9
```

We can refer to squares either absolutely, that means we know their index, side, square... Or we can place on them something called WALKER. WALKER is packed structure, that contains information not only about the position of the WALKER but also about its direction. This is very useful, because we can move WALKER forward, backward, turned it and etc.. without paying attention what is his actual absolute position. Anytime after, we can retrieve information about its position by simple predefined macros:

```
index=_i(walker)
side=_side(walker)
square=_square(walker)
```

Of course the WALKER will always stick on the cube surface, so he moves correctly over the edges! Many functions takes WALKER directly as argument. For example, if we want to draw on square defined by WALKER position, we simply use DrawPoint(walker). We do not need to add any macro.

Many times we construct WALKER from known square index or known side and square. In this cases WALKER will have its default direction. You can use debug function DrawTail() to display orientation of the WALKER automatically.

```
walker=_w(4,4)
walker=_w(25)
```

Macro _i(...), which retrieve the index from the walker is very useful to obtain look up index into array, that represents for example some playground.

# 5 Programming

Script interpreter runs as separate task in the cube as well as any other games. Good behavior of the script is to perform so called co-operative multitasking. That means, after you do what you needed to, use function Sleep(). In case Sleep() is not used, nothing bad happen. System will cut out the script for a while and gives some time for system tasks. Well if the script is not complicated, no one will ever notice that. If there is important to delay the program or wait for something, like until music playback is over, always use delay functions with built in Sleep() instruction. These are for example: Sleep(), WaitPlayOver(), WaitMelodyOver(). Following two pictures show simplified scheduling and recommended basic programming workflow.

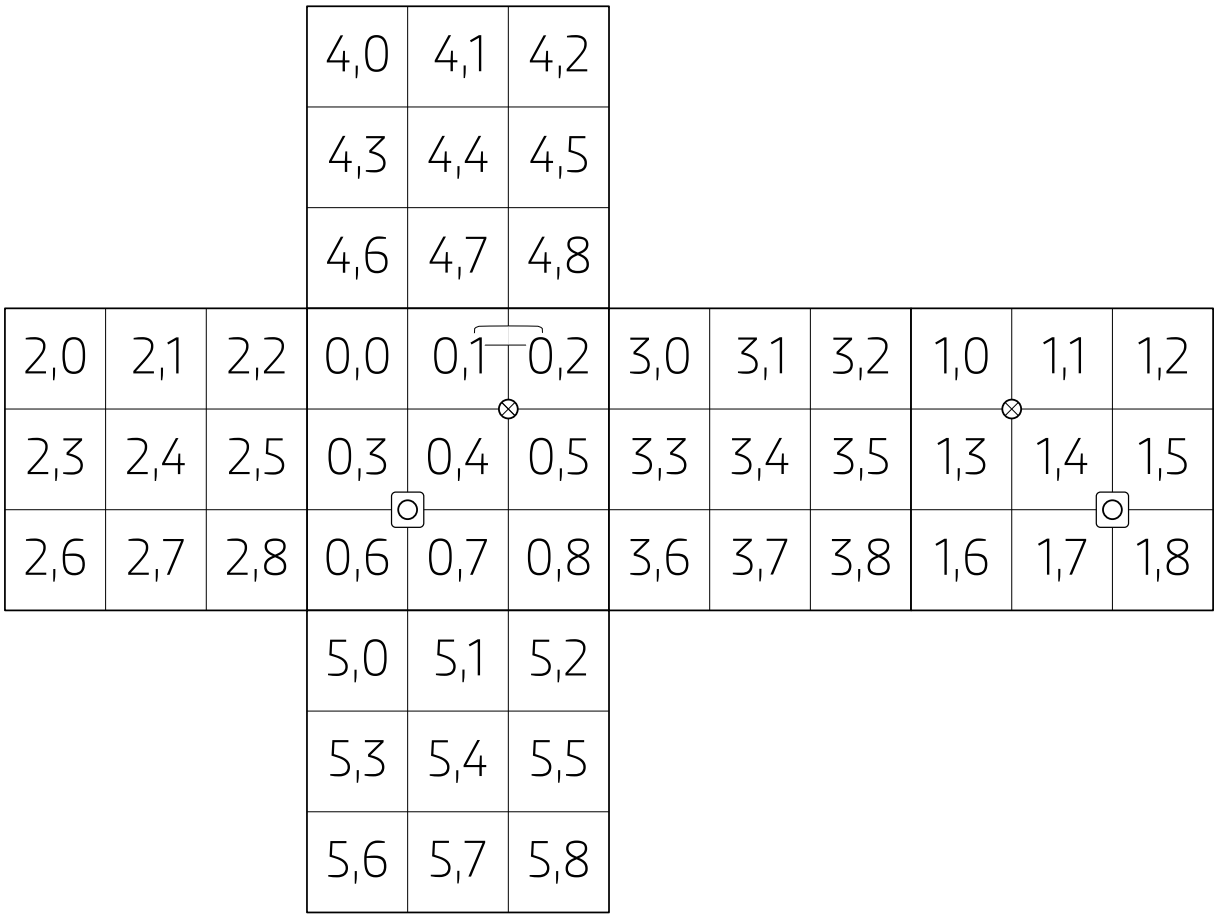


Figure 1: Side and squares numbers used in PAWN.

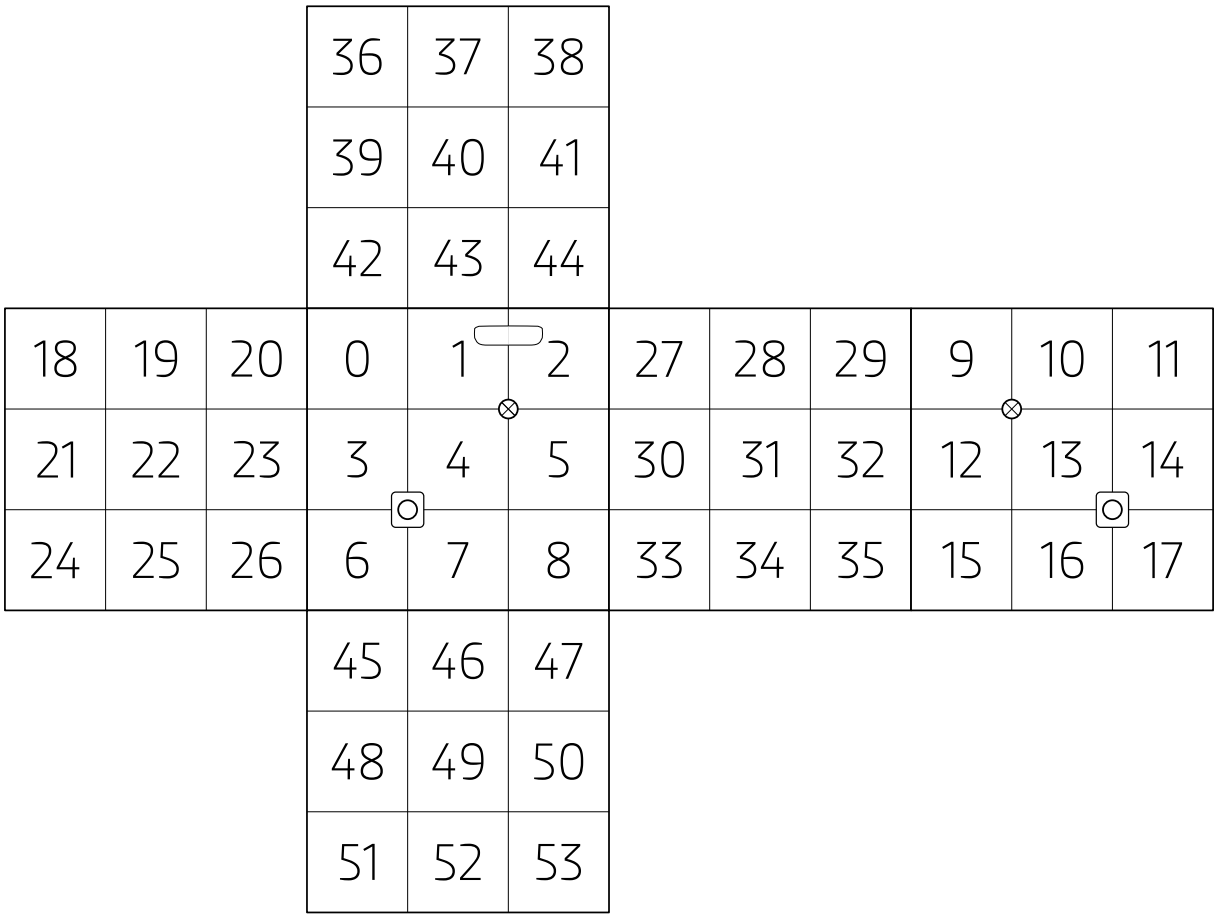


Figure 2: Square indexes used in PAWN.

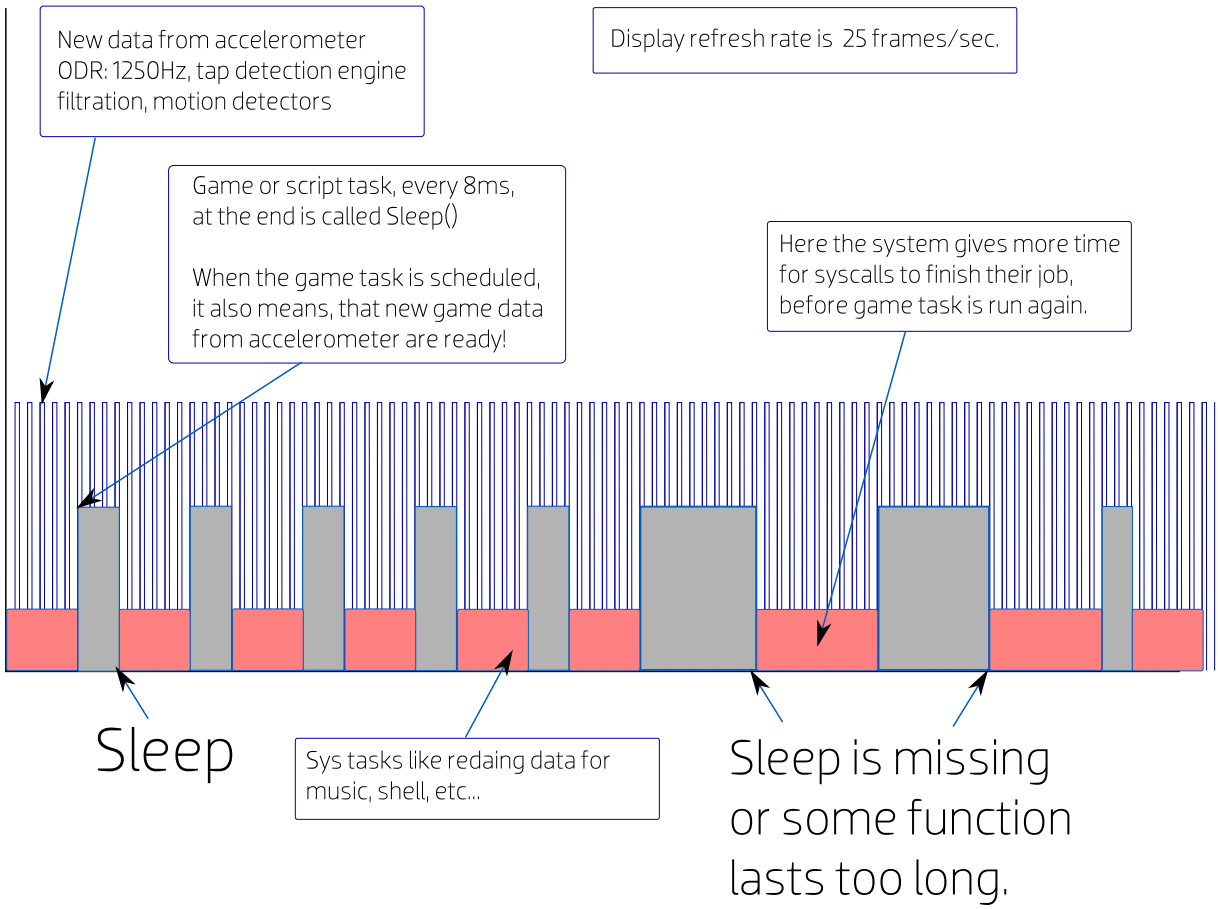


Figure 3: Simplified scheduling.

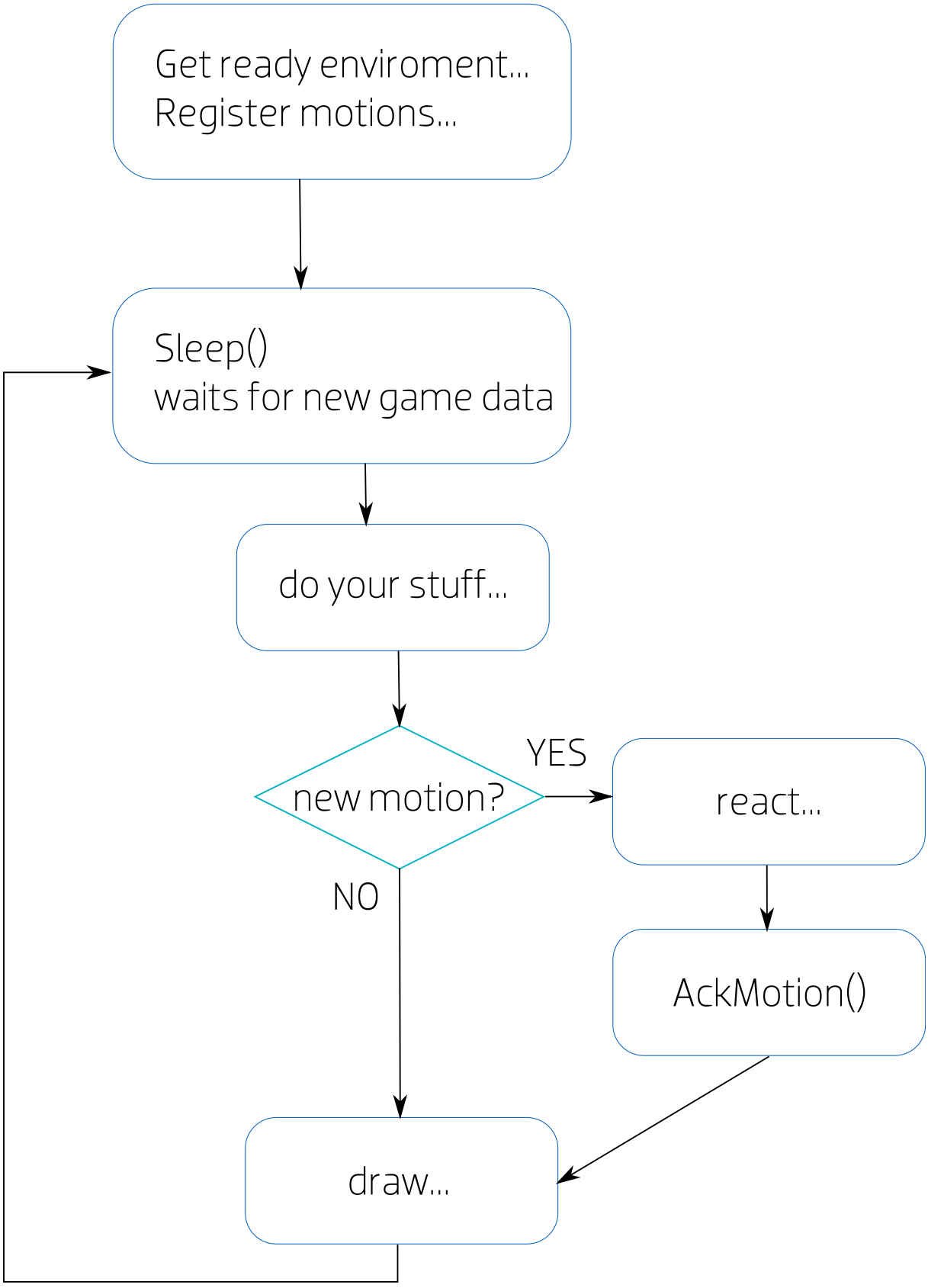


Figure 4: Recommended basic programming workflow.

# 6 Compiling, uploading and erasing scripts

Scripts can be uploaded into three destinations. Ram, Flash and Mycube Flash

## 6.1 Ram

In this area scripts are sharing data, code and stack. It is very useful for testing some functions or even whole scripts. If possible it is recommended to test the script into this area. And when if it works, upload it to flash. Limitation is size 11 KBytes including stack and data. Which is anyway pretty big for most of your initial work. RFCSuite has one button that automatically compiles, uploads and runs the selected script. So you can work pretty fast. If you need to test something where USB cable would create an obstacle, you don't have to upload script into flash. If it is stored in RAM, simply unplugged USB cable and the script restarts automatically. By next plug of USB the script is erased.

## 6.2 Flash

Scripts in flash may be 30 KBytes long (50 KBytes from FW 4.5) and use data and stack in whole RAM section (11Kbytes). This is main area for storing and running big scripts. If the cube is unplugged, script can be started form its ICON, which can be specified in its code. Or it creates default ICON in blue menu (from FW 4.5 in white menu). See further.

## 6.3 Mycube

MyCube scripts are scripts of size up to 10 KBytes (stack in RAM as well), that are written for special purposes. This is personalization of cube. To start them, one must simply do MENU GESTURE while being in MENU. Script will install automatically into MyCube area if its name contain .mycube. ("my.mycube.p")

## 6.4 Starting scripts in standard way

During debugging all scripts can be started by shell commands. But at regular run, there are differences for MyCube and scripts in FLASH. MyCube is special case, it has dedicated start. And it is repeating MENU GESTURE in GAME MENU. On the other hand, scripts in FLASH can have their own designed icon placed at specific side and menu, plus they can have their own sounds for name and explanations. It is achieved by adding ICON information into the script text file, which is compiled and later parsed by the cube and automatically set up in the menu. For details how to use look at the ICON() function in API.

In case no ICON is defined for the script resides in FLASH, there will appear new icon in BLUE MENU instead of CONNECT GAME, which will then starts the script by selecting it in standard way. From FW 4.5 at multiple scripts enviroment is situation slightly different. Scripts, that has no ICON defined are place in extra WHITE menu with dice symbols 1 to 6.

## 6.5 Multiple scripts support - from FW 4.5 and RFC 0.8

Rubik's Futuro Cube Suite from version 0.8 supports together with FW at least 4.5 mechanism to upload multiple scripts. This operation can be performed only over compiled AMX files which are not mycube. Use: "Tools-Install multiple scripts". Position of scripts, that do contain ICON information, can be changed by user before upload, however scripts without these information will placed in extra WHITE menu with cube symbols 1 to 6.

!!!!It is important to note, that any time button "Compile and Upload to FLASH" is used, than all installed multiple scripts are overwritten by new context!!!!

## 6.6 Starting scripts automatically

There is special setting available, that allows to start scripts automatically either when the USB is plugged or unplugged. Therefore is possible to replace standard charging appearance or the script can be started prior the GAME MENU. Following commands are involved, see details in shell command specification. In case both scripts have the same settings, than FLASH script has priority.

- setpawn type event .. this command is obsolote from FW 4.5, see "setpawnauto"
- setpawnstd ... put settings into default state .. this command is obsolote from FW 4.5

From FW 4.5 there is more inteligent command "setpawnauto" which works with multipl scripts as well.

Script running during charging process should not consume too much energy, because than the charging might be inefficient. In this case system might stop the script and start regular charging instead. Also if any script runs instead of menu, than if MENU GESTURE is performed, GAME MENU starts as usual.

## 6.7 Erasing scripts

Scripts in Ram are automatically erased each time USB cable is plugged in. Scripts in Flash or Mycube area can be erased by several ways:

- using command "pawnerase" from shell, it erases all areas, i.e Ram, Flash, and Mycube.
- in Rubik's Futuro Cube from version 0.6 pressing buttons either "Erase FLASH" or "Erase MYCUBE"
- from FW 4.5 there are two new commands: "pawneraseflash" and "pawnerasemycube"

# 7 Pawn specialties

we definitely recommend to study Pawn Language Guide at following address:

- http://www.compuphase.com/pawn/Pawn_Language_Guide.pdf

However next few articles can help with fast orientation in example codes and in language itself.

## 7.1 Default parameters

This is beautiful feature of PAWN known from higher languages. Definition of function can be done with some parameters having its default values. Later when calling this function, this parameter does not have to be filled. Many functions are structured in way that these default parameters are in last places and during call they are simply not mentioned.

```
//definition
AdjArray(arr[],pintensity,startindex=0,count=sizeof arr,size=sizeof arr)
{
...
}

AdjArray(ground,20)
//... calls AdjArray(ground,20,0,sizeof(ground),sizeof(ground))

AdjArray(ground,10,_,5)
//... calls AdjArray(ground,10,0,5,sizeof(ground))
//'_' means use default value, it is used, if the param is not at the end
```

## 7.2 Operator sizeof

Many functions are using operator sizeof as one of the default values. The nice thing is that for example as in function AdjArray() is the size of the array "arr" given at run time. And this value is usually used as boundary inside the function. So if really not needed, always leave this param out of the function and it will filled correctly for you.

## 7.3 Sizeof from multi-dimension arrays

With multi-dimensional arrays, the sizeof operator can return the number of elements in each dimension. For the last (minor) dimension, an element will again be a cell, but for the major dimension(s), an element is a sub-array.

```
new matrix[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } }
printf("%d %d", sizeof matrix, sizeof matrix[]);
```

Output is 3 a 2!

### 7.4 Packed and upacked strings

PAWN is using packed and unpacked strings. Packed string means that it will compress as many characters as possible into one cell and unpacked means that one cell contain only one character. How the string is constructed depends on used quotation marks. "stands for packed string" and "unpacked string". Using unpacked string is safer if you want to access the string as array. Than you can address characters directly. If you don't need this, than for saving memory is good to use packed strings. If any native function uses strings, it does not matter if is packed or unpacked. If you don't want to bother yourself with that, always use unpacked! Examples follows:

```
Play("music")              //music is packed string
new str[]=''ahoj''         //ahoj is unpacked
printf("%c\r\n",str[1])    //result is 'h'
```

# 8 Colors, palettes and drawing

If we close our eyes a little, how correct is the physical representation of colors on cube. We can use following coloring scheme: Each color is represented by values of RED, GREEN and BLUE component. Each value can be from 0 to 255. In hexadecimal representation we will have:

```
0xRRGGBB00
```

Last byte, which we left 00 is used for indexing color in palette. Palette is an array of 256 colors and any time we want to draw color which has lower byte non-zero, instead of taking its actual RGB values, index to palette is taken and color is retrieved from palette array. This array must be preloaded before use, otherwise it contains zeros.

All drawing is performed on so called Virtual Canvas. After the drawing is finished, Virtual Canvas can be printed on physical surface of the cube. All function that draw onto Virtual Canvas can use several draw styles. For details see SetDrawStyle() function.

# 9 PWM steps - 64 versus 256

Leds on cube surface are controlled by PWM modulation. At regular run there is for each LED 64 PWM steps available. Therefore each color component value before printing onto the cube surface (printing Virtual Canvas is not doing that yet) is divided by 4 and subtracted by 1. That has the effect, that for example SetRGB(7,7,7) gives still black. Full 256 steps can be switched on if needed. API will be provided later.

# 10 Push Pop Arrays

Push Pop Arrays are special containers, that work over existing arrays. Once initialized they provide 4 simple API access for pushing, poping and peaking various length arrays. Functionality can be used for example as LIFO storage for moves or structuring variables and many others. Some other functions like PushCanvas and PopCanvas works with them as well.

# 11 Default settings for scripts

Every time script is run, it has set up many defaults parameters. And if anything set in script, next run it becomes standard until changed again. These defaults has been selected to serve most common situations. Even some color is set up as active. So for beginning it is not necessary to check if everything has correct values....

# 12 Variables

PAWN script can access perzistance variables, where for example some game states can be stored for next run. Details about how to use please refer to "example_5_animated_rubiks.p" and to documentation of APIs related to variables.

# 13 Timers

Cube offers 10 countdown timers for various usage. Timers can be paused and resumed. They can be setup in milliseconds, however it is important to understand, that shortest interval and also actuall granularity is 8ms. That is the minimum tick for scheduling script application. If function Sleep() is called, it is same as if Delay(2) or Delay(8) would be called instead.

# 14 Basic commands to start with

There is lots of API functions. But following list enables you to quickly start writing some code with some basic functionality. Later on you can learn more complicated features.

```
ClearCanvas(), PrintCanvas(), SetColor(), DrawPoint(),
DrawPC(), RegMotion(),  RegAllSideTaps(), Motion(),
_is, AckMotion(), Sleep(), GetCursor(), _w(), printf(),
WalkerMove(), WalkerDiff(), ...
```

# 15 Shell commands

When Rubik's Futuro Cube is connected to Rubik's Futuro Suite. It is possible to switch to SDK mode and access shell commands. Shell is available for other applications as well, because USB provides VIRTUAL COM PORT. RFCSuite communicates with cube using more complicated multiplexed packets. Before accessing shell from other applications it is recommended to cycle USB to reset communication on character basis. Note that in this case characters are not automatically send back to terminal. Use command "echoon" if needed.

## 15.1 clr

Clears video memory at cube.

Syntax: `clr`

## 15.2 pled

Lights up leds according PAWN numbering.

Syntax: `pled side square led pwm`

| | |
|---|---|
| `side` | side number <0,5>, this value can be higher than 100, than "clr" is performed and 100 is subtracted! |
| `side` | square number <0,8> |
| `side` | led number <0,3>, if 3 it means all colors: WHITE |
| `pwm` | intensity <0,255> |

Notes: Intensity range is always scaled to actual PWM turnover. For most of the cases it is 64. Therefore intensity is divided by 4.

## 15.3 pleds

Lights up whole side according PAWN numbering.

Syntax: `pled side led pwm`

## 15.4 pledarr

Putting all leds value at once in array.

Syntax: `pledarr pwmr0,pwmg0,pwmb0,pwmr1,pwmg1,pwmb1,.,.,.,.,.,.`

| | |
|---|---|
| `pwmr0` | intesity of RED led at index 0 |
| `pwmg0` | intesity of BLUE led at index 0 |
| `pwmb0` | intesity of GREEN led at index 0 |

Notes: It is recommended to use "echooff" prior this function! Because echo works on character basis and for long commands it slows down the shell.

## 15.5 sn

Gives serial number of cube.

Syntax: `sn`

### 15.6 echoon

Enables echo of every character on shell line.

Syntax: `echoon`

### 15.7 echooff

Disables echo of every characters on shell line. (default)

Syntax: `echooff`

### 15.8 prompton

Enables prompt. (default)

Syntax: `prompton`

### 15.9 promptoff

disables prompt.

Syntax: `promptoff`

### 15.10 ver

Gives back version of FW and HW.

Syntax: `ver`

### 15.11 appinfo

Gives back memory layout of FW.

Syntax: `appinfo`

### 15.12 play

Plays sound resources from NAND FLASH

Syntax: `play HAVOK`

## 15.13 dir

List all sound resources on NAND FLASH

Syntax: `dir`

## 15.14 q

Quiet all ongoing music

Syntax: `q`

## 15.15 kill

Kill currently running script or game.

Syntax: `kill`

## 15.16 tinfo

Displays current task information

Syntax: `tinfo`

Notes: "no_idle" count value represents how many time game or scripts did not call Sleep within given time frame. If this number is high or constantly growing, something is wrong

## 15.17 pdir

Lists of all installed scripts with detailes and indexes - from FW 4.5

Syntax: `pdir`

## 15.18 prun

Start script in RAM

Syntax: `prun`

## 15.19 prunf

Start script in FLASH

Syntax: `prunf`

## 15.20 prunf N

Start script with index N in FLASH - from FW 4.5

Notes: Index is position of script in FLASH, which is created during upload of multiple scripts.

Syntax: `prunf 2`

## 15.21 prunm

Start script in FLASH - MYCUBE

Syntax: `prunm`

## 15.22 pawnerase

Erases scripts in RAM, FLASH and MYCUBE FLASH

Syntax: `pawnerase`

## 15.23 pawneraseflash

Erases scripts in FLASH - from FW 4.5

Syntax: `pawneraseflash`

## 15.24 pawnerasemycube

Erases scripts in MYCUBE - from FW 4.5

Syntax: `pawnerasemycube`

## 15.25 pawn

Displays info about installed scripts and current settings for automatic start

Syntax: `pawn`

## 15.26 setpawn

Sets automatic script starts. Obsolete in FW 4.5

Syntax: `setpawn type event`

| | |
|---|---|
| `type` | either "flash" or "mycube" |
| `event` | either "charge" or "menu" or "none" |

Example: `setpawn flash charge`, sets script in flash to be automatically started during charging

`setpawn mycube menu`, sets script mycube to be automatically started prior to menu

`setpawn mycube none`, sets script mycube to be never started automatically

See also: `setpawnstd`

## 15.27 setpawnstd

Reset any automatic script start settings. Obsolete in FW 4.5

## 15.28 setpawnauto

Auto script start for multiple scripts - from FW 4.5

Syntax: `setpawnauto event source [index]`

| | |
|---|---|
| `event` | "std" or "charge" or "menu" |
| `source` | "std" or "flash" or "mycube" |
| `index` | index of script in FLASH, if blank, than it equals to 0 |

Example: `setpawnauto std`, resets everything to default

`setpawnauto menu flash 2`, sets script of index 2 in FLASH to be automatically started prior to menu

`setpawnauto menu flash`, sets script of index 0 in FLASH to be automatically started prior to menu

`setpawnauto charge flash 3`, sets script of index 3 in FLASH to be automatically started instead of standard charging

`setpawnauto charge std`, sets standard charging

`setpawnauto charge mycube`, sets MYCUBE script to be automatically started instead of standard charging

## 15.29 pawnmem

Displays info about pawn memory layout

Syntax: `pawnmem`

## 15.30 picon

Displays info if ICON information is present in script FLASH area.

Syntax: `picon`

## 15.31 amxinfo

Displays info about stacks and data when the script is running.

Syntax: `amxinfo`

## 15.32 var

Prints out all available variables with its storage information

Syntax: `var`

## 15.33 varpawn

Prints out variables that has prefix _P_ (pawn variables)

Syntax: `varpawn`

## 15.34 varload

Prints out variable content if available

Syntax: `varload _P_xxxx`

## 15.35 varpawnerase

Erases pawn variable for debugging purposes

Syntax: `varpawnerase _P_xxxx`

# 16 Shell cmd "motion"

"motion" is special shell command that runs task which is giving all important data output for parsing outside of the cube. Output rate is 125 Hz. Structure and control sum is inspired at GPS output. It is usually not necessary to check control sum when USB transfer is used. No other messages can be in the middle of lines. However there is possible, that if you for example press enter, prompt will be placed in front of the message. Prompt can be switch off and than there is for example possible to use command "ledarr..." without disturbing lines.

```
$MOT,0002,-254,0001,4,4,00,1,-,-,-,-,-*57
$MOT,-003,-255,-001,4,4,00,1,-,-,-,-,-*57
$MOT,-001,-249,-005,4,4,00,1,-,-,-,-,-*5C
$MOT,0004,-257,-003,4,4,00,1,-,-,-,-,-*4D
$MOT,-003,-253,-007,4,4,00,1,-,-,-,-,-*57
$MOT,-010,-249,-010,4,4,00,1,-,-,-,-,-*58
$MOT,-017,-251,-017,4,4,00,1,-,-,-,-,-*51
$MOT,-016,-254,-018,4,4,00,0,-,-,-,-,-*5B
$MOT,-015,-252,-013,4,4,00,0,-,-,-,-,-*55
```

| | |
|---|---|
| MOT | standard beginning |
| accx | acc data x |
| accy | acc data y |
| accz | acc data z |
| side | cursor side |
| square | cursor square |
| shake | shake percentage |
| still flag | still flag "-" or "1" |
| still flag | still flag |
| gtap | generic tap "-" or "G" |
| tap side | tap side "-" or "0..5" |
| tap type | tap type "-" or "SIDE, TOP, BOT" |
| shake flag | shake flag "-" or "R" |
| menu | menu gesture flag "-" or "M" |
| control sum | NMEA control sum |

# 17 API-Native functions

## 17.1 ClearCanvas

Clear virtual canvas

Syntax: `ClearCanvas()`

Notes: This function clears virtual canvas. Usually it is used before drawing new picture on the cube. It is usually followed by sets of drawing functions and than by PrintCanvas(). This function itself does not clear LEDS on the cube!

Example: `ClearCanvas()`

See also: `PrintCanvas, DrawPoint`

## 17.2 PrintCanvas

Print virtual canvas to the physical cube surface, turns on LEDS

Syntax: `PrintCanvas()`

Notes: This function prints virtual canvas into physical cube surface. Virtual canvas is not automatically cleared after this operation.

Example: `PrintCanvas()`

See also: `ClearCanvas, DrawPoint, FlashCanvas`

## 17.3 SetIntensity

Set current intensity for all color components

Syntax: `SetIntensity(value)`

`value` represent current intesity in range <0..256>

Notes: Each color component when draw onto the canvas is scaled by intensity number. 256 means means no scale. Default value, which is set when the script starts is 128. This value reflects full and correct display of several presets colors.

Example: `SetIntensity(30)`

See also: `SetColorDefaults, SetColor`

## 17.4 SetColor

Set the current color

Syntax: `SetColor(value=0xffffff00)`

| | | |
|---|---|---|
| | `value` | 32 bit number, that represent the current color, see the explanation how the colors are represented |

Example: `SetColor(cGREEN)`

`SetColor(RED)`

`SetColor(0xFFAB1200)`

`SetColor(3)`, set color at pallete number 3

See also: `SetIntensity, SetColorDefaults`

## 17.5 Preset color definition

Definition of presets colors for classic cube puzzles

| | |
|---|---|
| `cGREEN` | `0x00640000` |
| `cBLUE` | `0x00008000` |
| `cRED` | `0x64000000` |
| `cORANGE` | `0x80340000` |
| `cMAGENTA` | `0x00785C00` |
| `cPURPLE` | `0x6C005C00` |

## 17.6 Basic color definition

Definition of basic colors

| | |
|---|---|
| `RED` | `0xFF000000` |
| `GREEN` | `0x00FF0000` |
| `BLUE` | `0x0000FF00` |
| `WHITE` | `0xFFFFFF00` |

## 17.7 SetRgbColor(r,g,b)

Set the current color by each color component

Syntax: `SetRgbColor(r,g,b)`

| | |
|---|---|
| `r` | red component of the color <0..255> |
| `g` | green component of the color <0..255> |
| `b` | blue component of the color <0..255> |

Example: `SetRgbColor(120,0,40)`

See also: `SetColor`

## 17.8 DrawPoint

Draw point of current color to virtual canvas

Syntax: `DrawPoint(wi)`

| | |
|---|---|
| `wi` | walker or square index |

Notes: Draw spot with current color to place defined by walker or index

Example: `DrawPoint(wi)`

`DrawPoint(23)`, draw to square index 23

`DrawPoint(_w(4,2))`, draw to side 4, square 2

See also: `DrawSide, DrawSquare, DrawCross`

## 17.9 DrawPC

Draw point with given color and intensity

Syntax: `DrawPC(wi, color, int=128)`

| | |
|---|---|
| `wi` | walker or square index |
| `color` | requested color |
| `int` | requested intensity |

Notes: Draw spot with given color and intensity to place defined by walker or index. Given values do not modify current color and current intensity. They apply just for one call of this function.

Example: `DrawPC(GetCursor(),WHITE)`, draw white spot at cursor

`DrawPC(2,cORANGE,256)`, draw to square index 2 cORANGE with max intensity

See also: `DrawPoint, DrawSide, DrawSquare, DrawCross`

## 17.10 DrawSide

Fill side with current color to virtual canvas

Syntax: `DrawSide(side)`

`side` side number

Notes: Fill given side with current color.

Example: `DrawSide(_side(wlk))`, fill side defined by walker wlk

Example: `DrawSide(_side(27))`, fill side defined by square index 27, that is side 3

Example: `DrawSide(4)`, fill side 4

See also: `DrawSide, DrawSquare, DrawCross`

## 17.11 DrawSquare

Draw square 3x3 with current color around center spot

Syntax: `DrawSquare(wi)`

`wi` walker or square index

Notes: Draws square around center spot of preset color. Center spot is defined by walker or square index

Example: `DrawSquare(wlk)`, draw square with center of wlk

`DrawSquare(27)`, draw square with center of square index 27

`DrawSquare(_w(4,2))`, draw square around center spot side 4 square 2

See also: `DrawPoint, DrawSide, DrawCross`

## 17.12 DrawCube

Fill cube with current color to virtual canvas

Syntax: `DrawCube()`

Notes: Fill cube current color.

Example: `DrawCube()`

See also: `DrawSide, DrawSquare, DrawCross`

## 17.13 DrawCross

Draw cross of specified length

Syntax: `DrawCross(wi, length=1)`

| | |
|---|---|
| `wi` | walker or square index |
| `length` | length of the arm of the cross |

Notes: Draw cross with center spot defined by walker or index with preset color and attributes.

Example: `DrawCross(wlk)`, draws cross with center wlk with length=1

`DrawCross(23,2)`, draws cross on index 23 with length=2

`DrawCross(_w(2,2),3)`, draws cross with center side 2, square 2 and length=3

See also: `DrawSide, DrawPoint, DrawCross`

## 17.14 PushCanvas

Push virtual canvas to Push Pop Array

Syntax: `PushCanvas(ppindex=0)`

| | |
|---|---|
| `ppindex` | push pop array number, default is zero |

Returns: Funtion returns error code as function Push

Notes: This function internally uses Push, so the ppindex array must be initialized!

Example: `PushCanvas()`, push canvas to array with index 0

`PushCanvas(1)`, push canvas to array with index 1

See also: `PopCanvas, Push, Pop, PushPopInit`

## 17.15 PopCanvas

Pop virtual canvas from Push Pop Array

Syntax: `PushCanvas(ppindex=0)`

| | |
|---|---|
| `ppindex` | push pop array number, default is zero |

Returns: Funtion returns error code as function Pop

Notes: This function internally uses Pop, so the ppindex array must be initialized!

Example: `PopCanvas()`, pop canvas from array with index 0
`PopCanvas(1)`, pop canvas to array with index 1

See also: `PushCanvas, Push, Pop, PushPopInit`

## 17.16 CanvasToArray

Store virtual canvas to array

Syntax: `CanvasToArray(arr[],size=sizeof arr)`

| | |
|---|---|
| `arr[]` | destination array |
| `size` | size of the array, must be 54, otherwise exception is reised |

Returns: Funtion always returns 0

Notes: This function basically copies virtual canvas to user array.

Example: `CanvasToArray(temp)`, stores canvas to array temp

See also: `ArrayToCanvas, Push, Pop, PushPopInit`

## 17.17 ArrayToCanvas

Load virtual canvas from array

Syntax: `ArrayToCanvas(arr[],size=sizeof arr)`

| | |
|---|---|
| `arr[]` | source array |
| `size` | size of the array, must be 54, otherwise exception is reised |

Returns: Funtion always returns 0

Notes: This function basically copies user array to virtual canvas.

Example: `ArrayToCanvas(temp)`, store array temp to canvas

See also: `CanvasToArray, Push, Pop, PushPopInit`

## 17.18 DrawArray

Draw array to virtual canvas

Syntax: `DrawArray(arr[],size=sizeof arr)`

| | |
|---|---|
| `arr[]` | source array |
| `size` | size of the array, must be 54, otherwise exception is reised |

Returns: Funtion always returns 0

Notes: This function draws user array to virtual canvas. That means it really uses all drawing procedure and preset drawing style.

Example: `DrawArray(temp)`, draw array temp to canvas

See also: `SetDrawDefaults, SetDrawStyle`

## 17.19 PrintCnv

Debug function, print virtual canvas values to terminal

Syntax: `PrintCnv()`

Example: `PrintCnv()`

See also: `PrintArray, printf`

## 17.20 SetDrawDefaults

Set all draw properties to its default states

Syntax: `SetDrawDefaults()`

Notes: This function sets: Intensity: 128, Color to 0x80340000, Blend to 50 percent and drawing style to STD. This funtion does not affect pallete settings.

Example: `SetDrawDefaults()`

See also: `SetDrawStyle`

## 17.21 Drawing style definition

Drawing styles definition and explanation

| | |
|---|---|
| `STD` | `0, standard - new color overrides everything` |
| `ADD` | `1, addition - colors are added (and clipped) by components` |
| `SUB` | `2, substitution - colors are substitued (and clipped) by components` |
| `MSKA` | `3, MASKING by color A` |
| `MSKB` | `4, MASKING by color B` |
| `MSKADD` | `5, MASKING by color A plus addition` |
| `BLD` | `6, blending colors by blend value` |

## 17.22 SetDrawStyle

Set draw style for all drawing functions

Syntax: `SetDrawStyle(style,blend=50)`

| | |
|---|---|
| `style` | style from style definition |
| `blend` | blend percentage value used for BLD style <0..100> |

Notes: Set up style is than used for all drawings and is used accordingly.

Example: `SetDrawStyle(ADD)`, set draw style to addition

`SetDrawStyle(BLD,20)`, set draw style to blending, 20 and 80 percent

See also: `SetPallete, SetDrawDefaults`

## 17.23 SetPalette

Fill the palette with given color

Syntax: `SetPalette(index,color)`

| | |
|---|---|
| `index` | index in palette, not zero!, range <1..255> |
| `color` | color that will be stored at given index |

Notes: Palette at index zero have to contain zero value for color, therefore index zero is not accessible. By starting script, palette values are reset to zeroes as well.

Example: `SetPalette(1,WHITE)`, set palette with index 1 to value WHITE

`SetPalette(2,0xABCD1100`, set palette with index 2 to value 0xABCD1100

## 17.24 PaletteFromArray

Fill the palette with values from array

Syntax: `PaletteFromArray(arr[],size=sizeof array)`

| | |
|---|---|
| `arr` | source array |
| `size` | size of array, maximum size is 255 |

Notes: This function set up palette from array, index 0 from array is stored into index 1 in palette. Index 0 in palette is allways 0.

Example: `new parr[3]={WHITE,BLUE,RED}`

`PaletteFromArray(parr)`, set palette from parr

## 17.25 FlashCanvas

Prints virtual canvas to LEDS with fading attributes

Syntax: `FlashCanvas(step=1,diff=3,exclusive=0)`

| | |
|---|---|
| `step` | Indicates occurrence of fade towards frame counter, value 1 means proceed with fade every new frame, value 2 means every second frame... |
| `diff` | indicate what value will be subtracted from each color component every step |
| `exclusive` | if the parameter is set to 1, than LEDS are updated only if their value was zero. That means if they were for example still fading from previous call, their values are not overriden |

Notes: This function is usually used for some graphical effects, where canvas is not needed to be updated so often

Example: `FlashCanvas`

See also: `PrintCanvas, ClearCube`

## 17.26 Flicker type definition

Flicker types definition and explanation

| | |
|---|---|
| `FLICK_STD` | `0, flickering min->max,max->min,...` |
| `FLICK_RAZ` | `1, flickering min->max,min->max,...` |

## 17.27 DrawFlicker

Draw cursor type blinking spot

Syntax: `DrawFlicker(wi,speed=20,type=0,phase=0)`

| | |
|---|---|
| `wi` | walker or spot index |
| `speed` | blinking speed, each frame this value is used to adjust the spot |
| `type` | type of flicker, FLICK_STD or FLICK_RAZ |
| `phase` | this value adjust phase of blinking, it can be in range <-127..127> |

Notes: In order to make this function work properly, it must be called repeatedly during drawing. It actually just draw a spot of current color, but it calculates its intensity by given values and current frame counter. Therefore it creates nicely blinking spot on virtual canvas.

Example: `DrawFlicker(GetCursor()`

See also: `GetCursor`

## 17.28 AdjCanvasPoint

Adjust intensity of the already drawed point

Syntax: `AdjCanvasPoint(wi,pint)`

| | |
|---|---|
| `wi` | walker or spot index |
| `pint` | percentage of new intensity, 0 means no change <-100..300> |

Notes: This function changes intensity of point which is already drawn. So the color component might not be in correct ratio if the change is big.

Example: `AdjCanvasPoint(GetCursor(),-20)`, decrease intensity under cursor to 20 percent

See also: `AdjCanvas, AdjArray`

## 17.29 AdjCanvas

Adjust intensity of the virtual canvas

Syntax: `AdjCanvas(pint)`

| | |
|---|---|
| `pint` | percentage of new intensity, 0 means no change <-100..300> |

Example: `AdjCanvas(-20)`, decrease intensity of all canvas to 20 percent

See also: `AdjCanvasPoint, AdjArray`

## 17.30 AdjArray

Adjust intensity of the array

Syntax: `AdjArray(arr[],pint,sindex=0,count=sizeof arr,size=sizeof arr)`

| | |
|---|---|
| `arr` | array to be adjusted |
| `pint` | percentage of new intensity, 0 means no change <-100..300> |
| `sindex` | start index in array |
| `count` | number of cells to adjust in array |
| `size` | size of array |

Example: `AdjArray(arr,-50)`, decrease intensity of arr to 50 percent

`AdjArray(arr,50,20,2)`, decrease intensity of arr[20] and arr[21] to 50 percent

## 17.31 ReadCanvas

Read color from virtual canvas

Syntax: `ReadCanvas(wi)`

`wi` walker or spot index

Returns: Color that contains virtual canvas at given position.

Example: `col=ReadCanvas(2)`, read the canvas color from square index 2

See also: `ReadRGBLed`

## 17.32 ReadRGBLed

Read actuall RGB color that is currently displayed

Syntax: `ReadRGBLed(wi)`

`wi` walker or spot index

Returns: Color that is in reality diaplayed on cube surface at the moment of calling the function.

Example: `col=ReadRGBLed(2)`, read the RGB led color from square index 2

See also: `ReadCanvas`

## 17.33 ClearCube

Clear all RGB leds on the cube

Syntax: `ClearCube()`

Notes: The visible result is same is if sequence ClearCanvas() and PrintCanvas() would be called, however virtual canvas is not modified. This function is useful as example with FlashCanvas(), because it access surface directly.

Example: `ClearCube()`

See also: `ReadRGBLed`

## 17.34 DrawTail

Debug function for displaying direction of walker

Syntax: `DrawTail(w)`

Example: `DrawTails(w)`

## 17.35 Motion pattern type list definition

Definition of basic motion pattern

| | |
|---|---|
| TAP_GENERIC | 5, generic tap any direction |
| SHAKING | 8, shaking side to side |
| TAP_XPLUS | 9, side tap in X+ axes (side 3) |
| TAP_XMINUS | 10, side tap in X- axes (side 2) |
| TAP_YPLUS | 11, side tap in Y+ axes (side 4) |
| TAP_YMINUS | 12, side tap in Y- axes (side 5) |
| TAP_ZPLUS | 13, side tap in Z+ axes (side 0) |
| TAP_ZMINUS | 14, side tap in Z- axes (side 1) |
| TAP_DOUBLE | 15, double tap flag |

## 17.36 RegMotion

Register specific motion pattern for recognition

Syntax: `RegMotion(type)`

`type` motion type from pattern type list

Notes: Motion type is defined basically by position of bit in motion mask. Later, if new motion is detected, output from Motion() function has corresponding bit set to logic 1. By calling RegMotion(...) consequently, another pattern is added.

Example: `RegMotion(TAP_GENERIC)`, registers TAP_GENERIC

See also: `Motion, AckMotion, RegAllSideTaps, UnregAllMotion`

## 17.37 SetDoubleTapLength

Set maximum length of double tap detection in milliseconds

Syntax: `SetDoubleTapLength(max_ms=700)`

`max_ms` maximum length in milliseconds for double tap detection

Notes: Window for double tap detection is set from 50ms to user defined value. Standard length in 700ms, minimum length for double tap detection is 100ms.

Example: `SetDoubleTapLength(200)`, sets double tap detection length to 200ms

`SetDoubleTapLength()`, sets double tap detection length to 700ms

See also: `Motion, AckMotion`

## 17.38 Motion

Detects if any registered motion pattern has been recognized

Syntax: `type=Motion()`

Returns: This function returns number, that has set bits at positions, that corresponds to recognized registered motion pattern.

Notes: If for example more than one pattern is registered, than output from this function should be tested for each pattern separately. The typical way is to use predefined macro _is, which is testing, if the bit on its specified position is set or reset.

Example: `motion=Motion()`, variable motion holds information about recognized patterns
`if (motion) {...}` if there is any motion, we should handle it ...

`if (_is(motion,TAP_XPLUS)) {...}` if there is specifically TAP_XPLUS, we should do ...

See also: `RegMotion, AckMotion, RegAllSideTaps, UnregAllMotion`

## 17.39 AckMotion

Acknowledge reception of recognized motion patterns

Syntax: `AckMotion()`

Notes: This function tells to the system, you have received and served all recognized motion patterns previously obtained from Motion() and therefore any new motion pattern can be recognized. If this function is not called, bits on output from Motion() remain set if as there is still some pattern presented.

Example: `AckMotion()`

See also: `RegMotion,Motion,RegAllSideTaps,UnregAllMotion`

## 17.40 RegAllSideTaps

Register taps to all sides

Syntax: `RegAllSideTaps()`

Notes: This function does exactly same as one would call several times RegMotion with all side taps parameters. TAP_GENERIC is not registered!

Example: `RegAllSideTaps()`

See also: `RegAllTaps`

## 17.41 UnregMotion

Unregister specified motion pattern

Syntax: `UnregMotion(type)`

| | | |
|---|---|---|
| | `type` | motion type from pattern type list |

Notes: This function does exactly opposite than RegMotion

Example: `UnregMotion(TAP_GENERIC)`

See also: `RegMotion`

## 17.42 UnregAllMotion

Deregister all motion pattern

Syntax: `UnregAllMotion()`

Notes: After this function is called, there will no pattern recognized

Example: `UnregAllMotion()`

See also: `RegMotion`

## 17.43 GetTapSide

Gives back number of side, that has been tapped

Syntax: `result=GetTapSide(&side)`

| | | |
|---|---|---|
| | `&side` | variable side will contain number of sied that has been tap to |

Returns: Function returns 1 if the tap to side has been recognized, otherwise 0

Notes: Function fill the side variable with value 0 to 5 according which side has been tapped. Function reads Motion(), so for correct operation side taps must be registered. If you call AckMotion() prior this function, result will become 0.

Example: `if (GetTapSide(side)) {...work with side...}`

## 17.44 RegAllTaps

Register generic tap and all side taps.

Syntax: `RegAllTaps()`

Notes: This function register generic tap and all sides taps.

Example: `RegAllTaps()`

See also: `RegAllSideTaps`

## 17.45 GetTapType

Decision between side, top and bottom tap

Syntax: `res=GetTapType(topside_wi)`

| | |
|---|---|
| `topside_wi` | walker or index from which is used for information, which side points towards top |

Returns: One of the following results is returned:

| | |
|---|---|
| `0` | no decision can be made, perhaps no side tap has been detected |
| `1` | tap to gravity side has been detected |
| `2` | tap to gravity top has been detected |
| `3` | tap to gravity bottom has been detected |

Notes: This function same as GetTapSide reads Motion() to retrieve pattern info. Than it combines this motion pattern info with user input which side is pointing up and returns type of gravity tap.

Example: `res=GetTapType(GetCursor())`

## 17.46 Easy Tap Type Detection

Easier to use set of functions for tap type Detection

Syntax: `eTapSideOK()`

Notes: Check if is detected any valid SIDE TAP. It is usually called before eTapSide() to confirm correct results. If for example TAP_GENERIC would be registered as well. eTapSide() might give wrong output, because if function does not know, it gives back 0.

Syntax: `eTapSide()`

Notes: Gives back number of side that has been tapped. If motion pattern do not include any specific taps, it returns 0 as well. That is why it is good to use eTapSideOK(). If only sides taps are registered, than there is no problem.

Syntax: `eTapToSide()`

Notes: Returns 1, if last tap was to side. It takes top side from acc data, past=4

Syntax: `eTapToTop()`

Notes: Returns 1, if last tap was to top. It takes top side from acc data, past=4

Syntax: `eTapToBot()`

Notes: Returns 1, if last tap was to bottom. It takes top side from acc data, past=4

Example:
```
if (eTapSideOK()) {side=eTapSide();...  work with side..}
if (eTapToTop) {...}
```

## 17.47 IsStill

Detection of general movements

Syntax: `res=IsStill()`

Returns: This functions returns 1 if there is some noticeable movements or 0 if the cube is still.

Notes: This function has no special parameters, it is simpler version of GetShake(), that can be use for more complicated situations.

Example: `res=IsStill()`

See also: `GetShake`

## 17.48 GetShake

Detection of shaking in percentage scale

Syntax: `res=GetShake()`

Returns: This function returns values in range <0..100>

Notes: Value changes depending on how long and how much the cube is "shaken". The value constantly rises up while the shaking is in progress and when it ends , it goes quickly down to 0

Example: `res=GetShake()`

See also: `IsStill`

## 17.49 Sleep

Gives control to system and waits until new game accelerometer data arrives

Syntax: `Sleep(value=0)`

Notes: This function is essential for cooperative multitasking. It should be called after all events are processed. If the program does not call this function regularly system can block the task for non-cooperative behavior.

Example: `Sleep()`, most general use of sleep, waits for new data, gives time to system

`Sleep(100)`, same as Delay(100)

See also: `Delay`

## 17.50 Delay

Delays script by given millisecond

Syntax: `Delay(value=1000)`

`value` delay time in milliseconds

Notes: This function delays at least given millisecond. Internally uses Sleep() so the delay is fully cooperative.

Example: `Delay()`delays for 1 secs

`Delay(100)`delays for 100 milliseconds

See also:

## 17.51 SetTimer

Sets in milliseconds one of 10 countdown timers

Syntax: `SetTimer(timer=1,value=1000)`

`timer` 0,1,2..9 - available timer

Notes: This function sets one of 10 available countdown timers to value that represents millisecond. From that point counter is counting down to zero. This function also resumes the timer if was paused.

Example: `SetTimer()`, set timer 0 to value 1000 milliseconds

`SetTimer(1,200)`, set timer 1 to value 200 milliseconds

See also: `GetTimer, PauseTimer, ResumeTimer`

## 17.52 GetTimer

Returns value of one of 10 countdown timers

Syntax: `res=GetTimer(timer=0)`

`timer` 0,1,2..9 - available timer

Returns: Current value of the timer, no lower than 0

Example: `res=GetTimer`, returns value of timer 0

`res=GetTimer(2)`, returns value of timer 2

```
if (!TimerGet(0))
```

See also: `SetTimer, PauseTimer, ResumeTimer`

## 17.53 PauseTimer

Pause one of the 10 countdown timers

Syntax: `PauseTimer(timer=0)`

| | |
|---|---|
| `timer` | 0,1,2..9 - available timer |

Notes: Timer remains paused until ResumeTimer() or SetTimer() function is called.

Example: `PauseTimer()`, pause timer 0

`PauseTimer(2)`, pause timer 2

See also: `SetTimer, ResumeTimer`

## 17.54 ResumeTimer

Resume one of the 10 countdown timers

Syntax: `ResumeTimer(timer=0)`

| | |
|---|---|
| `timer` | 0,1,2..9 - available timer |

Example: `ResumeTimer()`, resume timer 0

`ResumeTimer(2)`, resume timer 2

See also: `SetTimer, PauseTimer`

## 17.55 printf

Debug function for text output

Syntax: `printf(const format[], Fixed,_:...)`

| | |
|---|---|
| `const format[]` | array with formatting text |
| `...` | variable arguments used to feed into formatting text |

Notes: This function works similar as standard printf function, only there is just subset of format specifiers

| | |
|---|---|
| `%s` | inserts a string that can be either packed or unpacked |
| `%d` | signed integer number |
| `%x` | hexadecimal number |
| `%X` | hexadecimal number |
| `%c` | character |

Example:
```
printf("%s\r\n",data)
printf("%s\r\n",data)

printf("this is %d, this is text %s\r\n",number, data)

printf("0x%08x\r\n",hexnumber)
```

See also: `snprintf`

## 17.56 snprintf

Formatting output into unpacked string

Syntax: `snprintf(dest[],size=sizeof dest,const format[], {Fixed,_}:...)`

| | |
|---|---|
| `dest[]` | string array where to store formatted output |
| `size` | size of output array |
| `const format[]` | string array with formatting text |
| `...` | variable arguments used to feed into formatting text |

Notes: This function works same as printf with exception that it stores the output into destination. Size of destination must be specified and usually is retrieved during runtime by leaving standard value.

Example: `snprintf(dest,"My name is %s\r\n",_,name)`, `_` stands for sizeof dest and it is filled automatically in runtime

See also: `printf`

## 17.57 cellset

Sets the array of predefined constant

Syntax: `cellset(arr[],val=0,size=sizeof arr)`

| | |
|---|---|
| `arr[]` | array to be filled by val |
| `val` | value used for filling |
| `size` | size of array |

Notes: This is similar function to memset. It basically put value into every cell in array. Size of array can be automatically inserted in runtime by using standard value.

Example: `cellset(playground,20)`, fills whole playground array by value 20

`cellset(playground,20,3)`, fills first 3 cells by value 20

See also: `cellcopy`

## 17.58 cellcopy

Copying one array to another with using memmove

Syntax: `cellcopy(dest[],source[],index=0,numcells=sizeof source,maxlength=sizeof dest)`

| | |
|---|---|
| `dest` | destination array |
| `source` | source array |
| `index` | index of cell in source from where to start copy |
| `numbytes` | number of cells to copy |
| `maxlength` | size of the destination |

Notes: This function copies cells from one array to another. If last 3 parameters are left defalt it performs safe array to array copy. This function internally uses memmove, so arrays can overlap.

Example: `cellcopy(dest,source`, tries to whole source to max size of dest

`cellcopy(dest,source,2,10,_`, tries copy whole 10 cells from source at index 2, but maximum up to size of dest, however size of source might be violated

`cellcopy(dest,source,_,10`, same as above from index 0

See also: `cellset`

## 17.59 PushPopInit

Initialization of Push Pop Arrays

Syntax: `PushPopInit(arr[],ppindex=0,size=sizeof arr)`

| | |
|---|---|
| `arr` | array that will be used as Push Pop Array |
| `ppindex` | index of available PPArray arbiter, <0..3> |
| `size` | size of array to be stored into arbiter as initial size |

Notes: Before any usage of Push, Pop or PushCanvas, PopCanvas, array must be initialized!

Example: `PushPopInit(arr)`, initialized arbiter with index 0 with array arr

`PushPopInit(temp,1)`, initialized arbiter with index 1 with array temp

See also: `Push, Pop`

## 17.60 Push

Push array into Push Pop Array

Syntax: `Push(arr[],ppindex=0,size=sizeof arr)`

| | |
|---|---|
| `arr` | array that is going to be pushed in |
| `ppindex` | index of available PPArray arbiter, <0..3> |
| `size` | size of pushed array |

Returns: Returns 1 if operation was successful, otherwise exception is raised.

Notes: If the Push Pop Array if full, simply part of oldest items are thrown away and lost. This must be considered as possible effect and be careful about that.

Example: `Push(arr)`, push arr into initialized array with index 0

`Push(temp,1)`, push temp into initialized array with index 1

See also: `Pop, PPReady`

## 17.61 Pop

Pop array from Push Pop Array

Syntax: `Push(arr[],ppindex=0,size=sizeof arr)`

| | |
|---|---|
| `arr` | array that is going to be popped |

`ppindex` index of available PPArray arbiter, <0..3>

`size` size of popped array

Returns: Returns 1 if operation was successful, otherwise exception is raised.

Notes: If the Push Pop Array if empty or more than available cells are popped, excpetion is raised as well!

Example: `Pop(arr)`, pop arr from initialized array with index 0

`Pop(temp,1)`, pop temp from initialized array with index 1

See also: `Push, PPReady`

## 17.62 PPReady

Number of ready cells for Pop in Push Pop Array

Syntax: `PPReady(ppindex=0)`

`ppindex` index of PPArray arbiter, <0..3>

Returns: Number of available cells to pop.

Notes: If there is possibility to call Pop without being sure about available cells, than this function should be use to check it. Otherwise exception will be raised if Pop is performed on empty or not enough filled PPArray.

Example: `if (PPReady() {Pop(arr);...}`

See also:

## 17.63 PPFree

Number of free cells for Push in Push Pop Array

Syntax: `PPFree(ppindex=0)`

`ppindex` index of PPArray arbiter, <0..3>

Returns: Number of free cells to push.

Notes: Using this function is ensured that oldest Push will not be thrown away.

Example: `if (PPFree() {Push(arr);...}`

See also: `PPReady`

## 17.64 STEP definition

Defined values for step types

| | |
|---|---|
| STEP_NOTHING | 0 |
| STEP_FIRST | 1 |
| STEP_FORWARD | 2 |
| STEP_BACKWARDS | 3 |
| STEP_RIGHT | 4 |
| STEP_LEFT | 5 |
| STEP_UPRIGHT | 6 |
| STEP_UPLEFT | 7 |
| STEP_DOWNRIGHT | 8 |
| STEP_DOWNLEFT | 9 |
| STEP_HEAD | 10 |

## 17.65 TURNS definition

Defined values for turn types

| | |
|---|---|
| TURN_RIGHT | 0 |
| TURN_LEFT | 1 |

## 17.66 _w

Creates initialized walker with its default direction

Syntax: `_w(side,square=-1)`

| | |
|---|---|
| `side` | side number or index number of square |
| `square` | square number |

Returns: Returns initialized walker with its default direction.

Notes: Input to this function can be either side and square number or just square index. Second case is used if square argument is used with its default value.

Example: `walk=_w(4)`, initialized walk to square index number 4

`walk=_w(4,3)`, initialized walk to side 4, square 3

See also: `WalkerMove, WalkerTurn`

## 17.67 WalkerMove

Move walker on the cubic surface

Syntax: `WalkerMove(&w,step=STEP_FORWARD)`

| | |
|---|---|
| `w` | walker that will be moved |
| `step` | type of move to proceed with |

Returns: This function return 1 if the walker went over the edge, otherwise returns 0.

Notes: The step parameter is one step types.

Example: `WalkerMove(walk)`, move walk one step forward

`WalkerMove(walk,STEP_RIGHT)`, move walk one step to the right without turning

See also: `WalkerTurn`

## 17.68 WalkerTurn

Turns walker to the right or left

Syntax: `WalkerTurn(&w,step=TURN_RIGHT)`

| | |
|---|---|
| `w` | walker to be turned |
| `step` | type of turn to proceed |

Returns: This function returns always return 0.

Notes: The step parameter may be one of these:

| | |
|---|---|
| `TURN_RIGHT` | turns walker to the right |
| `TURN_LEFT` | turns walker to the left |

Example: `WalkerTurn(walk)`turns walk to the right

`WalkerTurn(walk),TURN_LEFT)`turns walk to the left

See also: `WalkerPlace, WalkerMove`

## 17.69 WalkerDiff

Calculates distance in 2D coordinate system and hint the next step

Syntax: `WalkerDiff(wa,wb,&dx,&dy)`

| | |
|---|---|
| `wa` | walker or index of spot A |
| `wb` | walker or index of spot B |
| `dx` | place where to store difference in x coordinate |
| `dy` | place where to store difference in y coordinate |

Returns: Immediate return value gives step type how to wa should continue to meet wb.

| | |
|---|---|
| `STEP_NOTHING` | walkers are at same spot or at opposite sides!!!!! |
| `STEP_ ...` | if walker wa proceed with this step, he will be closer to wb |

Notes: Dx and dy will contain values that represent number of steps in each direction that walker wa would have to make to reach wb in order dx and dy. If wa would continue by given suggested step and than ask for another one, finally it would reach wb. Returned step is determined by highest absolute value between dx and dy. Dx relates to STEP_FORWARD and STEP_BACKWARDS, while dy is STEP_LEFT and STEP_RIGHT. If wa is on opposite side of wb, than STEP_NOTHING is performed and returned.

Example: `WalkerDiff(23, GetCursor(),dx,dy)`, difference from index 23 to actuall cursor

See also: `WalkerStepTo, GetCursor, ReadAcc`

## 17.70 WalkerStepTo

Easier version of WalkerDiff

Syntax: `WalkerStepTo(&wa,wb)`

| | |
|---|---|
| `wa` | walker that will be moved towards wb |
| `wb` | target walker or spot index for direction |

Returns: Function returns same step that walker wa perfomed inside the function.

Notes: If wa is on opposite side of wb, than STEP_NOTHING is performed and returned.

Example: `WalkerStepTo(player,GetCursor()`, moves player towards the cursor

See also: `WalkerDiff`

## 17.71 OppositeStep

gives opposite step

Syntax: `OppositeStep(step)`

| | |
|---|---|
| `step` | type of step to obtain its opposite |

Returns: Returns type of step that is exact opposite

Example: `move_back=OppositeStep(last_step)`

## 17.72 GetSymmetrySquare

Returns central symmetric point

Syntax: `GetSymmetrySquare(wi)`

| | |
|---|---|
| `wi` | walker or spot index |

Returns: Returns walker placed on central symmetric square to the input

Notes: Returned walker has its default orientation

Example:
```
s=GetSymmetrySpot(w)
s=GetSymmetrySpot(_w(2,4))
```

## 17.73 WalkerTap

Moves walker based on recognized taps toward side that has been tapped

Syntax: `WalkerTap(&w,&up_down_flag)`

| | |
|---|---|
| `w` | walker to be moved |

Returns: Returns value that is related to the result of the move, also it fills up_down_flag by: -1 tap was from opposite side or 1 - tap was from the same side, note that at two above cases function returns 0 and no move has been proceed, it is up to user to decide what to do in such cases.

| | |
|---|---|
| `0` | no move, either taps has not been recognized or is not registered, or the tap was either from same or opposite side |
| `1` | walker moved 1 step towards the tap side |
| `2` | walker moved 1 step towards the tap side and went over the edge |

Notes: For proper work of this function, all sidetaps must be registered, because inside this function is motion output is read. Also walker itself is being turned to look towards the tap!

Example: `result=WalkerTap(w,up_down)`

See also: `WalkerDirUp`

## 17.74 WalkerGetDir

Retreive direction vector from walker

Syntax: `WalkerGetDir(wlk,vect[3])`

| | |
|---|---|
| `wlk` | walker for examination |
| `vect[3]` | vector of size 3, where to store direction vector |

Notes: This function unpack direction vector from the structure of walker.

Example: `WalkerGetDir(wlk,dir)`, retrieves direction vector form wlk into dir

See also: `WalkerSetDir`

## 17.75 WalkerSetDir

Tries to apply to walker given direction vector

Syntax: `res=WalkerSetDir(&wlk,vect[3])`

| | |
|---|---|
| `wlk` | walker or index to be modified |

Returns: Returns 1 if operation was successful or 0 if the operation failed, that means for example that this direction vector can not be forced into current spot.

Notes: None all of the direction vectors can be forced to every spot over the cube, that is one should be careful to using this function and also that is why this function returns 0 when the operation fails.

Example: `new vect[3]=[1,0,0]; WalkerSetDir(wlk,vect)`

See also: `WalkerGetDir`

## 17.76 WalkerGetNorm

Retreive normal vector from walker

Syntax: `WalkerGetNorm(wlk,vect[3])`

| | |
|---|---|
| `wlk` | walker for examination |
| `vect[3]` | vector of size 3, where to store normal vector |

Notes: This function unpack normal vector from the structure of walker.

Example: `WalkerGetNorm(wlk,norm)`retrieves normal vector form wlk into norm

## 17.77 WalkerDirUp

Modifies walker direction to look up according to accelerometer output

Syntax: `WalkerDirUp(&w,all_dirs=1,treshold=50,past=0)`

| | |
|---|---|
| `w` | walker to be modified |
| `all_dirs` | if set to 1, walker direction will be update all around, if set to 0, only turns to left and right will be allowed |
| `treshold` | gives minimum threshold for gravity, which is usually applied on top side |
| `past` | how old accelerometer data to use |

Returns: This function returns value that represent what happen to walker.

| | |
|---|---|
| `0` | no turn and acc data below threshold |
| `1` | no turn but acc over threshold |
| `2` | turn according to all_dirs |

Notes: This function works with accelerometer data, so it is possible to add threshold that must be overcome to perform the turn. This is useful is the change of direction is happening on the top side, where the biggest acceleration is masked.

Example: `WalkerDirUp(w)`, update walker w

`WalkerDirUp(w,0,100)`, update walker w only to turn right, left with threshold 100

See also: `WalkerTap`

## 17.78 WalkerCompareDir

Campares direction vectors of two walkers

Syntax: `WalkerCompareDir(wa,wb)`

| | |
|---|---|
| `wa` | walker a to compare with walker b |
| `wb` | walker b to compare with walker a |

Returns: Returned number reflects relation between two direction vectors

| | |
|---|---|
| `0` | Direction vectors are perpendicular |
| `1` | Directions are same |
| `-1` | Directions are opposite |

Example: `WalkerCompareDir(wlka,wlkb)`

## 17.79 WalkerBuddy

Check walker neighborhood and step suggestion

Syntax: `WalkerBuddy(wa,wb,&step)`

| | |
|---|---|
| `wa` | source walker for neighborhood check |
| `wb` | walker to check if in neighborhood |
| `step` | place where suggested step will be stored |

Returns: This function returns value that represent relation between wa and wb.

| | |
|---|---|
| `0` | wa is on same spot as wb |
| `1` | wa must do perpendicular step to reach wb |
| `2` | wa must do diagonal step to reach wb |
| `3` | wa is too far to reach wb in one signle step |

Notes: This function checks if wb is in close neighborhood to wa. It also suggest step which wa can perform to reach spot if wb. In case wb is too far away form wa number 3 is returned and step is filled by STEP_NONE.

Example: `if (WalkerBuddy(wa,wb,step)<3) {...}`, if wb close to wa do ...

`WalkerBuddy(wa,wb,step)`, if wb close to wa, put the step into step

See also: `WalkerTap`

## 17.80 Play

Plays sound resource from file system

Syntax: `Play(song{})`

| | |
|---|---|
| `song{}` | name of the the sound resource from file system |

Notes: This function starts to play given file name immediately on first free audio channel. If no channel is free, it override the channel, that has the oldest track to play

Example: `Play("myvoice")`

See also: `SetAudioForce`

## 17.81 SetAudioForce

Enables forcing audio to channel 0

Syntax: `SetAudioForce(value)`

`value` 1 - enables audio force, 0 - disable audio force

Notes: This function can enable audio force on channel zero, after enabling, channel zero is not used for overriding when new play command is invoked. It is used to force play audio data at situation where more resources are mixed and one sound must be completely played without interruption. To use channel zero with forcing function Play must have in its song name at the beginning character "@"

Example:
```
SetAudioForce(1)
Play("@forcesong")
```

See also: `Play`

## 17.82 SetVolume

Sets audio playback volume

Syntax: `SetVolume(value)`

`value` level of volume from range <0,32768>

Notes: This overrides current volume settings. Recommended maximum number is 20000

Example: `SetVolume(5000)`

See also: `Play`

## 17.83 Melody

Play ring tone melody

Syntax: `Melody(song{})`

`song{}` string array containing ring tone melody

Notes: This plays standard ring tone melody, which is automatically generated and system uses Play() for sending notes to output

Example: `mel="name:d=4,o=5,b=125:p,8p,16b,16a,b"`

Example: `Melody(mel)`, starts to play mel

See also: `Play`

## 17.84 WaitPlayOver

Wait until any play in progress is over

Syntax: `WaitPlayOver()`

Notes: This function cooperatively blocks the program until current playback is over.

Example: `WaitPlayOver()`

See also: `WaitMelodyOver`

## 17.85 WaitMelodyOver

Wait until current melody in progress is over

Syntax: `WaitMelodyOver()`

Notes: This function blocks the program until current melody playback is over. It differs from previous function, because in melody playback can be played pauses, which would appear as the playback is over in case the previous function would be used

Example: `WaitMelodyOver()`

See also: `WaitPlayOver`

## 17.86 IsPlayOver

Check is playback is in progress

Syntax: `IsPlayOver()`

Notes: This function returns 1 if there is no playback in progress, otherwise returns 0.

Example: `if (IsPlayOver()) {...}`

See also: `IsMelodyOver`

## 17.87 IsMelodyOver

Check is generation of ring tone melody is in progress

Syntax: `IsMelodyOver()`

Notes: This function returns 1 if there is no ring tone generation in progress, otherwise returns 0.

Example: `if (IsMelodyOver()) {...}`

See also: `IsPlayOver`

## 17.88 Quiet

Stops all current playback with melody generation.

Syntax: `Quiet()`

Notes: This function stops all ongoing playbacks and also melody generation.

Example: `Quiet()`

## 17.89 ReadAcc

Retreive accelerometer game data

Syntax: `ReadAcc(data[3],past=5)`

| | |
|---|---|
| `data` | 3 cell size vector for storing acc data |
| `past` | how deep in acc buffer reach into history <0,50> |

Notes: When there is any motion like tap or so, it is much better to take data before this even happened, because they reflects to situation when user wanted to do some action. recommended and standard value is 4, that means data are 32 ms old.

Example: `ReadAcc(data)`, reads 32 ms old acceleration data

`ReadAcc(data,0)`, reads newest acceleration data

See also: `GetCursor`

## 17.90 GetCursor

Retreiveing of gravity cursor

Syntax: `cursor=GetCursor(past=4)`

| | |
|---|---|
| `past` | tells how old data from accelerometer used to cursor computation |

Returns: This function return walker on spot that relates to gravity cursor with default orientation

Example: `cusor=GetCursor()`, standard cursor

`cusor=GetCursor(0)`, cursor from newest data

See also: `ReadAcc`

## 17.91 GetMsecs

Miliseconds from system start

Syntax: `res=GetMsecs()`

Returns: Number of milliseconds from system start

Example: `res=GetMsecs()`

See also: `GetAppMsecs`

## 17.92 GetAppMsecs

Miliseconds from application start

Syntax: `res=GetAppMsecs()`

Returns: Number of milliseconds from application start

Example: `res=GetAppMsecs()`

See also: `GetMsecs`

## 17.93 StartGameMenu

Start of the game menu

Syntax: `StartGameMenu()`

Notes: This function immediately starts game menu.

Example: `StartGameMenu()`

## 17.94 SetRndSeed

Set random seed into random generator.

Syntax: `SetRndSeed(seed)`

| | |
|---|---|
| `seed` | seed to random generator |

Example: `SetRandomSeed(500)`

See also: `GetRnd`

## 17.95 GetRnd

Read value from random generator

Syntax: `GetRnd(size)`

`size` defines output size

Returns: Returns (value%size)

Notes: This function returns value from random generator with its modulo operation over given size

Example: `GetRnd(3)`, gives back numbers in range <0,2>

See also: `SetRndSeed`

## 17.96 SetRandomizeFlag

Enables or disables randomizing by accelerometer

Syntax: `SetRandomizeFlag(flag)`

`flag` 1 - enables randomizing (default), 0 - disables randomizing

Notes: Regulary the random number generator is irregularly read according lowest bits at accelerometer output. But at some case is necessary that generator is predictable, in this is by disabling randomizing by accelerometer results in stable sequence of data from same random seed.

Example: `SetRandomizeFlag(1)`

See also: `GetRnd, SetRndSeed`

## 17.97 SetStillClick

Enables or disables still_click feature

Syntax: `SetStillClick(value)`

`value` 1 - enables, 0 - disables (default)

Notes: Still_click feature if enabled, waits for cube to be still, before any motion is recognized. In such case, user usually not do any phantom "clicks" when for example transfers cube from hand to hand, but before tapping, he must hod the cube very still.

Example: `SetStillClick(1)`

## 17.98 SCORE definition

Defined values for score types.

| | |
|---|---|
| SCORE_NORMAL | 0 |
| SCORE_WINNER | 1 |
| SCORE_LOSER | 2 |

## 17.99 Score

Display and announce reached score

Syntax: `Score(score,flag=SCORE_NORMAL,voice=1,double_tap=1)`

| | |
|---|---|
| `score` | actual score to display <0,999> |
| `flag` | score color version type from SCORE definition |
| `voice` | if set to 1, score is announced |
| `double_tap` | if set to 1, function is waiting for double tap to restart and also voice "double tap to restart the game" is announced |

Notes: This function draws score on the cube without using standard drawing method. It is recommended to use it at the end of game without any other drawing. Color scheme is selected by score definition type. It can announce the score by voice and it also can wait for double tap to restart the game. To draw the score in standard way, see the example bellow.

| | |
|---|---|
| SCORE_NORMAL | `blue score - middle range` |
| SCORE_WINNER | `colorfull score - high range` |
| SCORE_LOSER | `red score - low range` |

Example: `Score(199)`, draw 199 in NORMAL colors, announces the score and waits for double tap

## 17.100 DrawScore

Example of drawing score at regular way

```
//this function draws up to three digit number, same as score
draw_score(side,number)
{
 new walker=_w(side,0)  //creates walker with default direction at square 0
 new i
 new h=number/100
 number%=100
 new t=number/10
 number%=10
```

```
 new o=number

 if (h) DrawDigit(walker,h)
 for (i=0;i<3;i++) WalkerMove(walker,STEP_BACKWARDS)
 if (h || t ) DrawDigit(walker,t)
 for (i=0;i<3;i++) WalkerMove(walker,STEP_RIGHT)
 DrawDigit(walker,o)
}
```

## 17.101 DrawDigit

Draw digit with font 3x3

Syntax: `DrawDigit(w,digit)`

| | |
|---|---|
| `w` | walker from where it starts to draw line by line |
| `digit` | digit to draw <0..9> |

Notes: This function draws digit by preset color and attributes. Drawing starts from the walker position and continues to the right and down. Font is 3x3.

Example: `DrawDigit(wlk,5)`

## 17.102 IsGameResetRequest

Game reset request from user

Syntax: `IsGameResetRequest()`

Notes: Function returns positive value if during script start user tapped three times to the icon. This function is usually used with puzzles that stores their state in variables.

Example: `if (IsGameResetRequest()) {...init game...}`

## 17.103 Vibrate

Vibrate for given amount of milliseconds

Syntax: `Vibrate(msec=100)`

| | |
|---|---|
| `msec` | number of milliseconds for vibration |

Notes: Maximum number of milliseconds is 5000. Each time this function is called, vibration is block for double amount of time. Vibrate(150) blocks another call within next 300 milliseconds.

Example: `Vibrate(150)`, vibrates for 150 milliseconds

## 17.104 ICON magics

Defined values for ICON magics

| | |
|---|---|
| ICON_MAGIC1 | 0x78948396 |
| ICON_MAGIC2 | 0xAABBEEDD |

## 17.105 ICON

Insertion of icon and sound information into code

Syntax: `ICON(icon[])`

| | |
|---|---|
| `icon` | array that defines icon and sound resources |

Notes: This function uses array icon and therefore it is not removed form the code during optimization. Array for icon must have special format to be recognized by the cube. Icon itself is colorful icon that replaces selected game at the game menu and by its taping, user script selected and started. Icon has mandatory length and MAGIC fields. Additionally into icon array are added two names of sound file that are used as name and explanation.

| | |
|---|---|
| ICON_MAGIC1 | icon MAGIC1 |
| ICON_MAGIC2 | icon MAGIC2 |

| | |
|---|---|
| `menu` | menu number <0..2>, on which game menu shall the icon appear, blue menu is menu number 0 |
| `side` | absolute side <0..5>, on which side at selected menu the icon shall appear |
| `9 cells` | nine colors that represent the icon on 3x3 matrix |
| `name` | unpacked name of sound file that represents the name of the script in menu |
| `expl` | unpacked name of sound file that represents the explanation of the script in menu |

```
...
icon[]=[ICON_MAGIC1,ICON_MAGIC2,1,1,
        RED,BLUE,RED,BLUE,RED,BLUE,GREEN,GREEN,GREEN,
        ''INSTMR2'',''UFO'']
...
icon[]=[ICON_MAGIC1,ICON_MAGIC2,1,1,
        RED,BLUE,RED,BLUE,RED,BLUE,GREEN,GREEN,GREEN,
        '''','''']

ICON(icon)
...
```

Notes: Second example shows case when sounds resources are not setup. If you do not want to have them, you should add there empty unpacked strings for correct operation. NOTE THAT ARRAY "icon" MUST BE DEFINED IN GLOBAL NAME SPACE!!! Otherwise the icon might be wrongly recognized in bytecode.

`ICON(icon)`, icon is used and will be recognized in byte code by the cube

## 17.106 PrintArray

Debug function, print array values to terminal

Syntax: `PrintArray()`

Example: `PrintArray()`

See also: `PrintCnv, printf`

## 17.107 VARIABLE magics

Defined values for VARIABLE magics

| | |
|---|---|
| `VAR_MAGIC1` | `0x12AAF0B5` |
| `VAR_MAGIC2` | `0x190203BD` |

## 17.108 RegisterVariable

Register named variable array to notify system to keep the variable active.

Syntax: `RegisterVariable(var[])`

| | |
|---|---|
| `var` | array that contains VAR_MAGIC1, VAR_MAGIC2 and variable name, variable name must be longer than 6 characters and shorter than 24 characters |

Notes: Variable array must have specified format in order to be correctly registered, if the variable is not registered, it will be erased from the system ad unused one. NOTE THAT ARRAY "var" MUST BE DEFINED IN GLOBAL NAME SPACE!!! Otherwise the variable might be wrongly recognized in bytecode.

```
...
var[]=[VAR_MAGIC1,VAR_MAGIC2,''my_variable_name'']
RegisterVariable(var)
...
```

See also: `StoreVariable, LoadVariable`

## 17.109 StoreVariable

Stores given array as named variable

Syntax: `StoreVariable(name[],arr[],size=sizeof arr)`

| | |
|---|---|
| `name` | name of the registered variable |
| `arr[]` | array af data to be stored |
| `size` | size of the array, maximum is 501 cells |

Notes: Variable is store into non-volatile memory over buffer in RAM. As long as the script is using same one variable, it can store it as many times as needed. Write into non-volatile memory is performed when different variable is used or if the cube is switched off. One script should not use more than one variable. If does so, number of writing to non-volatile memory is limited to 5/5secs and total limit 20/script.

Example: `StoreVariable(''my_variable_name'',data)`, stores data as named variable

See also: `LoadVariable, RegisterVariable`

## 17.110 LoadVariable

Load data of named variable to given array

Syntax: `LoadVariable(name[],arr[],size=sizeof arr)`

| | |
|---|---|
| `name` | name of the registered variable |
| `arr[]` | destination array |
| `size` | size of the array, maximum is 501 cells |

Returns: This function returns 1 if variable has been loaded or 0 if there is problem or variable does not yet exists.

Example: `LoadVariable(''my_variable_name'',data)`, load named variable to data

See also: `SaveVariable, RegisterVariable`

## 17.111 Restart

Restart current script

Syntax: `Restart()`

Example: `Restart()`, script will be restarted

## 17.112 CollisionTest

Test two arrays for collisions

Syntax: `CollisionTest(source1[],source2[],dest[],val=1;sizes1=sizeof source1,sizes2=sizeof source2,sized=sizeof dest)`

| | |
|---|---|
| `source1` | first source array for collision test |
| `source2` | second source array for collision test |
| `dest` | destination array to store result from collision test |
| `val` | value indicating collision to be stored into destination array |
| `sizes` | sizes of all arrays that must be same |

Returns: Function returns number of collision points.

Notes: Destination array is cleared prior the operation. Collision is performed by easy condition if (source1[index] && source2[index]) {dest[index]=val}

Example: `if (CollisionTest(a,b)) {....}`, test number of collision points with using dest

`if (CollisionTest(a,b,dest,WHITE)) {....}`, test number of collision points and stores WHITE into dest

## 17.113 ApiVer

Version of current API

Syntax: `res=ApiVer()`

Returns: Actuall version of API

Example: `res=ApiVer()`, retrieve version of API

## 17.114 Useful macros

Useful defined macros

```
INDEX_MASK          0x000000FF, mask for retrieving index
_i(val)             (val&INDEX_MASK), index macro
_is(data,bit)       (data&(1<<bit)), test bit macro
_side(index)        (index/9)
_square(index)      (index%9)
abs(val)            ((val) < 0 ?  -(val) :  (val))
min(a,b)            (a < b ?  a :  b)
max(a,b)            (a > b ?  a :  b)
```

### 17.115 Macros examples

Typical situation where macros are used

```
playground[_i(GetCursor())]=1          //conversion of cursor
                                       //square to array index

playground[_i(walker)]=1               //conversion of walker
                                       //square to array index

if (_side(prev)!=_side(GetCursor())    //test for different sides

if (_i(prev)!=_i(GetCursor())          //test for different indexes
                                       //of squares

if (_square(prev)!=_square(GetCursor()) //test for different squares
                                       //numbers

w=_w(_side(GetCursor()),4)             //center walker w at side
                                       //given by cursor

if (_is(motion,TAP_DOUBLE))            //test if double tap flag
                                       //is set up
```

# 18 Release notes

## 18.1 SDK manual

Release notes for SDK maual document version 1.4

| | |
|---|---|
| + | multiple script support from FW 4.5 and RFC 0.8 |
| - | removed "setpawn" and "setpawnstd" commands, that become obsolete for FW 4.5 |
| + | new command "setpawnauto" that replaces previous two mentioned for FW 4.5 |
| + | new command "pdir" that lists all installed scripts for FW 4.5 |
| + | new commands "pawneraseflash" and "pawnerasemycube" for FW 4.5 |

Release notes for SDK maual document version 1.3

| | |
|---|---|
| + | varpawnrease changed to varpawnerase |

## 18.2 futurocube.inc

Release notes for futurocube.inc version 1.2

| | |
|---|---|
| * | No change in API |
| - | Removed refferences to WalkerInit, it was previous version of _w |
| + | SCORE_LOOSER changed to SCORE_LOSER |
| + | Updated info about Score and added example how to draw score in regular way |
| + | Added macros examples |
| + | Added important note to ICON and RegisterVariable |