



MADRIX 2 Script Help and Manual

© 2014 inoage GmbH

MADRIX Version: 2.14h

MADRIX Script Version: 1.43.

November 2014

Table of Contents

Part I What is New	6
Part II MADRIX Script (Introduction)	31
1 Getting Started.....	31
2 Working With The Script Editor.....	34
3 Basics.....	38
Writing A Script	38
Syntax Highlighting	39
Identifiers	41
Functions	42
Data Types & Variables	46
Using Variables.....	47
Using Data Types.....	50
Conversion Between Data Types.....	52
Fields	53
Strings & String Operations	56
Expressions	62
Statements	67
'If' & 'Else If' Statements	68
'Switch' Statements	71
'For' & 'While' Loops.....	74
Reading From External Files	78
Using Comments	82
Including Extra Information	82
4 Advanced Techniques.....	83
Draw And Render Functions	83
Pixels Vs. Vectors.....	84
Using Colors.....	85
Predefined Colors.....	92
Using Filters	93
'ShiftMatrix'.....	96
'Draw PixelArea'.....	97
'PixelTranspose'.....	101
'SetPixel'	107
Manipulating Effects	110
Map An Effect.....	111
Tile An Effect.....	115
Mix Modes.....	117
Sound2Light & Music2Light	119
Sound2Light (S2L).....	119
Music2Light (M2L).....	123
Part III MADRIX Script (Programming Language Overview)	128
1 Keyword Search.....	128

2	List Of Functions (Alphabetical Order).....	128
3	List Of Functions (Grouped).....	144
4	List Of Global Variables And Constants.....	155
5	List Of Operations.....	161
6	List Of Structures.....	162
7	Table Of Frequencies.....	164
8	Table Of Notes.....	170
9	Examples.....	171

Part IV MAS Script Effect 185

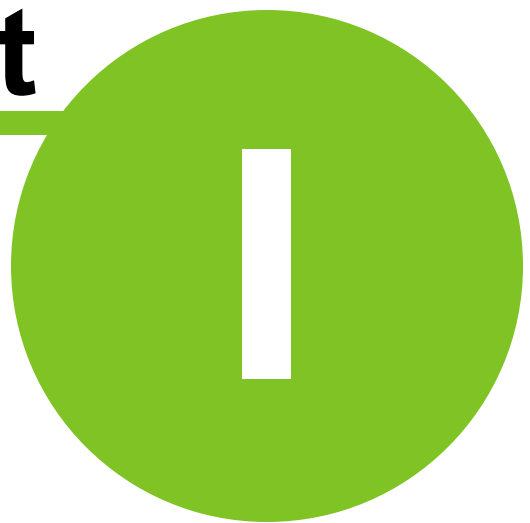
1	Overview.....	185
2	Functions.....	188
3	Using GUI Elements (User Interaction).....	190
4	Controlling The Tempo.....	209
	Increasing The Speed Of Effects	209
	Controlling A Script Via Frame ID	213
	Using A Fixed Render Frequency	216

Part V Macros For Effects 220

1	Overview.....	220
2	Functions.....	223
3	SCE Static Color Effects.....	226
	SCE Color	226
	SCE Bitmap	228
	SCE Bounce	232
	SCE Color Change	234
	SCE Color Fill	234
	SCE Color Ramp	238
	SCE Color Scroll	240
	SCE Drops	243
	SCE Explosions	246
	SCE Fire	250
	SCE Graph	254
	SCE Metaballs	258
	SCE Plasma	263
	SCE Pulse / Stroboscope	264
	SCE Radial	265
	SCE Shapes	270
	SCE Starfield	274
	SCE Ticker	276
	SCE Video	281
	SCE Wave	285
4	S2L Sound2Light Effects.....	287
	S2L Equalizer	287
	S2L EQ Drops	291
	S2L EQ Shapes	292
	S2L EQ Tubes	296
	S2L Frequency Flash	297

S2L Level Color	299
S2L Level Meter	300
S2L Level Ring	302
S2L Waveform	305
S2L Wavegraph	307
5 M2L Music2Light Effects.....	308
M2L Color Fade	308
M2L Color Rings	309
M2L Color Scroll	310
M2L Interval Drops	312
M2L Interval Tubes	314
M2L Single Tone Shapes	316
Part VI Storage Place Macro	322
1 Overview.....	322
2 Functions.....	325
Part VII Main Output Macro	331
1 Overview.....	331
2 Functions.....	334
3 Examples.....	339
Part VIII Imprint & Copyright	355
Index	357

Part



1 What is New

The latest MADRIX Script version is 1.43 (MADRIX 2.14h).

News For Script Engine Version 1.43 (MADRIX V2.14h)

New functions for Main Output Macro:

```
int CuelistCount()
```

News For 1.40 (MADRIX V2.14b)

New global functions:

```
int IsMidiInEnabled()
int GetMidiInNoteValue(int note, int midichannel, int device)
int GetMidiInControlValue(int control, int midichannel, int device)
void GetMidiInNote(int midivalues[], int startnote, int notecount, int midichannel, int device)
void GetMidiInControl(int midivalues[], int startcontrol, int controlcount, int midichannel, int device)
void Dim(float value)
void DimPixel(float value, int x, int y)
void DimPixelArea(float value, int x, int y, int width, int height)
```

New functions for Main Output Macro:

```
int ImportPatch(string name)
```

Updated Information:

```
void GetDmxIn(int DmxValues[], int startchannel, int channels, int universe)
int GetDmxInChannel(int channel, int universe)
int GetFilter()
void SetFilter()
```

```
void GetPixelArea(matrix[][], int xSrc, int ySrc, int w, int h, int xDst, int yDst)
void DrawPixelArea(matrix[][], int xDst, int yDst, int w, int h, int xSrc, int ySrc, color filter)
```

News For 1.39 (MADRIX V2.14a)

New functions for SCE Ticker:

```
void SetContinuous(int enable)
int GetContinuous()
void SetReverseWords(int enable)
int GetReverseWords()
```

Deprecated functions for SCE Ticker:

```
void SetContinous(int enable)
int GetContinous()
void SetReverseWord(int enable)
int GetReverseWord()
```

New functions (Macro) for the SCE Graph:

```
void SetContinuous(int enable)
int GetContinuous()
```

New functions for SCE Color Ramp:

```
void AddColor(color c, float position, int fade)
void RemoveColor(int index)
int SetColorPosition(int index, float position)
float GetColorPosition(int index)
void SetColorFade(int index, fade)
int GetColorFade(int index)
void FadeAllColors()
void FadeNoneColors()
```

void SetUniformDistances()

void InvertColorPositions()

void InvertColors()

Deprecated functions for SCE Color Ramp:

int SetPosition(int index, float pos)

float GetPosition(int index)

void SetFade(int index)

int GetFade(int index)

New Functions for SCE Plasma:

void SetColor(int idx, color c)

color GetColor(int idx)

int GetColorCount()

void AddColor(color c, float position, int fade)

void RemoveColor(int index)

int SetColorPosition(int index, float position)

float GetColorPosition(int index)

void SetUniformDistances()

void InvertColorPositions()

void InvertColors()

New functions for S2L Waveform:

void SetColor(int idx, color c)

color GetColor(int idx)

int GetColorCount()

void AddColor(color c, float position, int fade)

void RemoveColor(int index)

int SetColorPosition(int index, float position)

float GetColorPosition(int index)

void SetUniformDistances()

void InvertColorPositions()

void InvertColors()

New global functions:

time GetTimeCode()

New global defines:

CYAN, TURQUOISE, PINK, MAGENTA, SILVER, DARK_GRAY, ORANGE, BROWN, SKY, GOLD,
WHITE_ALPHA, BLACK_ALPHA

News For 1.38 (MADRIX V2.14)

New functions (Macro) and defines for the SCE Graph effect:

void SetBpm(int bpm)

int GetBpm()

void SetDirection(int direction)

int GetDirection()

void SetHeight(int value)

int GetHeight()

void SetHeightMax(int value)

int GetHeightMax(int value)

void SetWidth(int value)

int GetWidth()

void SetWidthMax(int value)

int GetWidthMax()

void SetPitch(int value)

int GetPitch()

void SetPitchMax(int value)

int GetPitchMax()

void SetFrequency(int index, float value)

float GetFrequency(int index)

```
void SetFrequencyMax(float value)
float GetFrequencyMax()
void SetPeak(int index, int value)
int GetPeak(int index)
void SetShape(int shape)
int GetShape()
void SetColorMode(int mode)
int GetColorMode()
void SetModeHeight(int mode)
int GetModeHeight()
void SetModeWidth(int mode)
int GetModeWidth()
void SetModePitch(int mode)
int GetModePitch()
void SetModeFrequency(int mode)
int GetModeFrequency()
void SetText(string text)
string GetText()
void SetRotation(int angle)
int GetRotation()
void SetFontWidth(int value)
int GetFontWidth()
void SetFontHeight(int value)
int GetFontHeight()
void SetFontItalic(int value)
int GetFontItalic()
void SetFontUnderline(int value)
int GetFontUnderline()
void SetFontStrikeOut(int value)
int GetFontStrikeOut()
void SetFontWeight(int value)
int GetFontWeight()
void SetFontFaceName(string name)
string GetFontFaceName()
void SetMode(int mode)
```

```
int GetMode()
void SetGraphMode(int index, int mode)
int GetGraphMode(int index)
```

```
MODE_SINE, MODE_COSINE, MODE_TRIANGLE, MODE_SQUARE, MODE_NONE, MODE_UNIFORM,
MODE_LINEAR, MODE_QUADRATIC, MODE_SQRT, MODE_CUBIC, MODE_RANDOM, DRAW_RECT,
FILL_RECT, DRAW_CIRCLE, FILL_CIRCLE, DRAW_CROSS, DRAW_STAR, DRAW_DIAMOND,
FILL_DIAMOND, DRAW_RANDOM, DRAW_TEXT, CM_LOOP, CM_SHUFFLE, CM_RANDOM,
MODE_SENTENCE, MODE_WORD, MODE_CHAR
```

New functions for Macros for Effects:

```
void SetSolo(int)
int GetSolo()
void SetBlind(int)
int GetBlind()
```

New functions for the Main Output Macro:

```
void ImportStoragePlace(int storage, int place, string name)
void ImportStorage(int storage, string name)
```

News For 1.35 (MADRIX V2.13b)

New global functions:

```
void SetVectorPixel(color, float x, float y)
color GetVectorPixel(float x, float y)
```

Updated functions:

```
void GetDmxIn(int DmxValues[], int startchannel, int channels, int universe)
int GetDmxInChannel(int channel, int universe)
int ReadAsync(string file, string txt) (FILE_OK)
```

News For 1.33 (MADRIX V2.13)

New global functions and defines:

`void TRACE(variable)`

`FILTER_NONE, FILTER_INVERT_H_MATRIX, FILTER_INVERT_V_MATRIX, FILTER_INVERT_HV_MATRIX`
(Filters)

New functions for the Main Output Macro:

`void SetStorageFilter(int storage, int filter)`

`int GetStorageFilter()`

New defines for the Main Output Macro:

`HWIPEFADE, VWIPEFADE, HXWIPEFADE, VXWIPEFADE, HSLIDEFADE, VSLIDEFADE, HXSLIDEFADE, VXSLIDEFADE` (Function `SetFadeType`)

New functions for the Storage Place Macro:

`void SetFilter(int filter)`

`int GetFilter()`

`void SetLayerFilter(int number, int filter)`

`int GetLayerFilter(int number)`

New functions for Macros for Effects:

`void SetFilter(int filter)`

`int GetFilter()`

New or improved functions for the SCE Color Fill effect:

`void SetWidth(int size)`

`int GetWidth()`

```
void SetPitch(int pitch)
int GetPitch()
void SetShape(int shape)
int GetShape()
void SetMirror(int value)
int GetMirror()
void SetCircle(int value)
int GetCircle()
void SetCenter(int value)
int GetCenter()
```

News For Script Engine Version 1.30 (MADRIX V2.12a)

New functions for MADRIX Script:

```
void DrawPixelDiamond(color col, int x, int y, int w, int h)
void DrawVectorDiamond(color col, float x, float y, float w, float h)
void FillPixelDiamond(color col, int x, int y, int w, int h)
void FillVectorDiamond(color col, float x, float y, float w, float h)
```

New functions and defines for the SCE Color Fill effect:

```
void SetWidth(int size)
int GetWidth()
void SetPitch(int pitch)
int GetPitch()
void SetShape(int shape)
int GetShape()
MODE_RANDOM, MODE_DROPS, MODE_SNAKE, MODE_FLAT, MODE_COLLAPSE, MODE_TETRIS
```

News For Script Engine Version 1.29 (MADRIX V2.12)

New functions for MADRIX Script:

```
void DrawPixelText(color c, font f, string t, int x, int y, int rotation)
```

`void DrawVectorText(color c, font f, string t, float x, float y, int rotation)`

New structure:

`font`

News For Script Engine Version 1.28 (MADRIX V2.11)

New functions (Macro) and defines for the SCE Explosions effect:

```
void SetBpm(int bpm)
int GetBpm()
void SetExplosionSize(int size)
int GetExplosionSize(void)
void SetShapeSize(int size)
int GetShapeSize(void)
void SetGravity(float gravity)
float GetGravity(void)
void SetFadeOut(int fadeout)
int GetFadeOut(void)
void SetShapeCount(int count)
int GetShapeCount(void)
void SetRocketCount(int count)
int GetRocketCount(void)
void SetBlur(int enable)
int GetBlur(void)
void FireRocket(int posX, int posY, int explPosX, int explPosY, int ParticleCtn, int explSize, int
explShape, int drawShape, color Col, color sparkleCol)
void Detonate(int explPosX, int explPosY, int ParticleCtn, int explSize, int explShape, int
drawShape, color Col, color sparkleCol)
void SetExplosionMode(int mode)
int GetExplosionMode(void)
void SetExplosionShape(int shape)
int GetExplosionShape(void)
void SetDrawShape(int shape)
```

```
int GetDrawShape(void)
void SetColorMode(int mode)
int GetColorMode(void)
int GetColorCount()
void RemoveColor(int idx)
void SetColor(int idx, color c)
color GetColor(int idx)
void AddColor(int idx, color c)
void SetSparkleColorMode(int mode)
int GetSparkleColorMode(void)
int GetSparkleColorCount(void)
void RemoveSparkleColor(int )
void SetSparkleColor(int idx, color c)
color GetSparkleColor(int idx)
void AddSparkleColor(int idx, color c)
```

```
MODE_EXPLOSIONS, MODE_FIREWORKS, EXPLOSION_SHAPE_SPHERE,
EXPLOSION_SHAPE_SPHERE_GLOW, EXPLOSION_SHAPE_SPIRAL, EXPLOSION_SHAPE_RADIAL,
EXPLOSION_SHAPE_DIAMOND, EXPLOSION_SHAPE_STAR, EXPLOSION_SHAPE_RANDOM,
DRAW_RECT, FILL_RECT, DRAW_CIRCLE, FILL_CIRCLE, DRAW_CROSS, DRAW_STAR, DRAW_LINE,
DRAW_DIAMOND, FILL_DIAMOND, DRAW_RANDOM, CM_LOOP, CM_SHUFFLE, CM_RANDOM
```

New functions (Macro) and defines for the SCE Starfield effect:

```
void SetBpm(int bpm)
int GetBpm()
void SetLength(int size)
int GetLength()
void SetCount(int count)
int GetCount()
void SetWidth(int width)
int GetWidth()
void SetDepth(int depth)
int GetDepth()
void SetRotation(int rotation)
int GetRotation()
```

```
void SetDirection(int dir)
int GetDirection()
void SetColorMode(int mode)
int GetColorMode()
void SetShape(int shape)
int GetShape()
```

```
DRAW_RECT, FILL_RECT, DRAW_CIRCLE, FILL_CIRCLE, DRAW_CROSS, DRAW_STAR, DRAW_LINE,
DRAW_DIAMOND, FILL_DIAMOND, DRAW_RANDOM, DIR_OUTWARDS, DIR_INWARDS, CM_LOOP,
CM_SHUFFLE, CM_RANDOM
```

New functions (Macro) and defines for the SCE Metaballs effect:

```
void SetColorMode(int mode)
int GetColorMode()
int GetColorCount()
void RemoveColor(int idx)
void SetColor(int idx, color c)
color GetColor(int idx)
void AddColor(int idx, color c)
void SetColorMix(int mode)
int GetColorMix()
void SetSharpness(int mode)
int GetSharpness()
void SetColorMixLink(int value)
int GetColorMixLink()
```

```
CM_LOOP, CM_SHUFFLE, CM_RANDOM, MODE_CIRCLE, MODE_RECTANGLE, MODE_DIAMOND,
MODE_VERY_BLURRY, MODE_BLURRY, MODE_SLIGHTLY_BLURRY, MODE_MEDIUM,
MODE_SLIGHTLY_CLEAR, MODE_CLEAR, MODE_VERY_CLEAR
```

News For Script Engine Version 1.27 (MADRIX V2.10)

New universal functions:

```
string GetComputerName()
```



```
string GetUserName()  
void SetValid()  
void SetInvalid()
```

New functions and defines for the Main Output Macro:

```
void SetBlackout(int mode)  
int GetBlackout()
```

New functions for the SCE Ticker / Scrolling Text effect:

```
int GetFontWidth()  
void SetFontWidth(int width)  
int GetFontHeight()  
void SetFontHeight(int height)  
int GetFontItalic()  
void SetFontItalic(int val)  
int GetFontUnderline()  
void SetFontUnderline(int val)  
int GetFontStrikeOut()  
void SetFontStrikeOut(int val)  
int GetFontWeight()  
void SetFontWeight(int val)  
string GetFontFaceName()  
void SetFontFaceName(string)
```

New Defines for SCE Shapes, S2L EQ Shapes, M2L Single Tone Shapes:

```
DRAW_DIAMOND, DRAW_DIAMOND_EXPLODE, DRAW_DIAMOND_IMPLODE, FILL_DIAMOND,  
FILL_DIAMOND_EXPLODE, FILL_DIAMOND_IMPLODE
```

News For Script Engine Version 1.26 (MADRIX V2.9)

New functions for the SCE Wave effect:

```
void SetPeak(int peak)  
int GetPeak()
```

New functions and defines for the SCE Fire effect:

```
void SetMode(int mode)
int GetMode()
void SetFlameSize(int size)
int GetFlameSize()
void SetIntensity(int intensity)
int GetIntensity()
MODE_FIRE, MODE_FLAMES
```

New functions for effects that use the Color Ramp dialog:

```
void SetColor(int idx, color c)
color GetColor(int idx)
int GetColorCount()
int SetColorPosition(int idx, float position)
float GetColorPosition(int idx)
void SetColorFade(int idx, int mode)
int GetColorFade(int idx)
void FadeAllColors()
void FadeNoneColors()
void SetUniformDistances()
void InvertColorPositions()
void InvertColors()
void AddColor(color c, float position, int idx)
void RemoveColor(int idx)
```

New function for the Main Output Macro:

```
void SetColorFilter(color c)
color GetColorFilter()
```

News For Script Engine Version 1.25 (MADRIX V2.8a)

New universal function:

```
string GetApplicationPath()
string GetUserProfileDirectory()
string GetScriptEngineVersion()
string GetSoftwareVersion()
int CheckScriptEngineVersion(int major, int minor)
int CheckSoftwareVersion(int major, int minor, int subminor, int subsubminor)
```

New function for the Main Output Macro:

```
int CuelistCurrentCue()
```

News For Script Engine Version 1.24 (MADRIX V2.8)

New universal function:

```
void Filter(int filter)
```

New functions and defines for the SCE Metaballs effect:

```
void SeedRandom()
void SetMode(int mode)
int GetMode()
void SetScale(float value)
float GetScale()
MODE_UNIFORM, MODE_LINEAR, MODE_QUADRATIC, MODE_SQRT, MODE_CUBIC, MODE_RANDOM
```

News For Script Engine Version 1.23 (MADRIX V2.8 BETA)

New universal functions:

```
int IsDmxInEnabled()
int GetDmxIn(int map[], int startchannel, int channels)
int GetDmxInChannel(int channel)
int GetSubMaster()
void SetSubMaster(int value)
```

New functions for the Main Output Macro:

```
int GetStorageFullState(int storage, int place)
```

New effects with new functions:

SCE Drops

SCE Metaballs

New functions/defines for the S2L Equalizer effect:

void SetDrawMode(int mode)

int GetDrawMode()

void SetBandMode(int state)

int GetBandMode()

MODE_BAR, MODE_RADIAL, MODE_RADIAL_LINE, MODE_RADIAL_OUTLINE, MODE_RADIAL_DOT

A new function for the SCE Ticker effect:

string GetText()

Deprecated functions:

int GetSubmasterValue()

void SetSubmasterValue(int value)

New functionality of MADRIX Script, the Storage Place Macro:

float GetSpeedMaster()

void SetSpeedMaster(float value)

int GetPause()

void SetPause(int state)

int GetSubMaster()

void SetSubMaster(int value)

string GetDescription()

void SetDescription(string text)

int GetLayerCount()

int GetLayerSolo(int number)

void SetLayerSolo(int number, int state)

int GetLayerBlind(int number)

void SetLayerBlind(int number, int state)

int GetLayerSubMaster(int number)

void SetLayerSubMaster(int number, int value)

void EnableLayerFrameFade(int number, int enable)

int IsLayerFrameFadeEnabled(int number)

```
int GetLayerMixMode(int number)
void SetLayerMixMode(int number, int mix mode)
int GetLayerLink(int number)
void SetLayerLink(int number, int state)
void MapLayerEffectVector(int number, float x, float y, float w, float h)
void MapLayerEffectPixel(int number, int x, int y, int w, int h)
void GetLayerMapPixel(int number, int map[])
void GetLayerMapVector(int number, float map[])
void MapLayerTileEffectVector(int number, float x, float y, float w, float h)
void MapLayerTileEffectPixel(int number, int x, int y, int w, int h)
void SetLayerMapModeMirror(int number, int state)
void SetLayerMapModeTile(int number, int state)
void GetLayerMapTileEffectVector(int number, float map[])
void GetLayerMapTileEffectPixel(int number, int map[])
int IsLayerMapped(int number)
int GetLayerMapModeMirror(int number)
int GetLayerMapModeTile(int number)
void SetLayerVectorTileOffset(int number, float x, float y)
void SetLayerPixelTileOffset(int number, int x, int y)
void GetLayerVectorTileOffset(int number, float map[])
void GetLayerPixelTileOffset(int number, int map[])
void SetLayerVectorMapRotation(int number, float value, int anim)
void SetLayerDegreeMapRotation(int number, int value, int anim)
float GetLayerVectorMapRotation(int number)
int GetLayerDegreeMapRotation(int number)
int IsLayerMapRotation(int number)
float GetLayerFrameId(int number)
void SetLayerFrameId(int number, float value)
float GetLayerFrameSteps(int number)
float GetLayerFrameCount(int number)
int GetLayerOpacity(int number)
void SetLayerOpacity(int number, int value)
```

News For Script Engine Version 1.22 (MADRIX V2.7a)

New functions for the MAS Script effect, Macro, and Main Output Macro:

`void SetPixelGreyscale(int x, int y)`

New functions for Main Output Macro:

`int GetFadeType()`

`void SetFadeType(int fadetype)`

`color GetFadeColor()`

`void SetFadeColor(color)`

`float GetFadeTime()`

`void SetFadeTime(float fadetime)`

`int GetFadeValue()`

`void SetFadeValue(int fadervalue)`

`int GetFreeze()`

`void SetFreeze(int freeze/unfreeze)`

`int GetMasterFader()`

`void SetMasterFader(int masterfadervalue)`

`int GetAudioFader()`

`void SetAudioFader(int audiofadervalue)`

`void CuelistStop()`

`void CuelistPlay()`

`void CuelistGo()`

`void CuelistBack()`

`void CuelistGoto(int cuelistentry)`

`int GetStorageSpeedMaster(int storage)`

`void SetStorageSpeedMaster(int storage, float speed)`

`int GetStoragePause(int storage)`

`void SetStoragePause(int storage, int pause/nopause)`

`void GetStoragePlace(int storage)`

`void SetStoragePlace(int storage, int place, int autofade/noautofade)`

`int GetStorageSubMaster(int storage)`

`void SetStorageSubMaster(int storage, int submastervalue)`

News For Script Engine Version 1.21 (MADRIX V2.7 BETA)

New functions for Macro:

```
void SetLink(int enable)
int GetLink()
```

New Macro functions for SCE Ticker effect:

```
void SetSmooth(int enable)
int GetSmooth()
```

New Macro functions for SCE Color Ramp effect:

```
void SetColor(int index, color col)
color GetColor(int index)
int SetPosition(int index, float pos)
float GetPosition(int index)
void SetFade(int index, int enable)
int GetFade(int index)
int GetColorCount()
```

News For Script Engine Version 1.20 (MADRIX V2.6e)

New functions for the MAS Script effect, Macro, and Main Output Macro:

```
void CreatePixelTransposeTable(int size, int growsize)
void SetPixelTransposeEntry(int idx, int srcX, int srcY, destX, destY)
void AddPixelTransposeEntry(int srcX, int srcY, destX, destY)
void ExecutePixelTranspose(int clear)
void InvertMatrix()
void InvertColor()
void Greyscale()
```

News For Script Engine Version 1.19 (MADRIX V2.6d)

New functions (Macro) for SCE Radial effect:

```
void SetCurve(int curve)
int GetCurve()
```

```
void SetFactor(int factor)
int GetFactor()
void SetAmplitude(int amplitude)
void GetAmplitude()
```

News For Script Engine Version 1.18 (MADRIX V2.6c)

With reference to the new functionality of the **SCE Video** effect, there are three valid values for the mode 'function'. **NO_LOOP** for no looping, **F_LOOP** for only forward looping and **FB_LOOP** for forward and backward.

```
void SetLoop(int mode)
void StartVideoBackward()
void SetVideoStartTime(time t)
time GetVideoStartTime()
void SetVideoEndTime(time t)
time GetVideoEndTime()
void SetVideoPlaySpeed(float speed)
float GetVideoPlaySpeed()
void SetVideoAspectRatio(int mode)
```

News For Script Engine Version 1.17 (MADRIX V2.6a)

New functions for reading and modifying the opacity values for all effects.

```
int GetOpacity()
void SetOpacity(int value)
```

News For Script Engine Version 1.16 (MADRIX V2.6)

New functions (Macro) for the SCE Bounce effect.

```
void SetBpm(int value)
int GetBpm()
void SetObjects(int value)
int GetObjects()
void SetPoints(int value)
```



```
int GetPoints()
void SetFadeOut(int value)
int GetFadeOut()
void SetSize(int size)
int GetSize()
void SetShape(int shape)
int GetShape()
void SetCollision(int state)
int GetCollision()
```

News For Script Engine Version 1.15 (MADRIX V2.6 BETA)

New function `fmod(float denominator, float divisor)`, which is the modulo function for float values.

New direction parameters `SHIFT_C_IN_OUT` and `SHIFT_C_OUT_IN` for the `ShiftMatrix` function.

New functions for the new Map dialog of MADRIX

```
void SetVectorMapRotation(float value, int anim)
void SetDegreeMapRotation(int value, int anim)
float GetVectorMapRotation()
int GetDegreeMapRotation()
int IsMapRotation()
```

News For Script Engine Version 1.14 (MADRIX V2.5a)

There is a new `script information` called `name` to give a script a name.

For the `SCE Color Ramp` effect there is a new direction `DIR_RADIAL`, which could be used in a Macro, and two new functions `SetAngle(int angle)` and `GetAngle()` for Macro.

New functions `SetCurrentImage`, `GetCurrentImage` und `GetImageCount` for `SCE Bitmap available` to control the image list.

For effects with a color table (not the M2L color table) there are several new functions available to access the color table via Macro. This includes the following effects: [SCE ColorChange](#), [SCE ColorFill](#), [SCE ColorScroll](#), [SCE Shapes](#), [S2L Equalizer](#), [S2L EQ Shapes](#), [S2L LevelMeter](#), [S2L LevelRing](#) and [M2L SingleTones](#).

New [button \(GUI element\)](#) available for the MAS Script effect. Furthermore, a new [ctrledit3-element](#) is available. In addition to the [GUI elements ctrlslider](#), [ctrledit](#) and [ctrlcolortable](#), a tooltip has been added.

New mathematical functions [round](#), [ceil](#), and [trunc](#) have been added.

New functions [DrawPixelArea](#) and [GetPixelArea](#) to retrieve the content of the matrix into a script-accessible field of colors. New functions [ColorReplace](#), [PixelColorFill](#) and [VectorColorFill](#).

News For Script Engine Version 1.13

[GUI elements available for MAS Script effect](#). It is possible to create some GUI-Elements, so that the user can give some input to the script, like colors for use or other values.

There are several new [functions for strings](#) available. Furthermore, there are different new operations on strings available.

For [case labels](#) variables declared as constants may be used now. Moreover, it is possible to use strings.

Access to the frame id and frame count is now available via Macro for every effect.

News For Script Engine Version 1.11

The effects [SCE Bitmap](#), [SCE Ticker](#), and [SCE Video](#) have the new functions *GetRotation* and *SetRotation*.

There are new global values RED, GREEN, BLUE, AQUA, FUCHSIA, YELLOW, GRAY, SILVER, OLIVE, TEAL, PURPLE, NAVY, MAROON for usage as color parameter for several functions.

For the effect [SCE ColorScroll](#) two new functions *SetFade* and *GetFade* are available.

For the effect [SCE Ticker](#) there are the new functions *SetContinuous* and *GetContinuous*.

For the effect [S2L EQ](#) there are the new functions *GetMonochrome* and *SetMonochrome*.

For the effects [S2L EQ Tubes](#) and [S2L EQ Drops](#) there are the new functions *SetAmplify* and *GetAmplify*. In Addition, the meaning of the *SetSens* and *GetSens* functions have changed.

For the effect [S2L EQ LevelMeter](#) there are the new functions *SetMonochrome* and *GetMonochrome*.

For the effect [M2L ColorScroll](#) there are several new functions and new values for the direction.

There is a new [mix mode](#) `MIXMODE_MASK`.

The function *SetFrameId* has been added in order to control [the frame ID](#).

News For Script Engine Version 1.10

Due to a philosophy change for setting the effects speed from frames per second (FPS) to beats per minute (BPM), for a lot of effects there is an new *SetBpm* function available for the Macro.

The old *SetSpeed* functions are available, but declared as deprecated and may be removed during one of the next releases of MADRIX. For further details, please have a look at the descriptions of the single effects or of the [Macro in general](#). The following effects are affected:

- [SCE Color](#)
- [SCE Bitmap](#)
- [SCE ColorChange](#)
- [SCE Fire](#)
- [SCE Plasma](#)
- [SCE Radial](#)
- [SCE Shapes](#)
- [SCE Ticker](#)
- [SCE Video](#)
- [SCE Wave](#)
- [S2L EQ](#)
- [S2L EQ Drops](#)

- S2L EQ Tubes
- S2L LevelColor
- S2L LevelMeter
- S2L LevelRing
- M2L ColorRings
- M2L ColorScroll
- M2L IntervalDrops
- M2L IntervalTubes

During this change the meaning of the fade slider for the following effects has changed too. While the value for fade was mostly given in frames, now it is given in BPM. The new range of possible values for the function *SetFade* is now from 1 to 3000. This has been changed for the following effects:

- S2L EQ Tubes
- S2L Frequency Flash
- S2L LevelColor
- M2L IntervalTubes
- M2L SingleTones

For the **M2L SingleTones** effect there are new constants available to set the new drawing shapes. Furthermore, there are new functions to set the border, pitch, and the width in percent or in pixel in order to get more scalable effects. A new function *SetColorMode* exists, too. The old function *SetColor* has been removed since its functionality has been removed within MADRIX, too.

There is a new effect called **SCE Shapes** within MADRIX, which also has its own Macro functions.

The **MAS Script effect** has new functions in order to have more control over a running effect, especially when the *Speed Master* is used to increase or decrease speed or to run it backwards. Hence, it is possible to build scripts which can use the frame ID. [It is described here.](#)

There are several new string functions allowing to search for substrings within strings and getting a substring from a string. They are described in the chapter **String operations**.

News For Script Engine Version 1.8

According to new functionality of the effect **SCE ColorRamp**, there are two new values for the *SetDirection* function.

According to new functionality of **SCE ColorScroll**, there are several new values for its *SetDirection* function.

According to new functionality of the **M2L SingleTone**, there are different new values for its *SetDrawMode* function as well as new functions available.

There is new functionality to **read from external files or URLs** using HTTP.

News For Script Engine Version 1.6

Macro Vs. MAS Script Effect

A very powerful new feature in MADRIX is the usage of macros on effects. So the result of each effect can be manipulated by a script written in MADRIX Script. The set of available functions within a macro is almost the same as in the scripting effect except that it has different interface compared to MADRIX. Furthermore, there are special commands available for each effect. They are described in the chapter **Macros for Effects**.

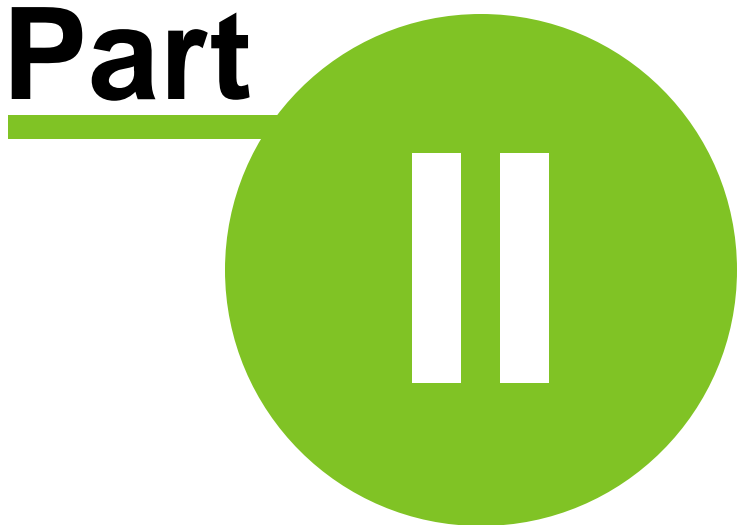
New Functions

GetValue() and *GetAllValues()* have been declared as deprecated functions and should not be used anymore. Please use *GetNoteValue* and *GetAllNoteValues* instead.

For the **ShiftMatrix** function new **shifting directions** are available. Furthermore, a new function called *ShiftVectorMatrix* is now available.

According to the new matrix mapping features in MADRIX, there are **new functions available to setup mirror and tile modes** as well as the *tile*-settings itself.

Part



2 MADRIX Script (Introduction)

2.1 Getting Started

About This Document

MADRIX Script is a scripting language within MADRIX. This document is for all those who want to develop light effects with the help of MADRIX Script. This does not require any programming knowledge.

The components of this language will be explained with the help of different examples. The first step for you is to copy these examples. In the second step, you could try to change and modify the examples. In this way, you will be able to get a feeling for the programming language and the syntax. It is the same like in real life: Skill comes with practice!

For the first steps we advise to use the MAS Script effect. So you will plainly see what a script does. All the examples you can find in this manual are written for the script effect unless stated otherwise.

Using MADRIX Script

MADRIX Script empowers the user to create completely new light effects. Furthermore, it can be used to control and manipulate running effects.

There are four possibilities to use MADRIX Script. The first one is to create a new effect from scratch (**MAS Script Effect**). The second involves modifying the settings of a running effect (**Macros for Effects**). The third possibility controls the main output directly (**Main Output Macro**). Fourth, you can use Storage Place macros to control every single Storage Place individually (**Storage Place Macro**).

MAS Script Effect — Create Your Own Effects

MADRIX offers numerous effects to create a light show. However, there are a lot more things that you are not able to do with the current stock effects. The script effect, called '**MAS Scrip Effect**', provides the possibility to program your own original effects. With MADRIX Script it is possible to put all your creative ideas into practice.



The MAS Script Effect is a normal effect of MADRIX and can be selected from the effect list like all the other effects.

(Chapter: MAS Script Effect)

Macros For Effects — Control Running Effects

Macros are also written in MADRIX Script, but are part of an effect. With macros it is possible to control effects (or layers) and change their results. For example, render parts of an effect transparent or change the color with a grey filter. An example for changing the settings of an effect is to set the text of SCE Ticker to current time.





To run a macro for an effect, please select the **"Macro"** button in the effect area. The **Script Editor** implemented in MADRIX will open. It is then possible to write new macros, an load or edit existing ones. For further information see the chapter **Effect Macros**.

(Chapter: Macros for Effects)

Main Output Macro — Control Your Final Output

Whereas the MAS Script Effect is an individual effect in itself and while macros can be used to manipulate single effects or layers, the Main Output Macro affects the whole output of the two effect pipelines of MADRIX.



The button to call up the Main Output Macro can be found between the **"FADE"** button and the **"Freeze"** button.

(Chapter: Main Output Macro)

Storage Place Macro — Control Individual Storage Place Incl. All Layers

The Storage Place Macro allows you to use a macro that affects your individual Storage Place including all layers.



The button to call up the Storage Place Macro can be found between the **"Description Field"** and the **"Pause"** button.

(Chapter: Storage Place Macro)

2.2 Working With The Script Editor

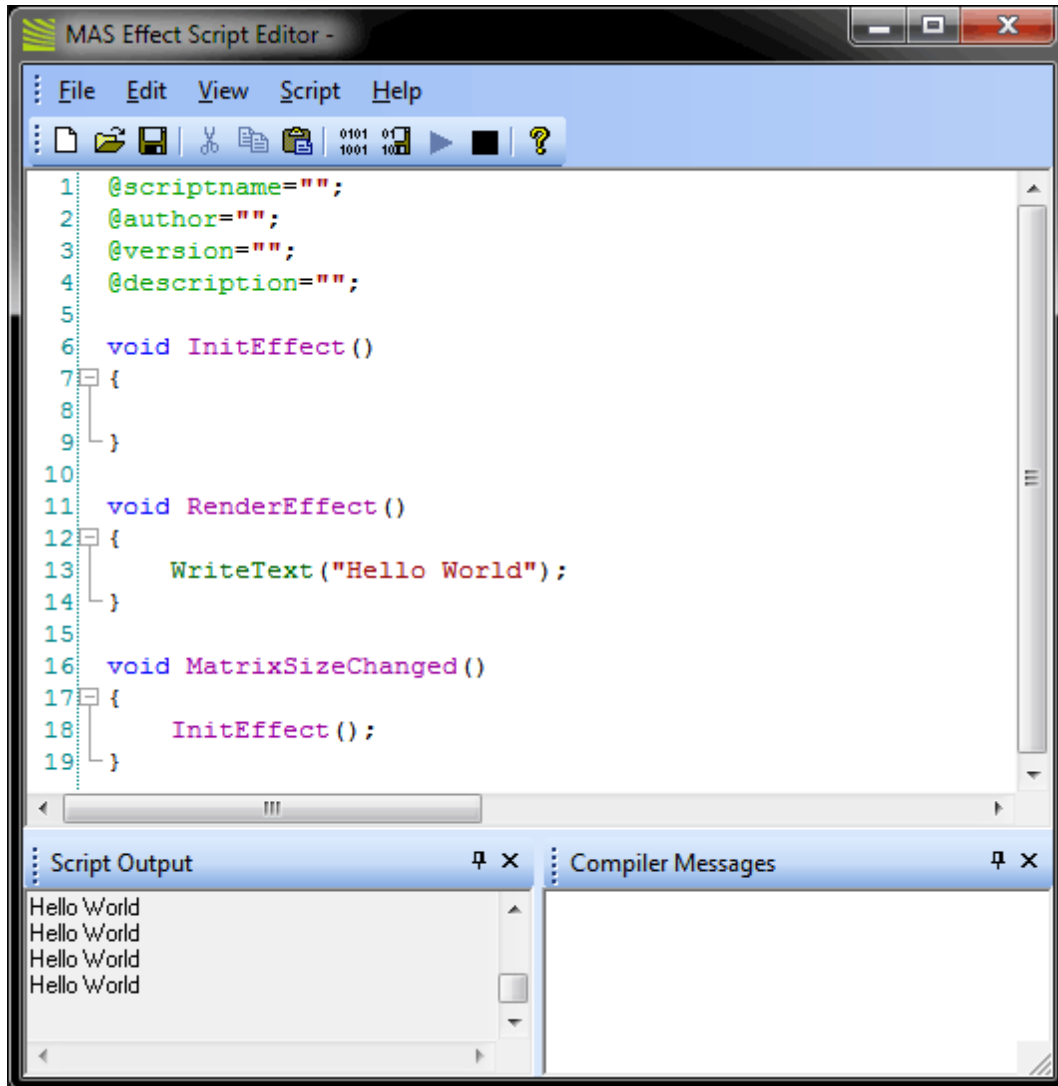
Introduction

The editor that is an integrated part of MADRIX allows to create new MADRIX Scripts and binary scripts. Or you can edit existing ones. Of course, you can load and save a script and start or stop its execution.

The Script Editor Window

In the upper area of the editor, the menu bar for the most common commands is located. In the main window below, you can enter the source code. A script can also write texts to the output window, the Script output. The Compiler Messages box is hidden by default. It can be used to read compiler messages, such as errors and warnings.

The position of each window can be customized by dragging their title bars. Toolbar, Script output, and the Compiler Messages box can be shown or hidden using the menu entry **"View"**.



The Script output starts empty, every time you compile a script. In order to delete the messages manually, please choose in the menu item **"Edit"** the point **"Clear Messages"**. It is also possible to clear the output of the compiler via **"Clear Compiler Messages"**, located in the same menu.

The Script editor supports syntax highlighting which will help you to read the source code, distinguish different types of phrases (e.g. comments, data types) and find errors. Furthermore, the code structuring is enhanced since every line is automatically indented like the last line.

Working with Scripts

Scripts, it does not matter if macros or scripts for the MAS Script effect, are stored together with the effect. Hence, they always form part of a setup or an effect that is saved. Furthermore, it is possible to save scripts to files in order to share them with others. Another way is to save a script as a compiled script. This is a script which does not contain any source code, but only the runtime code. This means that with the help of compiled scripts it is possible to share scripts, but to let the source code remain a secret at the same time.

Creating, Loading, and Saving Scripts



To create a new script please choose in the menu item **"File"** the point **"New"**. Now, the basis for a new script is generated. It consists of some templates for the script's meta data and several default functions. You can edit the new script now.

If you want to save the script, please choose in the **"File"** menu **"Save"** or **"Save as ..."**. The latter opens a dialog for you to enter a new file name for the script and to save it in a new file. The ending of the file will be *.mas*. If the file already exists, you have to confirm or cancel the saving process. When confirming, the file will be overwritten.

When loading a script with the menu item **"File" > "Open"**, MADRIX automatically starts the compilation of the source code, but the script will not start by itself. Alternatively, you can use the "Load" button in the interface of the MAS Script effect, which does not exist for Effect Macros or the Main Output Macro. Then, an (error-free) script will run instantly.

Compiling and Executing Scripts



Before a script can be run, it needs to be compiled. While compiling the script, it is analyzed and translated into the binary format MADRIX works with internally. To compile your script, choose "**Compile**" in the menu "**Compile**" (or press **F5**). After the script was compiled successfully, it will be executed automatically. If the compilation fails, the script cannot be executed. An error message will be displayed. Just double-click on the message and the cursor will jump to the referred position. In addition, the line number is printed with each compiler message.

The execution of a script can be interrupted and continued by using the "**Stop**" and "**Run**" item in the menu "**Compile**". The buttons are also available in the toolbar.

Using Compiled Scripts



A script can also be saved as a compiled script. You can find the function "**Compile and Save**" in the menu "**Compile**" or use the shortcut **F6**. After a successful compilation, a dialog will be opened in which you can enter a name for the compiled script.

Because a compiled script does not contain any source code, only the script's meta data will be displayed in the window when you load such a kind of file. You (or others) will not be able to edit the code.

2.3 Basics

2.3.1 Writing A Script

Introduction

Generally, a script consists of many instructions, which you can enter in any desired editor. The result is called source code or script.

In MADRIX a **Script Editor** is included, too. You can use it to program new scripts or to edit scripts that already exist. Like said before, the following examples mostly refer to the MAS Script effect.



Before a script can be run, it has to be loaded into MADRIX. Please use the **"Load"** button in the dialog of the MAS Script effect. The script will start automatically when loading was successful.

The First Example

A first example of a MADRIX Script can be seen below. You can simply copy the source code and execute it. At the moment, it is not so important to understand all the details. The example repeatedly writes a certain text line in the output window of the editor.

Please open the Script Editor of the MAS Script Effect. Simply copy the whole example into the Editor (and replace the existing code):

```
@scriptname=" ";
@author=" ";
@version=" ";
@description=" ";

void InitEffect()
{

}

void RenderEffect()
{
    WriteText("Hello World");
}

void MatrixSizeChanged()
```

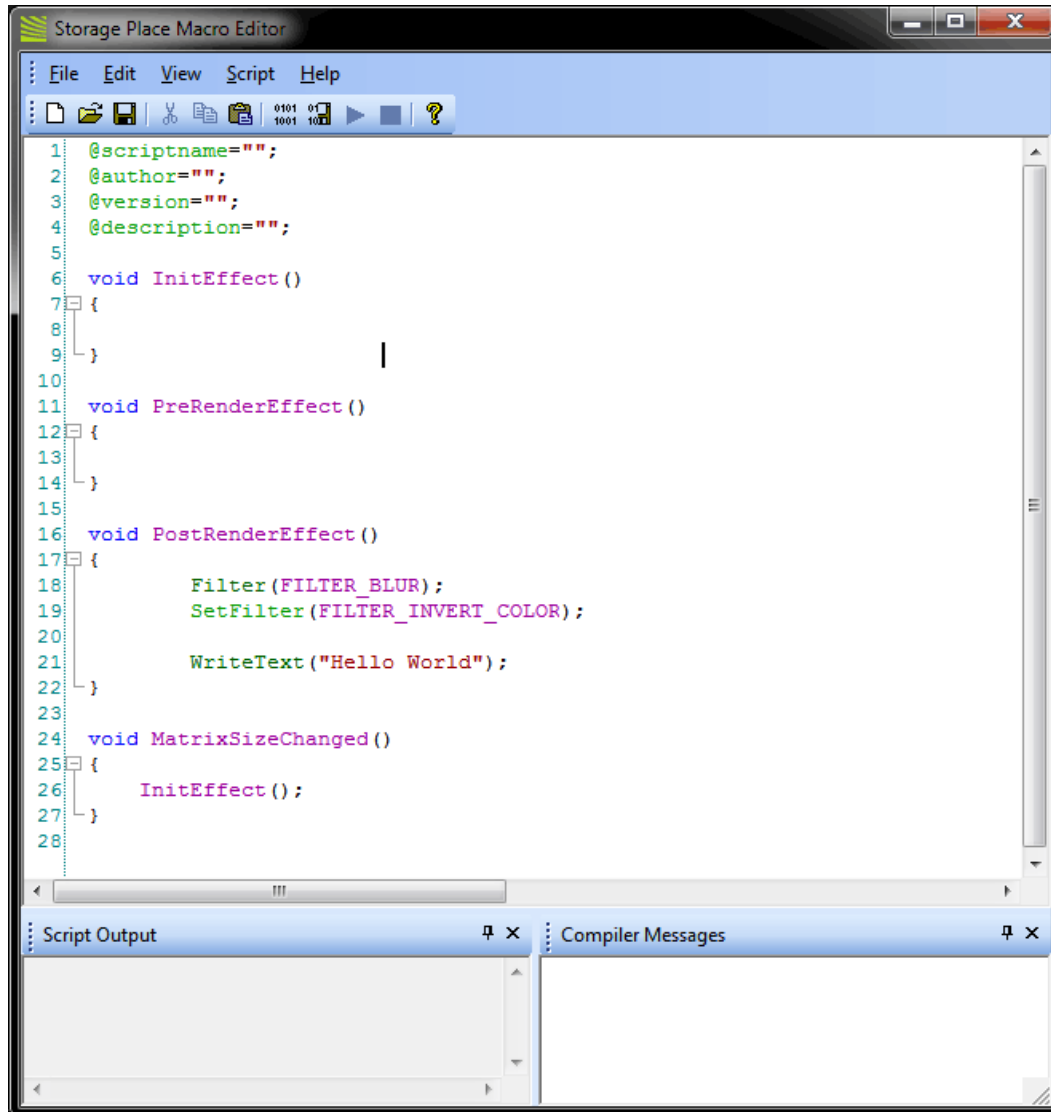
```
{  
    InitEffect();  
}
```

Then, **"Compile"** the script (to be found in menu **"Script"** or use the key **F5**). The function *WriteText(string text)* writes a given character string into the output window of the Script Editor. In this case it is "Hello World". You should see this message in the output area.

You can save the script by choosing **"File > Save as ..."**.

2.3.2 Syntax Highlighting

Overview



As you can see from the screenshot above, MADRIX Script highlights code according to a color scheme (syntax highlighting).

@scriptname, SetFilter Represents functions that can only be used in certain areas of MADRIX Script (for example, Storage Place Macro).

void Represents data types.

<code>FILTER_BLUR</code>	Represents global variables and constants.
<code>Filter, WriteText</code>	Represents functions that can be used in all areas of MADRIX Script (MAS Script Effect, Macros for Effects, Storage Place Macro, Main Output Macro)
<code>"Hello World"</code>	Represents written text for output.

2.3.3 Identifiers

Identifiers are the names of functions or variables. They start with a letter or an underline (_). Other letters, underlines, or numbers can follow afterwards. The exception to this are all characters that do not belong to the English alphabet, e.g. 'ä' or 'é'. There is no restriction for the length of an identifier. Furthermore, there is a distinction between capitalized letters and the use of small letters. For example, *Name* and *name* are two different identifiers.

Examples for valid identifiers: *textFunc*, *_testVar2*, *new*, *NEW*, *New_12340*

Examples for invalid identifiers: *12help*, *1234*, *grösser*, *straße*

2.3.4 Functions

Working With Functions

A script in MADRIX Script consists of a set of functions. Some of them are necessary and called by MADRIX. Others may be used to split the script into smaller parts. Functions form a small parts of a script and hold a number of statements. They can be called from other parts of the script in order to execute their statements. Having statements used outside of functions is not allowed in MADRIX Script.

Creating Functions

Functions consist of a head and a body. The head describes the name of the function, its parameters, and its return value. Whereas, the body includes a block of statements, like this one:

```
void function(int p)
{
    if(p * p > 2)
        do something;
    do something more;
}
```

The first data type, stated in front of the function, describes the kind of value the function returns and it may be of any known data type. In the case above, no value is returned by the function and therefore *void* is declared. The actual name of the function can be any name that follows the rules of identifiers in MADRIX Script as was discussed above. But it has to be unique. It is not allowed to have several functions with the same name or with the name of global variables or constants.

The parameter list following the name of the function may be left empty, but it is necessary to keep the brackets (). Different parameters are separated by comma. A parameter can take on any data type possible. Here are three examples for function declarations:

```
void setPixel(int point[])
{
    do something
}

int[] CreatePoint(int x, int y)
{
```

```
        do something
    }

    string getTag()
    {
        do something
    }
```

Passing Parameters In MADRIX Script

Parameter are always passed via copy by value. (The exception are **fields**.) This means that a parameter may be used as another local variable of a function. Changing the value of a variable does not change the variable the caller has provided. **Please note:** For fields a reference is created. Hence, changing a field results also in changing the field of the caller.

```
void testFunc(int i, int ia[])
{
    i = 5;
    for(int n = 0; n < i; ++n)
    {
        ia[n] = n * n;
    }
}

void RenderEffect()
{
    int testField[];
    int len = 2;
    testFunc(len, testField);
}
```

In *testFunc* the parameter *i* is set to 5 and the field that is passed is filled with several values. After the return of the function in *RenderEffect*, the field is now filled with the values set in *testFunc*. Whereas the variable *len* has not changed and still has a value of 2.

Please note: Passed parameters are always copied to a function, while this is not the case with fields.

Returning A Value

To return a value, the `return` statement must be used followed by an expression. The given expression must result in the same or at least a compatible data type of the declared function's type. It must be the last statement of any function which returns a value unequal to `void`. In addition, `return` can be used to leave a function early. For void functions `return` will be used without an expression. Here are some examples:

```
int[] CreatePoint(int x, int y)
{
    int res[] = {x, y};
    return(res);
}

string getTag()
{
    date d = GetDate();
    switch(d.weekday)
    {
        case 0: return("Sunday"); break;
        case 1: return("Monday"); break;
        case 2: return("Tuesday"); break;
        case 3: return("Wednesday"); break;
        case 4: return("Thursday"); break;
        case 5: return("Friday"); break;
        case 6: return("Saturday"); break;
    } return("unknown day");
}
```

Functions Called By MADRIX

There are several functions called by MADRIX in order to let the script react to different events.

- `void InitEffect()`
- `void RenderEffect()`
- `void PreRenderEffect()`
- `void PostRenderEffect()`
- `void MatrixSizeChanged()`

If a function is not needed by a script, it is not necessary to implement it. Regarding *InitEffect* and *RenderEffect* a message is printed out if one of them is missing. This is not an error, but only an information for the developer of the script. Please note that each component of the MADRIX Script language (MAS Script Effect, Macros, Main Output Macro, Storage Place Macro) may include a different combination of these five functions as this is just an overview.

InitEffect

InitEffect is called by MADRIX whenever the script needs to be initialized. This is the case after compiling and starting a new script or when the user pressed the **"Start"** button of the **Script editor**. A script can assume that any global variable is initialized with 0 and that any global field is empty as long as it has not been initialized with any value.

This function is the right place to initialize global variables, reset any fields, set the speed of an effect, or whatever is necessary to (re)start the script.

RenderEffect

This function is called whenever the effect needs to be rendered. The number of calls per second depends on the currently set speed of the effect. It can be received with the help of the function *GetSpeed* and set with the function *SetSpeed*. This is the right place to calculate the effect and draw it onto the matrix.

PreRenderEffect

This function is called directly before *RenderEffect*. In contrast to the other functions, this one is requested by a script. In order to have this function called, you have to call *DoPreRender*. It may be used if the script has to initialize any settings before an effect is rendered.

```
void InitEffect()
{
    DoPreRender();
}
void PreRenderEffect()
{
    color c = {random(0, 255), random(0, 255)}; Clear(c);
}
```

This example uses the function *PreRenderEffect* to fill the matrix once after initializing a random color for this task.

PostRenderEffect

This function is called directly after *RenderEffect*. After an effect has been rendered completely, certain functions might want to be called. That could be a filter, for example.

MatrixSizeChanged

MatrixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings. Or because new map settings were set, e.g. caused by the call of a map function.

Further Information

There are a lot of functions which can be used to draw objects the matrix, get the data of the sound analysis, or mathematical functions which can be used by a script. The summary contains a [complete list about all available functions](#) within MADRIX Script.

2.3.5 Data Types & Variables

In MADRIX Script variables may be used to store different data. Each variable will be defined with a certain data type. This data type describes the kind of values the variable can store and the operations which are possible with the variable. Here is a small example to get a feeling for variables.

The following source code renders a yellow pixel on a random position each time *RenderEffect* is called. This is the script "SetRandomPixel":

```
void RenderEffect()
{
    color col = {255, 255, 0, 0};
    int px,py;
    px = random(0,GetMatrixWidth()-1);
    py = random(0,GetMatrixHeight()-1);
    SetPixel(col, px, py);

    //a color variable called 'col' is declared and its values are set to yellow (RGB)
    //two variables of type int are declared to store the coordinates of a pixel
    //coordinates for x and y inside the matrix are chosen by chance
    //the pixel is drawn on the matrix
```

```
}
```

2.3.5.1 Using Variables

Introduction

In order to use a variable it must be declared first. This is done by stating the data type of the variable and a name, followed by a semicolon. Furthermore, it is possible to initialize them during the declaration using an equal sign. This means that a particular value can be assigned to the variable already during initialization. Here are some examples:

```
int i;  
float f;  
  
int k = 4;  
string text = „Hello World“;  
int width = GetMatrixWidth();
```

A structure is initialized with a list of expressions separated by comma and written in curly brackets.

```
color white = {255, 255, 255, 255, 0};  
color red = {255};  
date d = {24, 11, 1980};
```

If not all elements of a structure are initialized, the rest will be set to 0.

Constants

It is also possible to declare a variable as a constant. Those variables cannot be changed while the script is running and must be initialized during their declaration. They may be used to give different values names to simplify the reading of the script code. For example, there is a global constant called *PI*. To declare a variable as a constant use the keyword *const*.

```
const int width = 10;  
const int maxPixel = 20 * width;
```

Global And Local Variables

A variable exists within the block in which it has been defined. This may be a function or a block like it is described in statements. It does not exist outside this area. A variable *i*, which was defined in the function *RenderEffect*, does not exist in any other function. Whenever such a block is entered, due to a function call, a loop or something different, the variable is re-initialized. Because of that, a local variable loses its assigned value between two calls of the function.

Global variables are available within the whole script beginning at the position of their definition. They can be used to hold data during the run of a script. Global variables do also exist between two calls of *RenderEffect*. If the script needs to hold data between two runs, global variables are the correct way to do this.

```
int g_iPos = 0;
void RenderEffect()
{
    SetPixel(WHITE, g_iPos, 0);
    g_iPos = (g_iPos + 1) % GetMatrixWidth();
}
```

g_iPos is increased by 1 each time the script is called. It is used to determine the new position of a pixel set to white.

```
void InitEffect()
{
    g_iPos = 0;
}

int g_iPos = 0;

void RenderEffect()
{
    SetPixel(WHITE, g_iPos, 0);
    g_iPos = (g_iPos + 1) % GetMatrixWidth();
}
```

This script will fail since *g_iPos* is unknown in function *InitEffect*. It has been declared after this function. In this way, only *RenderEffect* can use it.

Saving Data

Effects in MADRIX can be stored to a single file or into a whole setup file. Furthermore, it is possible to change a Storage Place to show another effect. If a script effect is reloaded with a compiled and running script, *InitEffect* is called and the script starts from the beginning. Sometimes it is useful that a script does not start from its beginning (let's say a black matrix) but from the same state where it was when it has been stored or the Storage Place has been changed.

The values of global variables do not only remain between two calls of a script, but may be stored when the effect is saved, too. Therefore, the variable should be declared as *persistent*.

```
persistent int g_iPos;
```

Whenever the script is saved, the content of *g_iPos* is also saved. It is loaded again, when the script is loaded. This loading procedure is executed **after** *InitEffect* has been called. Even if the variable is originally initialized in *InitEffect*, it will contain the saved data after *InitEffect* has been called, nevertheless.

More Information

In MADRIX Script several constants are defined by default. They may be used to make the source code more legible. The summary contains an [overview with all available global variables and constants](#).

2.3.5.2 Using Data Types

Primitive Data Types

The example before used two variables with two different data types. Variables offer the possibility to store data. The kind of data depends on the data type. MADRIX Script supports the following primitive data types:

Data Type	Exemplary Values	Description
int	3, -345, 234, 0	32 bit data type. It stores integral numbers between -2 million and +2 million.
float	0.0, -12.45, 3.1415	32 bit data type. It stores floating point numbers.
string	"Hello World", "-3"	Stores character strings of variable length.
bool	true (1), false (0)	Used for implicit use of logical values and comparisons.

Data Type Bool

The data type *bool* is only used internal and cannot be used to declare a variable. This data type only has two possible values, *true* or *false*. It is used for logical operations or for different statements like the *if statement*. For example, the following expression results in a *bool* data type and *false* as its value.

```
3 > 4
int i = 3 > 4 //results in 0
int i = 3 < 4 //results in 1

//usually it is used like this
if(3 > 4)
{
    do something
}
```

True And False

Like said before, a boolean expression results only in true or false.

Furthermore, the keywords *true* and *false* are used within MADRIX Script as function parameters or return values. Those parameters or functions are of the type *int*. In such cases *true* and *false* represent 1 and 0, respectively. They can be used in upper case (*TRUE* / *FALSE*) and lower case (*true* / *false*).

Non-Primitive Data Types - Structures

Complex data types, so-called structures, consist of different elements. The elements of a structure are accessed by their names in the following way: *nameOfVariable.nameOfElement*. For example, *col.r*, if *col* is a variable of data type *color*. The following table is an overview of the structures MADRIX Script provides.

Structure	Elements	Description
color	<ul style="list-style-type: none"> • int r • int g • int b • int w • int a 	<p><i>color</i> stores a color value. There are 5 channels (red, green, blue, white, alpha) with values between 0 and 255.</p> <p>Example: color c = {255, 255, 0, 0}; Members examples: c.r, c.g, c.b, c.w, c.a</p> <p>»Script Example</p>
date	<ul style="list-style-type: none"> • int day • int weekday • int month • int year 	<p><i>date</i> stores a date. Values for days include 1 to 31 for a single day of the month. Values for weekdays include: 0 = Sunday, 1 = Monday, ..., 6 = Saturday. Values for month include 1 to 12 for every single month of the year. Vales for year include year dates.</p> <p>Example: date d = {24, 11, 1980}; Members examples: d.day, d.weekday, d.month, d.year</p> <p>»Script Example</p>
time	<ul style="list-style-type: none"> • int hour • int min • int sec 	<p><i>time</i> stores a time. Valid values are: hours: 0 .. 23, minutes: 0 .. 59, seconds: 0 .. 59.</p> <p>Example: time t = {12, 05, 00}; Members examples: t.hour, t.min, t.sec</p> <p>»Script Example</p>

2.3.5.3 Conversion Between Data Types

Introduction

If there are different data types within an expression, they must be converted into the same type. MADRIX Script does those conversions implicitly, but in most times a warning will be displayed in the compiler message box of the Script Editor. It is also possible to do those conversions explicitly writing the destination data type in brackets before the expression, like this:

```
int i = (int)GetSpeed(); //if the speed was 42.8, i is now 42
string s = (string)i;    //s now consists of the characters "42"
```

GetSpeed returns a *float* value which has to be converted into *int* before it can be assigned to *i*. Be aware that the positions after the decimal point are abridged. Afterwards, the numeric value is assigned to *i*.

The following table shows an overview over possible conversions:

	int	float	string	structure	bool expressions
int	-	Converts the value to float.	Converts the value to string: e.g. 12 = "12".	N/A	Is true if the value is not 0.
float	Converts the value to int. The decimal part is truncated without rounding off.	-	Converts the value to string: e.g. 12.34 = "12.34"	N/A	Is true if value is not 0.
string	If the string is a number, the string is converted into an integer value, otherwise it results in 0.		-	N/A	Is true if string is not empty.
structure	N/A			Conversion between different structures: N/A.	N/A

Implicit Conversions With Math Expressions

If one of the two operands of a math expression is of the type *float* and the other is of the type *int*, the *int* value is converted to float and the expression results in the data type *float*.

If one of the two operands of a math expression is of the type *string*, the other operand is converted into a string and the expression results in a string.

If the conversion is not possible according to the table above, the compiler prints an error and you have to correct the expression. Here are some examples of expressions and their results:

Expression	Result	Explanation
<code>int i = 3 / 4 * 2</code>	0	<code>3 / 4</code> is an <i>integer</i> operation and results in <code>0</code> . Hence, the whole expression results in <code>0</code> .
<code>3 / 4.0 * 2</code>	1.5	Because <code>4.0</code> is a float value, <code>3</code> will be converted into float, too. In this way, <code>3 / 4.0</code> results in <code>0.75</code> . Then, <code>2</code> is also converted into float since the other operand is of the type float. And the result is therefore <code>1.5</code> .
<code>string test = "It is " + 9 + " o'clock."</code>	"It is 9 o'clock."	Since the first operand is a string, <code>9</code> is also converted into a string and concatenated with the first string.
<code>string test = 2 + 3 + "40"</code>	"540"	<code>2 & 3</code> are both integer and will be added up due to an integer operation. The second operation has a string as operand and therefore the other operand will be converted into a string and both are concatenated together.

2.3.5.4 Fields

Basics

Many programming languages provide arrays, vectors, lists or any other data type to store dynamic data. Dynamic data is not known yet when the program is written. For such tasks MADRIX Script provides dynamic fields. They are declared like variables, followed by `[]`.

```
int    aiIntField[];      //a 1-dimensional field of integer values
date   adDateField[];    //a 1-dimensional field of dates
float  aafFloatField[][]; //a 2-dimensional field of float values
```

It is also possible to initialize fields using a list of values. These are described by values separated with commas and written in curly brackets.

```
//initialize a field with 5 integer values
int aiIntField[] = {2, 3, 4, 5, 6};
```

The operator [expression] provides access to the elements of a field. The expression must result in an *integer* or compatible value. The lowest index of a field is 0. This means the first entry of a field is always indexed with 0; a field does not start with 1, but 0. When an element is accessed, the field grows automatically in order to provide the requested element. It is not necessary nor possible to request the size of a field explicitly. Here is an example to access a field with *integer* values.

```
int aiIntField[];  
aiIntField[0] = 10;  
aiIntField[1] = 20;  
aiIntField[2] = aiIntField[3];
```

After the last access the field will have a length of 4 because 3 was the last accessed element. The initial value of an element is "0" or an empty string or false. The length-attribute of a field tells the current size of a field, which is the number of currently provided elements.

The Length Or Size Of Fields

Each field has a *length*-attribute. It can be accessed through the "."-operator which is also used to access elements from a structure.

```
int l = aiIntField.length; //store length of the field in l
```

Please note: The length of a field is defined by the highest index that was used to request an element.

Multidimensional Fields

Up to this point, one-dimensional fields were introduced. But multidimensional fields are also possible. To declare a multidimensional field, a "[]" must be added to its declaration for each dimension. Up to now, the only limit to the number of possible dimensions is set by the resources of the computer on which the script is running. It is also possible to initialize fields using a list of elements for each dimension. Here are some examples:

```
int aaField[][];      //a 2-dimensional field  
int aaaField3[][][]; //a 3-dimensional field  
  
//initializes the field with two dimensions and three values each
```

```
int aaField2[][] = {{2, 3, 4}, {6, 7, 8}};

//a 2-dimensional field of color elements
color aaCField[][] = {
    { {0, 0, 0, 0, 255}, {255, 255, 255}, {255, 255} },
    { {255, 255, 255}, {255, 255, 255} },
    { {0, 255, 255, 255} } };

```

The operator *[expression]* accesses a single element of a field, which for a multi dimensional field may be another field. In order to access a single element, the applicable index must be used. For example, the 5th element must be accessed with the index 4, while the first entry has the index 0. The same is true for the attribute *length*. It returns the length of the currently accessed field. Here are two examples:

```
int aaField[][];
int aField1[] = {1, 2, 3, 4, 5, 6}; //intialize the field

aaField[0] = aField1; //assign aField1 to the first element of aaField
aaField[0][aField1.length] = aField1.length + 1;
aaField[1][0] = 1;
aaField[1][2] = 2;

```

Explanation: At the end of this example *aaField* consists of two fields of *int* values. The first one has a length of 7 and the second one a length of 3 (due to the access of the element with the index 2). These lengths can be received by reading the *length* attributes.

```
WriteText("Number of fields in aaField: " + aaField.length);
WriteText("Number of elements in aaField[0]: " + aaField[0].length);
WriteText("Number of elements in aaField[1]: " + aaField[1].length);

```

Memory Management Of Fields

Although the memory for a field is dynamic you have to think of it beforehand. Think about the following example:

```
int aiField[][];
aiField[10000][10000] = 1;

```

After the assignment, the field has indeed a size of 10.000 x 10.000 elements of *int*-values. An *int* value needs four bytes and $10.000 * 10.000 * 4 = 4.0000.0000$ bytes, add up to around 382 megabytes (MB) of memory. So please pay attention when using very big fields.

Full Example

This example just plays with the fields and its content. It is just to show how to work with fields and to get a feeling for them. A better example is given in the chapter [Loops](#).

```
void InitEffect()
{
    SetSpeed(1);
}

void RenderEffect()
{
    int aaField[][];
    int aField1[] = {1, 2, 3, 4, 5, 6}; //intialize the field

    aaField[0] = aField1; //assign aField1 to first element of aaField
    aaField[0][aField1.length] = aField1.length + 1;
    aaField[1][0] = 1;
    aaField[1][2] = 2;

    WriteText("Number of fields in aaField: " + aaField.length);
    WriteText("Number of elements in aaField[0]: " + aaField[0].length);
    WriteText("Number of elements in aaField[1]: " + aaField[1].length);

    WriteText("Element of aaField[0][0]: " + aaField[0][0]);
    WriteText("Element of aaField[0][1]: " + aaField[0][1]);
    WriteText("Element of aaField[0][2]: " + aaField[0][2]);
    WriteText("Element of aaField[0][3]: " + aaField[0][3]);
    WriteText("Element of aaField[0][4]: " + aaField[0][4]);
    WriteText("Element of aaField[0][5]: " + aaField[0][5]);

    WriteText("Element of aaField[1][0]: " + aaField[1][0]);
    WriteText("Element of aaField[1][1]: " + aaField[1][1]);
    WriteText("Element of aaField[1][2]: " + aaField[1][2]);
}
```

2.3.5.5 Strings & String Operations

MADRIX Script provides several functions to manipulate strings, find substrings, and to perform many more operations.

Operations On Strings

Assigning Data Types

It is possible to assign *integer*, *float*, and *strings* to another *string* like shown in the following example:

```
string s;  
string t;  
s = t;  
s = "Hello world";  
s = 5;  
s = 3.5;
```

Furthermore, it is possible to assign single characters of a *string* to a character of another *string* like shown below:

```
string s, t;  
s = "New";  
t = "new";  
  
s = t;  
s[0] = t[0];
```

Furthermore, it is possible to assign a double quoted *string* to a character of a *string*. But the assigned *string* must have exactly one character. Here is an example:

```
string s;  
s[0] = "T";  
s[1] = "1";  
s[2] = ".";
```

The following lines are invalid and will result in a compiler error since the given *strings* have more, or less than one character:

```
s[0] = "New"; //given string has three character but not one  
s[0] = "";    //given string is empty
```

Comparing Two Strings

As it is possible to compare two numbers using the compare operators, it is also possible to compare two *strings*. The following table provides an overview of the possible operations.

Operator	Description
<code>str1 == str2</code>	Checks for the equality of two strings.
<code>str1 != str2</code>	Checks for the inequality of two strings.
<code>str1 < str2</code>	Checks if the first string is less than the second string.
<code>str1 <= str2</code>	Checks if the first string is less or equal to the second string.
<code>str1 > str2</code>	Checks if the first string is greater than the other string.
<code>str1 >= str2</code>	Checks if the first string is greater or equal to the other string.

Please note: It is not possible to compare fields of *strings*.

Like in the case of assignments, it is also possible to compare single characters of a *string* against a double-quoted *string* with exactly one character:

```
if(s[0] == "A") ...  
else if(s[0] == "!") ...  
...
```

It is also possible to compare single characters of a *string* against *integer* numbers:

```
if(s[0] == 1) ...  
else if(s[0] == 2) ...
```

This also works for the switch/case statements. But the "1" as a label of a case means that the same as the 1. So the following two case labels mean the same and this would result in a compiler error:

```
string s = "1";  
switch(s[0])  
{  
    case "1": do something; break;  
    case 1:   do something else; break;  
    case "A": do something; break;  
    //is also valid to check for letters and other characters  
    ...  
}
```

Using Strings Within Switch/Case Statements

Another possibility is to use double-quoted *strings* of one character in case of labels. The following theoretical example demonstrates this:

```
string s = "New";
for(int i = 0; i < s.length; ++i)
{
    switch(s[i])
    {
        case "A": do something; break;
        case "B": do something else; break;
        case "!": do something; break;
    }
}
```

Functions For Strings

Function	Description
int findstring (int startIndex, string text, string substring)	This functions looks for the <i>substring</i> in the given text. The search starts at the given <i>startIndex</i> . The first character has an index of 0. The function starts its search at a specified position of the entire <i>text</i> using <i>startIndex</i> and returns an index that describes the position at which the <i>substring</i> begins. If the substring is not found, -1 is returned.
string substring (string text, int startIndex, int count)	The function extracts <i>count</i> characters from the given <i>text</i> starting with <i>startIndex</i> . If <i>count</i> is -1, all characters of the string starting at <i>startIndex</i> are returned. c
int rfindstring (int startIndex, string text, string substring)	This functions looks for the <i>substring</i> in the given text from its end to the beginning. The function starts its search at a specified position of the entire <i>text</i> using <i>startIndex</i> and returns an index that describes the position at which the <i>substring</i> begins. If the substring was not found, -1 is returned.
int startswith (string text, string substring)	This function checks if the <i>text</i> string starts with the given <i>substring</i> . If <i>text</i> starts with <i>substring</i> , <i>true</i> is returned, otherwise <i>false</i> .
int endswith (string text, string substring)	This function checks if the <i>text</i> string ends with the given <i>substring</i> . If <i>text</i> ends with <i>substring</i> , <i>true</i> is returned, otherwise <i>false</i> .
int isalnum (string text)	Returns <i>true</i> if the given string contains only characters and figures and its length is greater then 0, otherwise <i>false</i> is returned.
int isalpha (string text)	Returns <i>true</i> if the given string contains only characters and its length is greater than 0, otherwise <i>false</i> is returned.

int isnum (string text)	Returns <i>true</i> if the given text represents a number. This may be an integer number or a <i>floating</i> point number (e.g. 1.3). Otherwise it returns <i>false</i> .
void tolower (string text)	Converts each character of the given <i>string</i> into a lower-case character.
void toupper (string text)	Converts each character of the given <i>string</i> into an upper-case character.
void strip (string text)	Removes leading and ending white spaces like space, tabulator, line feeds and so on from the given <i>string</i> .
int strcmp (string str1, string str2)	Compares two given strings with each other. If they are equal, 0 is returned. -1 is returned if <i>str1</i> is less than <i>str2</i> . A value of 1 is returned if <i>str1</i> is bigger than <i>str2</i> .
void replace (string src, string old, string new)	Replaces any appearances of <i>old</i> within <i>src</i> with <i>new</i> .
void tokenize (string src, string delimiter, string reslist [])	Separates the string <i>src</i> into smaller pieces delimited by characters within <i>delemiter</i> . The result is returned in <i>reslist</i> . See below for further details.

Tokenizing Strings

The function [tokenize](#) enables you to tokenize a string. The single tokens will be delimited by the characters within the second parameter. Each character identifies a single delimiter. The following examples show the usage of the function and the results.

```
string s = "Have a wonderful,nice day".
string res[];
tokenize(s, " ,", res);
```

```
string s = "one two,three";
string res[];
tokenize(s, ",", res);
```

Explanation:

The variable *res* of the first example will be filled with the following five values: {Have; a; wonderful; nice; day}

The *res*-variable of the second example will be filled with the following two values: {one two; three}

The result of the second example will contain only two entries. "one two" is only one entry since the tokens of the second example are only delimited by coma but not by space.

Splitting Strings With White Spaces

There is a constant called `WHITE_SPACES` which can be used as *delimiter* in order to split a text by any white spaces like tabulator, new line, or space.

```
string s = "Have a wonderful, nice day".
string res[];
tokenize(s, WHITE_SPACES, res);

//or another example which also uses the comma as delimiter
tokenize(s, WHITE_SPACES + ",", res);
```

Examples

Substring

This example extracts a part from a *string*. Insert the source code into the function *RenderEffect*. As the result "World" should be printed in the output window of the Script Editor.

```
string txt = "Hello World";
string subText = substring(txt, 6, 5); //retrieves "World" from txt
WriteText(subText);
```

(Description)

2.3.6 Expressions

Introduction

Expressions are used to calculate values. Additionally, they can be assigned to variables or given as parameters to functions. If an expression is followed by a semicolon, it is called an expression statement. There are different operators available to write expressions.

Operands For An Expression

An expression is formed of operands and operators. The number of operands depends on the operator. There are unary operators, which use only one operand such as "-". And there are binary operators, which need two operands like "+".

Operands of an expression can be function calls, variables, constant values like "5" or another expression within brackets like $(3 + 5)$.

Assignment Operator

To assign any value to a variable the assign-operator "=" is used in the following way:

```
variable = expression;
```

The *expression* after the "=" may be any complex expression. But it has to be compatible with the given variable after [the conversion rules](#) of MADRIX Script.

```
float f = cos(0.5);  
float f2 = f;
```

Math Operations

'++'/'--' Operator

The ++ and -- operators are unary operators and only defined for usage with variables of the type *int*. They are used like this:

```
i++;  
i--;
```

They increment/decrement the variable by one. This operator is known from C/C++. This programming language offers two possible ways of using them, as suffix and as prefix operators (i+ and ++i). In MADRIX Script they are currently only available as suffix operators (i++) and they work in that way. Such a suffix operator results in the current value of the given variable and afterwards increments/decrements it.

```
int i = 4;  
i = i++ * 2;
```

In the end, *i* is 8. In the first line, 4 is assigned to the integer value *i*. In the second line, the expression *i++* results in 4. The multiplication therefore is "4 * 2" which results in 8. This value is assigned to *i* and therefore, *i* = 8.

'-' Operator

The "-" operator is also available as unary operator and negates the value of the given operand. It supports *int*- and *float*-values.

```
-4;  
-i;  
-(3 * 5);
```

The "+" operator may also be used as unary operator, but it does not make sense because it does not change the result of an expression.

Binary Operations

The operators +, -, * and / support *int* and *float* values and cause a usual arithmetical addition, subtraction, multiplication, and division of the two operands.

```
i = 4 + 4 * 5;  
i = (4 + 4) * 5;  
i = 4 / 5;
```

Operator precedence rules are considered. **Note** that operations with *integer* values result in *integer* values and are done as *integer* operations. So $3 / 4$ does not result in 0.75 as may be expected, but in 0. To get a result of 0.75, at least one of the operators must be a *float* value. An example would be $3.0 / 4$, where the 4 is also converted into a *float*-value.

Concatenating Strings

The "+" operator can also be used to concatenate strings together.

```
string s = "Hallo " + "Welt";
```

If one of the two operands is of the type *string*, the other one is converted and the two strings are concatenated together.

```
int i = 4;  
string s = 4 + "th run";
```

This example results in "4th run". **Please note** that the following example may be misinterpreted since the first part of the expression is of the type *integer* and will result in an *integer*. It is then converted and concatenated into one string.

```
string s = 3 + 4 + "th run";
```

The resulting string is "7th run" and not, as perhaps sometimes expected "34th run".

Modulo Operation

The modulo operation calculates the *integer* remainder of an *integer* division. The operator in MADRIX Script for modulo operations is %.

```
10 % 2;
```

The operator % is only defined for *int* values.

Additional Assignment Operations

To have less code and increase readability, there are additional assign operators: +=, -=, *=, /=, %=.

```
i += 3 + 4
```


is the same as

```
i = i + (3 + 4)
```

and so on. The resulting source code is much more easy to read.

Operations Of Comparison

With operations of comparison you can test two expressions for a certain relation. Possible comparisons (and operators) are "less than" (<), "less equal" (<=), "more than" (>), "more equal" (>=), "equal" (==) and "not equal" (!=). Make sure that you can distinguish the meaning of a single equal sign (assignment operator) and a double equal sign (compare operator). Operators of comparison always return a *bool* value, the comparison is either *true* or *false*.

```
i > 4  
3 < j
```

Logical Operations

The '!' Operator

The !-operator is an unary operator, which logical negates the value of the given expression. *false* becomes *true* and vice versa.

```
!3  
!(3 > 4)  
!"Hallo Welt"
```

'And'/'Or' Operator

The and and or-operators are logical operators. They need operands of the data type *bool* and always result in a *bool*-value. In MADRIX Script the and-operator is expressed via "&&" and the or-operator is declared with "||".

Unlike C/C++, in MADRIX Script both operands are always evaluated. So, even if the first operand of an "&&" - operator results in *false*, this means that the whole expression will be *false*. But the second operand will be calculated, too. Here some examples for using those operators:

```
int i = i || j  
int i = (3 < 4) || (4 > 3)
```

Those operations are usually used within statements, which require to make a decision like the **if-statement** as described later on.

Using Operands Of Different Data Types

Operands are automatically converted, e.g. from *bool* to *int*, when making an assignment (see also **Conversion Between Data Types**).

```
int i = GetMatrixWidth() > GetMatrixHeight();
int j = (2 * sin(PI) * i) + (2 * cos(PI) * !i);
```

In the first line, the resulting *bool*-value of the ">"-operator is implicitly converted into *int* and results in 0 or 1. In the second line, the 2 in both expression parts is converted to *float* since the *sin* and the *cos* functions result in *float*. The same holds true for the operands *i* and *!i*.

Full Example

The following example uses different expressions to calculate the coordinates on which the next pixel is to be set. It also calculates the color of the next pixel.

```
int g_point[];
persistent color g_color;
void InitEffect()
{
    g_point[0] = 0;
    g_point[1] = 0;
    color c = {random(0, 255), random(0, 255), random(0, 255), random(0, 255)};
    g_color = c;
    SetSpeed(50);
}

void RenderEffect()
{
    //calculate the color for the next pixel
    g_color.r += (int)(255.0 * (0.5 + 0.5 * sin(g_point[0] * g_point[0])));
    g_color.g += (int)(255.0 * (0.5 + 0.5 * cos(g_point[0] * g_point[1])));
    g_color.b += (int)(255.0 * (0.5 + 0.5 * sin(g_point[0] - g_point[1])));
    g_color.w += (int)(255.0 * (0.5 + 0.5 * sin(g_point[0] + g_point[1])));

    //make sure, colors are only between 0 and 255
    g_color.r %= 256;
    g_color.g %= 256;
    g_color.b %= 256;
}
```

```
g_color.w %= 256;

setPixel(g_point);
//setup the next point, x++
//if x > MatrixWidth x = 0 and y++
//if y > MatrixHeight y = 0
g_point[0] = (g_point[0] + 1) % GetMatrixWidth();
g_point[1] = ((int)(g_point[0] == 0) * 1 +
              g_point[1]) % GetMatrixHeight();
}

void setPixel(int pt[])
{
    SetPixel(g_color, pt[0], pt[1]);
}
```

2.3.7 Statements

Introduction

There are different statements available in a script, for example the if-statement or the call statement to call functions. Those will be described later on in this chapter. Another statement is the expression statement. A statement is an expression statement if it is followed by a semicolon. For example:

```
i = i + 1;
InitEffect();
```

Using Blocks

A block is started and finished (opened and closed) with the help of curly brackets. Blocks may be used to group different statements together. There are also different statements which require a block if more than one statement should be executed. For example the *while* or the *if* statement are good examples.

2.3.7.1 'If' & 'Else If' Statements

'If' Construct

Very often it is necessary to make decisions within a script. You could for example want to use red as your background color every day if it is after 9 a.m. Or for example, it could be your wish to clear the matrix and change the color if the matrix has been filled up completely. Therefore, in MADRIX Script the keywords *if* and *else* exist. They may be used like this:

```
if(condition)
    statement
else
    statement
```

The first statement is executed if the given condition is *true* or unequal to 0. Otherwise the second statement, stated after *else*, is executed.

Statement may include a single statement or a block of statements and the *else*-part is optional. Here are some examples for the *if*-statement:

```
if(x % 2 == 0)
{
    col.r = 0;
    SetPixel(col, x, y);
}
else
{
    col.r = 255;
    SetPixel(col, x, y);
}
```

```
if(testPixel(x, y) != 0)
{
    SetPixel(WHITE, x, y);
}
```

```
if(x + 2 > y)
    y++;
```

It is important to consider that an *else* always refers to the last *if*-statement. However, blocks may be used to make the intention clear. To give you a demonstration, please consider the following example. It may be interpreted wrong since *i* will be incremented if the *j > i*-condition fails and not if the *i > 3*-condition fails like it is implied by the given *else*.

```
if(i > 3)
    if(j > i)
        j = i;
else
    i++;
```

To let the compiler create the correct code, use blocks:

```
if(i > 3)
{
    if(j > i)
        j = i;
}
else
    i++;
```

'Else If' Construct

Else if is an additional structure to implement decisions. It may be used like this:

```
if(condition)
    statement
else if (condition)
    statement
```

Like described above, *else* is always followed by a statement. And *if* is such a statement. Then, it is logical that an *else* may be directly followed by an *if*. The *else if* structure is very useful to make code with lot a lot of decisions more readable. It enables you to check for different conditions, which shall only be checked if the previous condition was passed successfully. Here is an example which selects another color for different days. Simply copy it and paste it in the function *RenderEffect*.

```
date t = GetDate();
color c;
if(t.day < 31)
    c.r = 255;
else if(t.day <= 21)
    c.g = 255;
else if(t.day <= 11)
    c.b = 255;
else
    c = WHITE;
Clear(c);
```

Full Example

The following example renders a blinking cross onto the matrix. Instead of using random colors, predefined colors will be used. During each call of *RenderEffect*, the color will be chosen.

```
color colCross = {0, 255, 255, 255};
int g_iCol;

void InitEffect()
{
    g_iCol = 0;
    SetSpeed(1.0);
}

void RenderEffect()
{
    int x, y;
    if(g_iCol == 0)
    {
        colCross.r = 255;
        g_iCol = 1;
    }
    else if(g_iCol == 1)
    {
        colCross.r = 155;
        g_iCol = 2;
    }
    else
    {
        colCross.r = 0;
        g_iCol = 0;
    }

    for(int x = 0; x < GetMatrixWidth(); x++)
        for(int y = 0; y < GetMatrixHeight(); y++)
        {
            if(x == y)
                SetPixel(colCross, x, y);
            else if(GetMatrixWidth() - x-1 == y)
                SetPixel(colCross, x, y);
            else
                SetPixel(BLACK, x, y);
        }
}
```

2.3.7.2 'Switch' Statements

Introduction

If it is required to compare an integer variable with a lot of different values, using of the *if*-statement may be very exhausting. In this case the *switch*-statement may help. It has the following syntax:

```
switch(expression)
{
    case label1:
        list of statements
    case label2:
        list of statements
    default:
        list of statements
}
```

The expression must result in *int* or a compatible data type which can be converted implicitly. The *default*-label is optional. The label must result in a constant value and since Version 1.13 of MADRIX Script, it is possible to use *integer* values like "0" or "12", constant variables, or double-quoted strings like "A". After the colon one or more statements may follow, but blocks are also allowed. Moreover, each label has to be unique.

Labels are not an independent block of source code, but a marker where the code execution should be continued if the *expression* has the corresponding value. So after execution of the statements for *label 1* the statements of *label 2* will be executed and so on. In order to avoid that behavior, use the keyword **break**. If the expression does not match any of the given labels, the execution will be continued with the *default*-label.

The following sample code writes the name of the current day into the message window of the editor.

```
date d = GetDate();
switch(d.weekday)
{
    case 0: WriteText("Sunday"); break;
    case 1: WriteText("Monday"); break;
    case 2: WriteText("Tuesday"); break;
    case 3: WriteText("Wednesday"); break;
    case 4: WriteText("Thursday"); break;
    case 5: WriteText("Friday"); break;
    case 6: WriteText("Saturday"); break;
}
```

Now, we are also able to shorten the usage of the *if*-statement in the following way:

```
color colCross = {0, 255, 255, 255};
int g_iCol;

void InitEffect()
{
    g_iCol = 0;
    SetSpeed(1.0);
}

void RenderEffect()
{
    int x, y;
    switch(g_iCol)
    {
        case 0:
            colCross.r = 255;
            g_iCol = 1;
            break;
        case 1:
            colCross.r = 155;
            g_iCol = 2;
            break;
        default:
            colCross.r = 0;
            g_iCol = 0;
            break;
    } //switch[current color]
    for(int x = 0; x < GetMatrixWidth(); x++)
    {
        for(int y = 0; y < GetMatrixHeight(); y++)
        {
            if(x == y)
                SetPixel(colCross, x, y);
            else if(GetMatrixWidth() - x-1 == y)
                SetPixel(colCross, x, y);
            else
                SetPixel(BLACK, x, y);
        } //for[each line]
    }
}
```

Using Constant Variables

As was said before, it is possible to use variables declared as constants, but it is necessary that the compiler is able to compute the value during compilation time. Here is an example with a valid as well as an invalid case label.

```
const int label1 = 1;
const int label2 = GetMatrixWidth();
```



```
switch(<something>) {  
    case label1:    do something  
        break;  
    case label2: do something else  
        break;  
}
```

The first label (label1) is a valid case label since the compiler is able to compute the value of 1 during compilation time. The second label (label2) is invalid since it is computed during runtime and therefore it is not a constant value for the compiler, even though it is not possible to change its value later on.

The following examples are all valid case labels since the compiler can compute their values:

```
const int label1 = 2 + 4;  
const int label2 = label1 + 3;  
const int label3 = label2;
```

Using Double-Quoted Strings

It is also possible to use double-quoted strings for case labels. But they have to have the length of 1. Here is an example:

```
void writeText(string s)  
{  
    for(int i = 0; i < s.length; i++)  
    {  
        switch(s[i])  
        {  
            case "A": do something; break;  
            case "B": do something with b; break;  
            ...  
        }  
    }  
}
```

2.3.7.3 'For' & 'While' Loops

Introduction

Loops are used in programming languages to repeat tasks. For example, every pixel should be set to a green color. Loops run as often as a given condition is true. MADRIX Script offers two possible forms of loops, the *for* loop and the *while* loop. Both are similar to the loops in the programming language C.

'While'-Loop

The *while*-loop is built in this way:

```
while(condition)
    statement
```

The *statement* may be a single expression statement. If more than one statement needs to be executed, a block is needed, as shown in the following example:

```
int x = 0;
while(x < 10)
{
    SetPixel(WHITE, x, 0);
    i++;
}
```

'For'-Loop

The *for*-loop is built like this:

```
for(initialization; condition; expression)
    statement
```

This is the same as:

```
initialization;
while(condition)
{
    statement;
    expression;
}
```

The *initialization*-part may contain the declaration of a new variable or an expression. It is executed the first time the loop runs. There may be different initializations separated by comma. Newly declared variables only exist within the *for*-loop. After *initialization*, the *condition* is checked. If it is unequal to 0, the given statement is executed. (In this respect, a value of 0 represents *false*. But a while loop will only be executed, if the condition is true. And true is represented by a value of 1, which in turn is unequal to 0.)

The *condition*-part contains an expression. As long as the given expression is not 0 or *false*, the given statement is executed. If the *condition*-part is empty as discussed beneath, it will be interpreted as *true*.

The *expression* in the *expression*-part is executed each time before running the *statement* and before checking the *condition*. But not the first time. There may be different expressions separated by comma.

It is possible to leave different parts of the *for*-loop empty. But semicolons are necessary, nevertheless. This may be used to implement the initialization outside the loop. Here are three examples for *for*-loops. The first one is an endless loop:

```
for( ; ; )
{
    do anything;
}

for(int x = 0; x < GetMatrixWidth(); x++)
    for(int y = 0; y < GetMatrixHeight(); y++)
        SetPixel(WHITE, x, y);

for(int x = 0; x < GetMatrixWidth(); x++)
{
    if(x % 2)
        DrawPixelLine(WHITE, x, 0, x, GetMatrixHeight());
    else
        DrawPixelLine(BLACK, x, 0, x, GetMatrixHeight());
}
```

Controlling Loops: 'Break' And 'Continue'

There are two possibilities to control a loop. First, it is possible to interrupt the execution of a loop (break). Furthermore, there is a way to skip the rest of the statements of the loop and to go to its beginning (continue).

'Break'

With the keyword *break* a loop can be quit immediately. For example:

```
int x = 0;
while(x < 10)
{
    if(i >= GetMatrixWidth())
        break; //leave loop now!

    SetPixel(WHITE, x, 0);
    x++;
}
```

The execution of the script is continued after the loop. No other statement within the loop is executed after *break*.

'Continue'

With the keyword *continue* it is possible to skip the rest of the statements within a loop and to start anew. For example:

```
int x = 0;
while(x < 10)
{
    if(x % 2 == 0)
        continue;
    SetPixel(WHITE, x++, 0);
}
```

Examples

'While' Loop

The following example "cos" draws parts of a cosinus curve onto the matrix while changing the background color.

```
void InitEffect()
{
```

```

    SetSpeed(1);
}

void RenderEffect()
{
    color col = {200, 200, 100, 200};
    color colBK={random(0,150), random(0,255), random(0,100), random(0,255)};
    //set background
    Clear(colBK);
    //draw cosine curve
    int px = 0;
    int py = 0;
    float y;
    float t = 0;
    while(px < GetMatrixWidth())
    {
        y = cos(t) * GetMatrixHeight();
        py = (int)y;
        t = t + (PI * 3 / GetMatrixHeight());
        px++;
        SetPixel(col, px, py);
    }//while[x < GetMatrixWidth()]
}

```

'For' Loop

Here is another full example which uses fields to store random colors and to fill the matrix with them.

```

persistent color g_MatrixColors[][];
void RenderEffect()
{
    //select random color
    color col = {random(0,255), random(0,255), random(0,255), random(0,255)};

    //select random pixel coordinates
    int px = random(0,GetMatrixWidth()-1);
    int py = random(0,GetMatrixHeight()-1);

    //save the selected color
    g_MatrixColors[px][py] = col;

    //draw points of the field
    for(px = 0; px < GetMatrixWidth(); px++)
    {
        for(py = 0; py < GetMatrixHeight(); py++)
        {
            SetPixel(g_MatrixColors[px][py], px, py);
        }//for[each line]
    }//for[each column]
}

void InitEffect()
{

```

```
for(int x = 0; x < GetMatrixWidth(); x++)
    for(int y = 0; y < GetMatrixHeight(); y++)
    {
        g_MatrixColors[x][y].r = 0;
        g_MatrixColors[x][y].g = 0;
        g_MatrixColors[x][y].b = 0;
        g_MatrixColors[x][y].w = 0;
    }
    SetSpeed(5);
}
```

2.3.8 Reading From External Files

Asynchronous File Reading

The function

```
int ReadAsync(string file, string txt)
```

reads content as text from a file into the string *txt*. The file is opened and closed automatically. There is no "open" function like in other programming languages. The parameter *file* may contain a filename of a local file, like "C:\config.txt". In addition, the HTTP protocol is supported. That means it is possible to retrieve data from a web server. For example: "http://www.madrix.com". The following examples would read some content from different files.

```
string txt;
ReadAsync("C:\config.txt", txt);
ReadAsync("http://www.testserver.de/testfile.txt");
```

Here is an example which reads some numbers from a file and renders a curve on the matrix.

```
float g_pos[] = {0.8, 0.5};
string file = "C:\temp\src.txt";
string txt;

void InitEffect()
{
    SetSpeed(2.0);
}

void RenderEffect()
{
    ShiftVectorMatrix(0.0, 0.0, 1.0, 1.0, SHIFT_LEFT, 0.1);
    ReadAsync(file, txt);
    float f = (float)txt;

    DrawVectorLine(WHITE, g_pos[0], g_pos[1], g_pos[0] + 0.1, f);
    g_pos[1] = f;
}
```

```
}
```

Let's take for example a program which writes the temperature from an external sensor connected to your USB interface to the file and MADRIX draws the curve onto a matrix. As described later on, it would also be possible to write a macro for the SCE Ticker effect to display the values as text.

The function can return several codes/status updates for different scenarios:

Value	Description
FILE_OK	The function could read the file without problems.
FILE_NOT_EXIST	The specified file does not exist. This is returned if a local file has been specified that is not there. If the file was a HTTP request, this error is returned when the file does not exist on the host.
FILE_ERROR	This is returned if any error occurred while reading the content of a local file.
INVALID_HOST	This value is returned if the file was a HTTP request and the specified host does not exist.
NETWORK_ERROR	This value is returned on any network error, e.g. if no network adapter is available or the connection between the host and the client has been disconnected.

Example

For testing purposes there are two scripts which deliver random numbers at the end of this chapter. Or you could play with this test set-up.

```
@scriptname="ReadAsync Test Set-Up";
@author="";
@version="";
@description="";

string file = "C:\temp\src.txt"; //location of the source text file
string txt;
int interval = 600000; //interval = 10 minutes

void InitEffect()
{
    SetReadAsyncInterval(file, interval);
}

void PreRenderEffect()
{
    switch(ReadAsync(file, txt))
    {
        case FILE_OK : WriteText("FILE_OK"); SetText(txt); break; // txt is displayed
        case FILE_NOT_EXIST : WriteText("FILE_NOT_EXIST");break;
```

```
        case FILE_ERROR : WriteText("FILE_ERROR");break;
        case NETWORK_ERROR : WriteText("NETWORK_ERROR");break;
        case INVALID_HOST : WriteText("INVALID_HOST");break;
        default : break;
    }

    void PostRenderEffect()
    {

    }
```

Detailed Information About 'ReadAsync'

As you can see, it is neither necessary to explicitly open the file nor to close it. During the first call of the function, the specified file is opened. File reading happens asynchronously, which means that the function immediately returns a value but does not wait for the physical reading. Internally, the file will be read and the content is stored in a buffer. The function itself just reads from this buffer. This also means that after the first call of the function, it is more likely that no text will be read. There is a big advantage of this behavior. It is not necessary for the script to wait for potentially long reading times, which may occur, especially if you read content from internet servers.

Setting The Reading Interval

The file will be continuously read in the background. *ReadAsync* always receives the result of the last reading process. It is possible to control the interval of the reading process. The default value is 1000ms which means that an opened file will be read one time each second. The function

```
int SetReadAsyncInterval(string file, int interval)
```

sets the reading interval for a certain file. The *int interval* is given in milliseconds. The minimum reading interval is 10 ms. Imagine your sensor writes data every 500 ms. Therefore, you can set the interval to 500ms in order to let MADRIX instantly show the values. Or you can set it higher, e.g. to 5000 ms (5 seconds), in order to save resources.

If the file in question has not been opened yet, e.g. by a call of *ReadAsync*, the file will be opened to beginn internal reading. It makes sense to set the reading interval within the *InitEffect* function in order to have the content of the file read before the first call of *ReadAsync*, which is perhaps located in *RenderEffect*.

Tips For Using Files In MADRIX Script

Up to now, MADRIX Script is not designed to operate on strings. There aren't any proper functions available that may help you to parse strings or even manipulate them. Furthermore, such functionality requires a lot of computing time, which is not necessarily available within MADRIX and therefore within MADRIX Script. It may be better to have external tools (e.g. python scripts, PHP, or Visual basic scripts) which prepare files for MADRIX in order to have faster scripts.

Another interesting possibility is to have interactive scripts. Imagine a small application which retrieves input from an user and writes it to a file. A script may read the file and react to the input.

The following scripts are external scripts which may be used to test the script above or to play around with the reading functionality of MADRIX Script. Both scripts deliver random numbers using a local file or via HTTP request.

A Python Script To Create Random Numbers

The following script creates random values between 0 and 1 and writes them to the file "C:\temp\src.txt". In order to use a different file, set the *file* variable in the third line of the script to a different one. Please do not forget to change the script above to read the same file. Each second one value is written. In order to test HTTP functionality, this script may also run on a web server and you can request the written file from the web server.

```
import time
import random

file = "C:/temp/src.txt"

while True:
    f = random.random()
    s = "%.2f" % f
    try:
        wf = open(file, "w")
        wf.write(s)
```

```
        wf.close()
    except:
        print "Can't open file"

    print "Wert %.2f" %f

    time.sleep(1)
```

A PHP Script To Create Random Numbers

The following PHP script delivers a random number between 0 and 1 each time it is called.

```
<?php
    echo (rand() / getrandmax()) . "\n";
?>
```

2.3.9 Using Comments

During your study of this manual, surely you have encountered source code examples with text that is not part of the actual script. These so-called comments are a help for the programmer and other users of the script. There are two different kinds of comments in MADRIX Script.

Single line comments are induced with "//" and they end at the end of the line.

```
//This is a comment about a single line
//This is the next line
```

Multi-line comments are induced with "/*" and end with "*/". You have the possibility to inherit any comments in multi-line comments.

```
/* a comment starts here
/*
    one comment more
*/

*/ end of the complete comment
```

Comments are used for a better readability and understandability of the source code.

2.3.10 Including Extra Information

It is possible to provide some additional information about a script. This includes a script name, a

version number, the author's name, and a description. The information is visible within a script, because it is written into the Script Editor's input field if it has been loaded as compiled script.

Here is an example on how the information can be included:

```
@name="Name of the script"  
@version="1.24";  
@author="MichaelK";  
@description="Any text which describes the following script.";
```

As shown above, information is set using the following syntax:

```
@INFORMATION="any string";
```

Values after "=" have to be a string within double quotes. Therefore, it is also possible to set version to "1.2a" or any other string. Please note case sensitivity. It is not possible to set any of the values within a function and it is recommended to write the information at the beginning of a script. Overwriting one of the values results in a warning. The summary contains an [overview about the information that can be included](#) in a script.

2.4 Advanced Techniques

Overview

The following chapters describe advanced techniques for rendering effects or manipulating settings.

First, several higher functions are described which will foster your knowledge about MADRIX Script. Then map and tile settings as well as usage of mix modes is described in detail. Additionally, we included information referring to Sound2Light and Music2Light and how it can be used to create and manipulate effects.

2.4.1 Draw And Render Functions

MADRIX Script provides some low level drawing functions, like *SetPixel*, or functions to draw lines, circles, etc. But there are also more complex functions. Those are described in the following chapters.

2.4.1.1 Pixels Vs. Vectors

Introduction

For some effects, like the random SCE Color Fill, the matrix size does not matter. The matrix will be filled up with pixels, whatever the size of the matrix is. But what about an effect that draws vertical lines onto the matrix from top to bottom. It is easily possible to write such an effect for a well-known matrix size. The disadvantage is that if the size of the matrix changes or if someone else wants to use the script, it has to be changed to fit the new matrix size. Or you could be aware of the matrix size and calculate the number of pixels for the line to be drawn.

To have an easier way to create effects that are scalable, MADRIX Script provides two versions for the majority of render functions. One returns exactly the pixel coordinates as parameters and the other gets relative coordinates between 0.0 and 1.0.

Rules Of Calling Functions

Overview

The name of a draw or render function always starts with a description of what it does, like *Draw* or *Fill*. It is followed by *Pixel* or *Vector* to describe whether the function gets pixel coordinates or relative coordinates. The final part describes the application of the function, like *Rect* or *Line*. Here are some examples:

- **FillPixelRect** - Fills a rectangle
- **DrawVectorCircle** - Draws an unfilled circle
- **ShiftPixelMatrix** - Shifts the matrix

Using Absolute Coordinates

Functions with *Pixel* in their name operate on absolute pixel values. Picture a rectangle which is drawn from $x=5$, $y=5$ with a size of 10. The object will always be rendered starting from 5, 5 to 15, 15, on each matrix. On the one hand, the rectangle appears to be very small on larger matrices. On the other hand, it seems to be bigger on small matrices because it fills a larger area of the matrix.

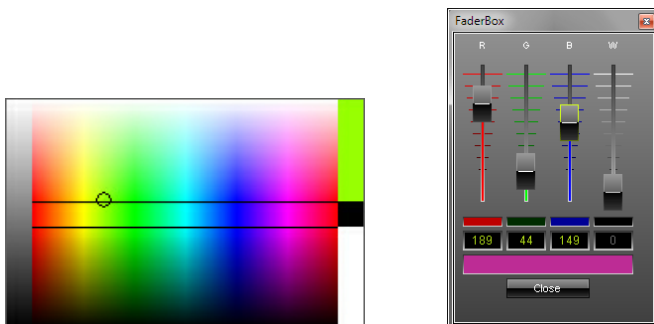
Using Relative Coordinates

Functions with *Vector* in their name operate on relative values. 0, 0 stands for the upper left corner of the effects matrix. 1.0 describes its lower right corner, or its whole length and height. A rectangle which is drawn from 0.25 with a size of 0.5 will look the same on every matrix. Its size is simply the half of the matrix size. On a matrix with more pixels the rectangle is also drawn with more pixels compared to fewer pixels on a smaller matrix. Whereas the usage of absolute coordinates dictates the utilization of the same amount of pixels.

2.4.1.2 Using Colors

Basically, there are three different possibilities an effect allows to choose a color.

Color Picker/FaderBox



Some simpler effects, such as SCE Color or SCE Pulse/ Stroboscope, only support one particular color at a time. Then, make use of the following functions:

Function	Description
void SetColor (color col)	Sets the color for the effect.
color GetColor ()	Returns the currently set color.

Please note: Some effects may have deviant functions, such as **SetFilterColor** or **SetTextColor**.

Example

Insert the following example into the Effect Macro Editor of the SCE Color effect.

```
@scriptname="Color Picker Example";
@author="";
@version="";
@description="";

color c = {111,111,111};

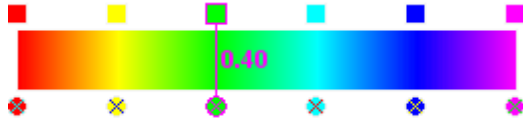
void InitEffect()
{

}

void PreRenderEffect()
{
    SetColor(c);
}

void PostRenderEffect()
{
    color g = GetColor();
    WriteText("Red: "+(string)g.r+", Green: "+(string)g.g+", Blue: "+(string)g.b);
}
```

Color Ramp (Control)



The Color Ramp appears in effects, like SCE Color Ramp, SCE Plasma, or S2L Waveform. Use the following functions to set the colors:

Please note: Not all effects may support all functions.

Function	Description
void SetColor (int index, color c)	Sets the <i>color c</i> at the specified <i>index</i> in the Color Ramp. If <i>index</i> is out of range, nothing happens.
color GetColor (int index)	Returns the <i>color</i> with the specified <i>index</i> in the Color Ramp. If the <i>index</i> is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the amount of colors currently used by the Color Ramp.
void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the <i>index</i> is lower or equal to 0, the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are 1 (On) or 0 (Off).
void RemoveColor (int index)	Removes the color at the specified <i>index</i> . If the given <i>index</i> is out of range, nothing happens.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the color position of the given <i>index</i> .
void SetColorFade (int index, fade)	Sets the color fade option for the given <i>index</i> . Valid values for <i>fade</i> are 1 (On) or 0 (Off).
int GetColorFade (int index)	Returns the color fade option for the given <i>index</i> .
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.

Example

Insert the following example into the Effect Macro Editor of the SCE Color Ramp effect.

```
@scriptname="";
@author="";
@version="";
@description="";

color c = {222,222,222};

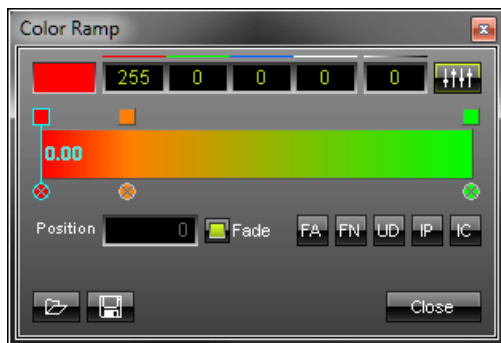
void InitEffect()
{

}

void PreRenderEffect()
{
    SetColor(2,c);
    int n = GetColorCount();
    WriteText((string)n);
}

void PostRenderEffect()
{
    color g = GetColor(2);    //the color index starts with 0
    WriteText("Red: "+(string)g.r+", Green: "+(string)g.g+", Blue: "+(string)g.b);
}
```

Color Ramp (Dialog)



The Color Ramp Dialog appears in effects, like SCE Fire, S2L Equalizer, and S2L Level Meter. Use the following functions to set the colors:

Please note: Not all effects may support all functions.

Function	Description
void SetColor (int index, color c)	Sets the <i>color c</i> at the specified <i>index</i> in the Color Ramp. If <i>index</i> is out of range, nothing happens.
color GetColor (int index)	Returns the <i>color</i> with the specified <i>index</i> in the Color Ramp. If the <i>index</i> is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the amount of colors currently used by the Color Ramp.
void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the <i>index</i> is lower or equal to 0, the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are <i>1</i> (On) or <i>0</i> (Off).
void RemoveColor (int index)	Removes the color at the specified <i>index</i> . If the given <i>index</i> is out of range, nothing happens.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the color position of the given <i>index</i> .
void SetColorFade (int index, fade)	Sets the color fade option for the given <i>index</i> . Valid values for <i>fade</i> are <i>1</i> (On) or <i>0</i> (Off).
int GetColorFade (int index)	Returns the color fade option for the given <i>index</i> .
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.

Example

Insert the following example into the Effect Macro Editor of the S2L Equalizer effect and monitor the Color Ramp dialog.

```
@author="jky";
@version="1.0";
@description="change second color from colorramp";
float pos;

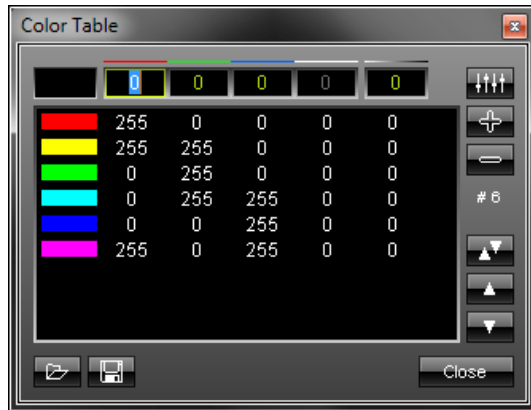
void InitEffect()
{
    pos=0.0;
}

void PreRenderEffect()
{
```

```
        if(GetColorCount(>2)
        {
            pos+=0.01;
            pos=fmod(pos,1.0);
            SetColorPosition(1,pos);
        }
    }

void PostRenderEffect()
{}
```

Color Table



SCE Bounce and SCE Shapes are two exemplary effects that use the so-called Color Table. Use the following functions to set the colors:

Function	Description
void SetColor (int idx, color c)	Sets the color with the specified index to the given color value. If the index is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the current number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color at the specified index. If the given index is out of range, nothing happens.

Please note: Not every function might be available for each MADRIX Effect. Also, some MADRIX effects require at least 2 entries in the Color Table. You will not be able to overwrite this requirement with a script. The S2L Level Ring effect always includes 3 entries. You cannot add or delete the number of colors, only the colors itself.

Example

Insert the following example into the Effect Macro Editor of the SCE Bounce effect.

```
@scriptname=" ";
@author=" ";
```

```

@version="";
@description="";

color c1 = {255,255,255};
color c2 = {255,0,0};

void InitEffect()
{
    AddColor(2,c2);
    RemoveColor(3);
}

void PreRenderEffect()
{
    SetColor(2,c1);
    int n = GetColorCount();
    WriteText((string)n);
}



void PostRenderEffect()
{
    color g = GetColor(2);    //retrieving color 3, the color index starts with 0
    WriteText("Red: "+(string)g.r+", Green: "+(string)g.g+", Blue: "+(string)g.b);
}

```

2.4.1.3 Predefined Colors

There are several predefined colors available for usage with draw functions. They are almost the same colors like known from HTML. The following table provides an overview. The white channel of these colors is 0.

Color	Description
WHITE	 White color without alpha.
BLACK	 Black color without alpha.
RED	 Red color without alpha.
MAROON	 Maroon color without alpha.
GREEN	 Green color without alpha.

MADRIX_GREEN	 MADRIX green without alpha.
BLUE	 Blue color without alpha.
NAVY	 Navy color without alpha.
AQUA	 Aqua color without alpha.
TEAL	 Teal color without alpha.
FUCHSIA	 Fuchsia color without alpha.
PURPLE	 Purple color without alpha.
YELLOW	 Yellow color without alpha.
OLIVE	 Olive color without alpha.
GRAY	 Gray color without alpha.
SILVER	 Silver color without alpha.

2.4.1.4 Using Filters

Introduction

You can use filters in order to render your output differently.

The functions looks as follows:

```
void Filter(int filter);
```

This function basically works whenever a matrix is used, which is the case with the MAS Script Effect, Macros for Effects, the Main Output Macro, and the Storage Place Macro. Insert the function into *PostRenderEffect* in order to use it.

Filters are useful tools to manipulate every MADRIX effect, videos, and images.

Several filters can be applied/used at the same time.

For MAS Script Effect, Macros for Effects, and the Storage Place Macro you can also use:

```
void SetFilter(int filter);
```

Types Of Filters

Filters are divided into several groups: general filters, blur/sharpen filters, color correction filters, color mask filters, style filters, transformation filters.

Constants for Filters / Storage Place and Layer Effects (FX)	
General Filters	
int FILTER_NONE	Deactivates the filter.
Blur/Sharpen Filters	
int FILTER_BLUR	This filter blurs the output.
int FILTER_BLUR_BSPLINE	This filter blurs the output applying a B-spline.
int FILTER_BLUR_CATMULL_ROM	This filter blurs the output applying a Catmull-Rom spline.
int FILTER_BLUR_GAUSS	This filter blurs the output applying the Gaussian function.
int FILTER_BLUR_MITCHELL	This filter blurs the output applying the Mitchell-Netravali function.
int FILTER_SHARPE	This filter sharpens the output.
Color Correction Filters	
int FILTER_BRIGHTEN	The brighten filter to light up the whole matrix.
int FILTER_DARKEN	The darken filter to darken the whole matrix.
int FILTER_GREYSCALE	The greyscale filter to render the matrix greyscale, i.e. in grey colors.
int FILTER_INVERT_COLOR	The invert color filter to invert every color channel.
Color Mask Filters	

Constants for Filters / Storage Place and Layer Effects (FX)	
int FILTER_RED	The red filter to filter out every color except the red color channel.
int FILTER_GREEN	The green filter to filter out every color except the green color channel.
int FILTER_BLUE	The blue filter to filter out every color except the blue color channel.
int FILTER_WHITE	The white filter to filter out every color except the white color channel.
int FILTER_RED_GREEN	The red/green filter to filter out every color except the red and the green color channel.
int FILTER_RED_BLUE	The red/blue filter to filter out every color except the red and the blue color channel.
int FILTER_GREEN_BLUE	The green/blue filter to filter out every color except the green and the blue color channel.
int FILTER_RED_WHITE	The red/white filter to filter out every color except the red and the white color channel.
int FILTER_GREEN_WHITE	The green/white filter to filter out every color except the green and the white color channel.
int FILTER_BLUE_WHITE	The blue/white filter to filter out every color except the blue and the white color channel.
int FILTER_RED_GREEN_BLUE	The red/green/blue filter to filter out every color except the red, the green, and the blue color channel.
int FILTER_RED_GREEN_WHITE	The red/green/white filter to filter out every color except the red, the green, and the white color channel.
int FILTER_RED_BLUE_WHITE	The red/blue/white filter to filter out every color except the red, the blue, and the white color channel.
int FILTER_GREEN_BLUE_WHITE	The green/blue/white filter to filter out every color except the green, the blue, and the white color channel.
Style Filters	
int FILTER_EDGES	The edges filter to make the edges of objects/motifs stand out.
int FILTER_EDGES_POPUP	The edges popup filter to make the edges of objects/motifs stand out.
int FILTER_EMOSS	The emboss filter to create an image with just highlights and shadows.
int FILTER_EMOSS_POPUP	The emboss popup filter to create an image with just highlights and shadows depending on the motif.
Transformation Filters	
int FILTER_INVERT_H_MATRIX	The filter flips the matrix horizontally.
int FILTER_INVERT_V_MATRIX	The filter flips the matrix vertically.
int FILTER_INVERT_HV_MATRIX	The filter flips the matrix horizontally and vertically. Therefore it instantly rotates the matrix by 180°.

Example

Paste the following example into the Main Output Macro Editor to see how this filter inverts all colors of your main output.

```
@scriptname="Filter: Invert Colors";
@author=" ";
@version="2.8";
@description=" ";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    Filter(FILTER_INVERT_COLOR);
}
```

2.4.1.5 'ShiftMatrix'

The shifting function allows you to move the content of the matrix into a given direction. It is declared as following:

```
void ShiftPixelMatrix(int x, int y, int w, int h, int dir, int step)
void ShiftVectorMatrix(float x, float y, float w, float h, int dir, float step)
```

Again there are two possibilities. One is used with absolute pixel coordinates and values, and one uses relative coordinates and values between 0 and 1.

x , y , w , and h define the pixel area that should be shifted. $step$ defines how many pixels the content should be shifted into a given direction, specified by dir . For dir the **SHIFT_ direction values** are allowed. If the value of dir is invalid, the default direction `SHIFT_TOP` will be used.

The following script fills the matrix with yellow and draws a red cross onto it during initialization. During the rendering the content of the whole matrix is shifted downwards. Hence, the cross is moving to the bottom of the matrix. You can simply copy and paste it.

```
void InitEffect()
{
    color colbg = {255,255};
```



```
    color col ={255};  
    Clear(colbg);  
    DrawVectorCross(col,0.0,0.0,1.0,1.0);  
}  
  
void RenderEffect()  
{  
    ShiftPixelMatrix(0,0,GetMatrixWidth(),GetMatrixHeight(),SHIFT_DOWN,1);  
}
```

The script also demonstrates an important behavior of the function. As one can see, the matrix remains yellow, but the cross moves to the bottom. Furthermore, red lines will be drawn on the left and right side. This is due to the fact that the *Shift* function copies the complete content of the matrix and redraws this picture on the new position. While the the content is moeved into the given direction, the original matrix is left unchanged. This leaves the first pixel "line" unchanged in our example.

2.4.1.6 'DrawPixelArea'

Two specific functions will be described in this chapter. First, MADRIX Script provides a function to be able to retrieve the content of a specific area of the matrix. Second, there is a function which is able to draw pixels onto the matrix using the obtained data field(s) as source.

Retrieving Content Of The Matrix

The function *GetPixelArea* retrieves data from the matrix and stores it into a 2-dimensional field of colors. Data is stored in the background at a certain position of the virtual matrix.

```
void GetPixelArea(matrix[][], int xSrc, int ySrc, int w, int h,  
int xDst, int yDst)
```

Explanation:

- *matrix[]* is a 2 dimensional field of colors in which the content of the virtual matrix is saved.
- *xSrc*, *ySrc* describe the position of the source area (upper left corner). The default values are 0.

- w , h describe the width and height of the source area. The default values are -1.
A value of -1 means that the whole width or height of the virtual matrix will be retrieved (the complete matrix).

- $xDst$ and $yDst$ describe the position of the destination area (upper left corner). The default values are 0.

The retrieved field will be stored in the background. It will be stored at a certain position of a matrix in the background. As such, you can define an individual target destination.

This behavior allows you, **for example**, to retrieve multiple source areas by calling *GetPixelArea* several times and to draw them only once by calling *DrawPixelArea* one time (see below).

If your field is larger than your target destination allows, it will be reduced to fit the size of the virtual matrix.

- In order to retrieve the whole matrix into the given field, it is possible to just call:

```
color matrix[][];  
GetPixelArea(matrix);
```

- Summary: You can store the complete virtual matrix or only parts of it in a field. The field can be stored at the default position or an individual position. The field is stored on a matrix in the background. The background matrix acts as source for *DrawPixelArea*. It can contain several fields.

Drawing Content Onto The Matrix

The function *DrawPixelArea* copies data from a 2-dimensional field of colors from the background and renders it onto the actual matrix.

```
void DrawPixelArea(matrix[][], int xDst, int yDst, int w, int h,  
int xSrc, int ySrc, color filter)
```

Explanation:

- *matrix[]* is a 2-dimensional field of colors that holds the source for *DrawPixelArea*. Use *GetPixelArea* as described above to retrieve the data.

- *xDst*, *yDst* describe the destination area. The default values are 0.

This allows you to draw the field onto the default position of your virtual matrix or an individual position.

If the source field is larger than your target destination allows, it will be reduced to fit the size of the virtual matrix.

- *w*, *h* describe the width and height of the render area. The default values are -1.

A value of -1 means that the whole width or height of the given field will be copied to the virtual matrix (the complete field).

- *xSrc*, *ySrc* describe the source area. The default values are 0.

DrawPixelArea can use and render the complete background matrix that was retrieved with *GetPixelArea*. Or it can only access and render a certain part of it.

- *filter* determines which color channels to draw. Default values are {255, 255, 255, 255, 255} for RGBWA. For more information, please see below.

- In order to draw the whole matrix it is possible to call:

```
color matrix[][];  
DrawPixelArea(matrix);
```

- Summary: *DrawPixelArea* can access the background matrix that was created with *GetPixelArea*. With *DrawPixelArea* you can render the complete field or only parts of it onto your virtual matrix. The field can be drawn at the default or an individual position.
- (For the drawing operation, it is assumed that the field describes a rectangular area in which every single line has the same number of columns.)

Using Filters For Drawing

The filter can be used to leave different color channels of the matrix unchanged. The following example draws just the red and the alpha channel onto the matrix and leaves other channels unchanged:

```
color matrix[][];  
color filter = {255, 0, 0, 0, 255};  
DrawPixelArea(matrix, 0, 0, -1, -1, 0, 0, filter);
```

Examples

DrawPixelArea (Without Color Filter)

The first example for the MAS Script Effect will simply draw a small, red square in the upper left corner of the matrix with a green center. The colors drawn are defined in the field variable *matrix* [[]] in *InitEffect*.

```
@scriptname="";
@author="";
@version="";
@description="";

color matrix[][];

void InitEffect()
{
    matrix[0][0]=RED;
    matrix[0][1]=RED;
    matrix[0][2]=RED;
    matrix[0][3]=RED;
    matrix[1][0]=RED;
    matrix[1][1]=GREEN;
    matrix[1][2]=GREEN;
    matrix[1][3]=RED;
    matrix[2][0]=RED;
    matrix[2][1]=GREEN;
    matrix[2][2]=GREEN;
    matrix[2][3]=RED;
    matrix[3][0]=RED;
    matrix[3][1]=RED;
    matrix[3][2]=RED;
    matrix[3][3]=RED;
}

void RenderEffect()
{
    DrawPixelArea(matrix, 0, 0, -1, -1, 0, 0);
}
```

DrawPixelArea (With Color Filter)

The second example for the MAS Script Effect uses a color filter. You will see that it filters the red color channel. As a result, the green center will remain, while the red square will not be displayed anymore.

```
@scriptname="";
@author="";
```

```
@version="";
@description="";

color matrix[][];
color filter = {0, 255, 255, 255};

void InitEffect()
{
    matrix[0][0]=RED;
    matrix[0][1]=RED;
    matrix[0][2]=RED;
    matrix[0][3]=RED;
    matrix[1][0]=RED;
    matrix[1][1]=GREEN;
    matrix[1][2]=GREEN;
    matrix[1][3]=RED;
    matrix[2][0]=RED;
    matrix[2][1]=GREEN;
    matrix[2][2]=GREEN;
    matrix[2][3]=RED;
    matrix[3][0]=RED;
    matrix[3][1]=RED;
    matrix[3][2]=RED;
    matrix[3][3]=RED;
}

void RenderEffect()
{
    DrawPixelArea(matrix, 0, 0, -1, -1, 0, 0, filter);
}
```

2.4.1.7 'PixelTranspose'

Introduction

PixelTranspose is a technic to transpose (move) pixels from their origin/source (srcX and srcY coordinate) to a new destination. Three steps are necessary to perform a pixel transposition.

1. Creating the pixel transpose table which holds the information for each pixel that shall be moved.
2. Setting or adding the information (source and destination coordinates) for each of those pixels to the table.
3. Executing the pixel transposition.

1. Creating The Pixel Transpose Table:

```
void CreatePixelTransposeTable(int size, int growsize)
```

The parameter *size* describes the amount of pixels in the table. The second parameter *growsize* describes the size that will be used to grow the table by using *AddPixelTransposeEntry* if the predefined size of *PixelTransposeTable* is reached. It is not necessary to use the second parameter because by default the *growsize* is set to 128.

2. Setting Or Adding Information

```
void SetPixelTransposeEntry(int idx, int srcX, int srcY, int destX, int destY)
```

Using *SetPixelTransposeEntry* requires 5 parameters. The first parameter *idx* defines the index of the predefined table. This index starts counting with zero and has to be lower than the *size* value of *CreatePixelTransposeTable(int size, int growsize)*. The second and third parameter, *srcX* and *srcY*, describe the source coordinate and the fourth and fifth parameter, *destX* and *destY*, set the destination coordinates of the pixel.

```
void AddPixelTransposeEntry(int srcX, int srcY, int destX, int destY)
```

Using *AddPixelTransposeEntry* requires only 4 parameters, i.e. the source and destination coordinates. The function validates if an entry already exists. If this is not the case, the function adds the entry to the end of the pixel transpose table. If the predefined *size* in *CreatePixelTransposeTable(int size, int growsize)* is then exceeded, the table automatically grows by the size defined with *growsize*.

The execution of *SetPixelTransposeEntry* is performed much faster than that of *AddPixelTransposeEntry*.

3. Executing The Pixel Transposition

```
void ExecutePixelTranspose(int clear)
```

Using this function executes all pixel transpositions that are defined in the pixel transpose table. The *clear* parameter defines how the part of the matrix is handled which is not defined as destination. If the *clear* parameter is set to `CLEAR`, the part will be erased using black. Otherwise, if this parameter is set to `NOCLEAR`, the color values will be left like they were before.

4. Releasing The Pixel Transpose Table

```
void ReleasePixelTransposeTable()
```

Using this function releases the created transpose table. That means, that the reserved memory is set free.

Examples

The following three examples will rotate the Main Output using the pixel transpose technique. Please note that this only works on quadratic matrices. Just use the SCE Color Ramp effect on Preview A and insert the following source code into the Main Output Macro. The difference will be easily visible.

Clockwise Rotation

This macro rotates the output 90° clockwise.

```
@scriptname="Output Rotation";
@author="jky";
@version="2.9";
@description="Rotates the main output";

int Init=0;

void InitEffect()
{
    int w = GetMatrixWidth();
    int h = GetMatrixHeight();
    Init=0;
```

```

if(w==h) //this example runs only on quadratic matrices

{
    int idx = 0;
    CreatePixelTransposeTable(w*h); //make a table with w*h entities

    for(int y=0;y<h;y++)
    {
        for(int x=0;x<w;x++)

            {
                SetPixelTransposeEntry(idx,x,y,h-y-1,x); //rotate clockwise
                idx++;
            }

        Init=1; //init ready, can use ExecutePixelTranspose()
    }

    else
    {
        WriteText("This script runs only on quadratic matrices,");
        WriteText("but your matrix is "+(string)w+"x"+(string)h);
    }

}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    if(Init==1)
        ExecutePixelTranspose(CLEAR); //execute to transpose all pixels
                                         //and to clear all non-transposed pixels
}

void MatrixSizeChanged()
{
    ReleasePixelTransposeTable(); //release the old transpose table if existent
    InitEffect();
}

```

Counter-Clockwise Rotation

This macro rotates the output by 90° counter-clockwise.

```

@scriptname="Output Rotation";
@author="jky";
@version="2.9";
@description="Rotates the main output";

```



```
int Init=0;

void InitEffect()

{
    int w = GetMatrixWidth();
    int h = GetMatrixHeight();
    Init=0;

    if(w==h) //this example runs only on quadratic matrices

    {
        int idx = 0;
        CreatePixelTransposeTable(w*h); //make a table with w*h entities

        for(int y=0;y<h;y++)
        {
            for(int x=0;x<w;x++)

            {
                SetPixelTransposeEntry(idx,x,y,y,w-x-1); //rotate counter-clockwise
                idx++;
            }

            Init=1; //init ready, can use ExecutePixelTranspose()

        }

    }

    else
    {
        WriteText("This script runs only on quadratic matrices,");
        WriteText("but your matrix is "+(string)w+"x"+(string)h);
    }

}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    if(Init==1)
    ExecutePixelTranspose(CLEAR); //execute to transpose all pixels
                                //and to clear all non-transposed pixels
}

void MatrixSizeChanged()
{
    ReleasePixelTransposeTable(); //release the old transpose table if existent
    InitEffect();
}
```

Mirror Diagonally

Use a SCE Color Ramp from bottom left to top right for example to see the result.

```
@scriptname="Output Rotation";
@author="jky";
@version="2.9";
@description="Rotates the main output";

int Init=0;

void InitEffect()

{
    int w = GetMatrixWidth();
    int h = GetMatrixHeight();
    Init=0;

    if(w==h) //this example runs only on quadratic matrices
    {
        int idx = 0;
        CreatePixelTransposeTable(w*h); //make a table with w*h entities

        for(int y=0;y<h;y++)
        {
            for(int x=0;x<w;x++)

            {
                SetPixelTransposeEntry(idx,x,y,y,x); //mirror diagonally
                idx++;
            }
        }

        Init=1; //init ready, can use ExecutePixelTranspose()
    }

    else
    {
        WriteText("This script runs only on quadratic matrices,");
        WriteText("but your matrix is "+(string)w+"x"+(string)h);
    }
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    if(Init==1)
        ExecutePixelTranspose(CLEAR); //execute to transpose all pixels
                                        //and to clear all non-transposed pixels
}
```

```
}

void MatrixSizeChanged()
{
    ReleasePixelTransposeTable(); //release the old transpose table if existent
    InitEffect();
}
```

2.4.1.8 'SetPixel'

Functionality

SetPixel functions offer the possibility to change the color of pixels. You can either specify a certain color or use greyscale.

Examples

SetPixel

To test this script, please use the MAS Script effect.

```
@scriptname="SetPixel test, use with MAS script effect";
@author="";
@version="";
@description="";

color col;
int maxX,maxY,x,y;
void InitEffect()
{
    maxX=GetMatrixWidth();
    maxY=GetMatrixHeight();
}

void RenderEffect()
{
    col.r=random(0,255);
    x=random(0,maxX-1);
    y=random(0,maxY-1);
    SetPixel(col,x,y);
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

SetPixel - Filling The Matrix

To test this script, please use the MAS Script effect.

This sample fills every pixel of every row of the matrix with the color white from left to right until the complete matrix is covered. Every second iteration black is used instead of white.

```
@scriptname="SetPixelSample";
@author="jky";
@version="MADRIX 2.13";
@description="a simple setpixel example to fill the matrix";

int x,y,c;
color col;

void InitEffect()
{
    x=0;
    y=0;
    c=0;
    col=WHITE;
}

void RenderEffect()
{
    SetPixel(col,x,y);
    x++;
    if(x>=GetMatrixWidth())
    {
        x=0;
        y++;
        if(y>=GetMatrixHeight())
        {
            y=0;
            c++;
            if(c%2==0)
                col=WHITE;
            else
                col=BLACK;
        }
    }
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

SetPixelGreyscale (MAS Script)

To test this script, please use the MAS Script effect.

```
@scriptname="sample of greyscale for a single pixel";
@author="";
@version="";
@description="";

int X,Y;
void InitEffect()
{
    X=GetMatrixHeight();
    Y=GetMatrixWidth();
    Clear(BLUE);
}

void RenderEffect()
{
    for(int i=0;i<X && i<Y;i++)
    {
        SetPixelGreyscale(i,i); // line from top left to bottom right
        SetPixelGreyscale(X-i-1,i); // line from top right to bottom left
    }
    // to render the complete matrix in greyscale, the greyscale() command
    // offers higher performance:
    // Greyscale();
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

SetPixelGreyscale (Macro)

To test this script, you can use the Main Output Macro. But first, please select for example the SCE ColorScroll effect in Storage A or B and display the effect on the output.

```
@scriptname="sample of greyscale for single pixel";
@author="";
@version="";
@description="";

int X,Y;
void InitEffect()
{
    X=GetMatrixWidth();
    Y=GetMatrixHeight();
}

void PreRenderEffect()
{
}
```

```
}

void PostRenderEffect()
{
    if(X>Y)// width larger than height
    {
        for(int i=0;i<Y;i++)
        {
            SetPixelGreyscale(i,i); // line from top left to bottom right
            SetPixelGreyscale(i,Y-i-1); // line from top right to bottom left
        }
    }
    else // height larger than width
    {
        for(int i=0;i<X;i++)
        {
            SetPixelGreyscale(i,i); // line from top left to bottom right
            SetPixelGreyscale(X-i-1,i); // line from top right to bottom left
        }
    }
    // to render the complete matrix in greyscale, the Greyscale() command
    // offers higher performance
    // Greyscale();
}
```

2.4.2 Manipulating Effects

With MADRIX Script it is possible to render live effects. Moreover, you can manipulate the effect settings. Those include the mapping area, the mix mode, the frame fade, and the submaster. This chapter describes several more complex settings which can be adjusted for an effect.

There are also functions available that are unique to single effects and therefore they can only be used with them. Examples include settings of directions or the text of the SCE Ticker effect.

2.4.2.1 Map An Effect

Functionality

MADRIX provides the possibility to map an effect to different areas of the matrix. Setting effect mapping parameters is also available using additional functions.

The functions discussed in this chapter provide almost the same functionality as the Map dialog used for effects in MADRIX. Therefore, the functionality of mapping is thoroughly discussed in the MADRIX manual. This manual only describes the functions provided by the scripting engine of MADRIX to manipulate map and tile settings of an effect.

Settings

Using Absolute Values

One possibility is the function *MapEffectPixel*. You will have to set the values of this function in pixels. It is declared as follows:

```
void MapEffectPixel(int x, int y, int w, int h)
```

x, *y* are the start coordinates to determine where the effect should be started on the virtual matrix. Negative values are allowed, too. *w*, *h* describe the size of the mapped matrix. Any value higher than 0 is valid. For example, to map the effect on 20, 10 with a size of 50 x 40 pixels, just write the following:

```
MapEffectPixel(20, 10, 50, 40);
```

Using Relative Values

Another possibility is the function *MapEffectVector*. It uses relative values between 0.0 and 1.0. Hereby, 1.0 represents the full size of the matrix. The declaration is the following:

```
void MapEffectVector(float x, float y, float w, float h)
```

x, *y* are the start coordinates to determine where the effect should be started on the virtual matrix. You can also use negative values. *w*, *h* describe the size of the mapped matrix. Here is an example: To map the effect using half the matrix size, centered, call the function as follows:

```
MapEffectVector(0.25, 0.25, 0.5, 0.5);
```

The following example uses negative start values. With negative coordinates the mapped matrix starts outside of the virtual matrix and has a height and width of 100% of the virtual matrix.

```
MapEffectVector(-0.5, -0.5, 1.0, 1.0);
```

Another possibility is to set the size of the mapped matrix to a larger size than the virtual matrix has. For that, you will have to use values greater than 1 for width and height. Negative values are not allowed.

Side Effects

If the effect matrix is mapped using a mapping function and you are changing the values for the width and height in your script, the effect will be reinitialized and restarted, beginning with a call of the function *InitEffect*. If the mapping only changes the position of the effect, the effect will not be reinitialized and restarted.

Getting The Current Map Settings

If you want to retrieve the values for the mapping, you should use the two functions *GetMapPixel* and *GetMapVector*. They are defined like this:

```
int GetMapPixel(int map[])  
int GetMapVector(float map[])
```

Both functions return *false* if no mapping is set. And both functions return *true* if mapping is set. The parameter *map* is a field which retrieves the mapping settings as follows:

- `map[0]` = x
- `map[1]` = y
- `map[2]` = width
- `map[3]` = height

This field can be found in the Map Effect dialog of MADRIX because it represent Pos X, Pos Y, size x, and Size Y in the section "Map (Position/Size)".

x, *y* describe the start coordinates and *width*, *height* describe the size of the mapped matrix. If *false* is returned, mapping is not activated and the standard values for *x*, *y*, *width*, and *height* (0.0, 0.0, 1.0, 1.0) are activate. Here is an example to show this. Just paste the source code into the MAS Script effect, use the map dialog and the Script Output window will display the information.

```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{
}

void RenderEffect()
{
    float map[];
    int result=GetMapVector( map);
    if(result==0)
        WriteText("mapping not used");
    else
        WriteText("mapping is used, map x: "+ (string)map[0]+", map y:
        "+(string)map[1] +", map width: "+(string)map[2]+", map height: "+(string)map[3]);
}
```

Ask If Mapping Is Activated

The function *int IsMapped()* returns *true* if mapping is active, or *false* if it is deactivated. Those are also the values the function *GetMap* returns.

Full Example

The following example fills the matrix with a random color which is changed every tenth call of the script. Furthermore, the effect is moved inside the matrix using mapping.

```
persistent float g_x;
persistent float g_y;
persistent int mode;
persistent int change;
persistent color g_col;

void InitEffect()
{
```

```
//start in upper left corner
g_x = -0.5;
g_y = -0.5;
change = 10;
//fill matrix with the color used last
DoPreRender();
}

void PreRenderEffect()
{
    /*
        ensures that after reloading (e.g. changing storage place)
        the matrix is filled with the color used last
    */
    Clear(g_col);
}

void RenderEffect()
{
    if(change++ >= 10)
    {
        color c = {random(0, 255), random(0, 255), random(0, 255),
            random(0, 255), random(128 / 2, 128 * 2)};
        g_col = c;
        Clear(c);
        change = 0;
    }
    mapEffect();
}

void mapEffect()
{
    const float move = 0.01;

    switch(mode)
    {
        case 0:
            g_y += move;
            if(g_y > 0.5)
            {
                g_y = 0.5;
                mode = 1;
            }
            break;

        case 1:
            g_x += move;
            if(g_x > 0.5)
            {
                g_x = 0.5;
                mode = 2;
            }
            break;

        case 2:
            g_y -= move;
```

```
        if(g_y < -0.5)
        {
            g_y = -0.5;
            mode = 3;
        }
        break;

    case 3:
        g_x -= move;
        if(g_x < -0.5)
        {
            g_x = -0.5;
            mode = 0;
        }
        break;
    } //switch[mode]
    MapEffectVector(g_x, g_y, 1.0, 1.0);
}
```

2.4.2.2 Tile An Effect

Settings

As is the case for mapping and drawing functions, for tiling functions there is a *vector* and a *pixel* version. They work with relative values/coordinates between 0 and 1 or with absolute pixel values. The function to set tiles for an effect is called:

- **MapTileEffectVector(float x, float y, float w, float h)**
- **MapTileEffectPixel(int x, int y, int w, int h)**

The parameters *x* and *y* describe the start coordinates, whereas *w* and *h* describe the width and height of the tile. Values which may lead to the fact that the tile is partial or fully outside of the effect matrix are also possible.

Side Effects

If the size of the effect matrix is changed due to a call of the function *MapTileEffect*, the script is reinitialized and restarted with a call of *InitEffect*. This does not happen if solely the position of the effect matrix is changed.

Getting the Current Tile Settings

Via a call of one of the following functions, it is also possible to retrieve the current tile settings:

- `int GetMapTileEffectVector(float map[])`
- `int GetMapTileEffectPixel(int map[])`

Both functions return *true* if the effect is currently mapped and *false* if the effect uses the whole matrix. The parameter *map* needs to be a field, which is filled with the current tile settings as described below:

- `map[0] = x`
- `map[1] = y`
- `map[2] = width`
- `map[3] = height`

Values *x* and *y* describe the start coordinates of the tile relative to the eventually mapped effect matrix. The parameters *width* and *height* describe the width and height of the tile and therefore of the effect's matrix.

Setting and Getting Tile Modes

Setting Modes

For tiles it is possible to set a mirror and a tile mode. This can be done using the functions

- `void SetMapModeMirror(int mirrorMode)`
- `void SetMapModeTile(int tileMode)`

For the parameter *mirrorMode* one of the **MAP_MIRROR_x constants** must be used. The parameter *tileMode* needs to be one of the **MAP_TILE_x constants**. (Those constants are describe in the List of Global Variables and Constants).

Retrieving Modes

There is also the possibility to get the current settings.

- `int GetMapModeMirror()`
- `int GetMapModeTile()`

GetMapModeMirror returns a **MAP_MIRROR_x constant**, which describes the current mirror settings. *GetMapModeTile* returns a **MAP_TILE_x constant**, which describes the current tile settings.

2.4.2.3 Mix Modes

Available Functions

The function *SetMixMode* offers the possibility to set the mix mode of an effect. The function *GetMixMode* retrieves the mix mode currently in use. They are declared as follows:

- `void SetMixMode(int mode)`
- `int GetMixMode()`

For the parameter *mode* one of the values described in the table below must be used. If *mode* has an invalid value, nothing will happen and a message will be displayed inside the Script output window of the Script editor. You can find more information about mix modes and their usage in the MADRIX manual.

Available Parameters

Identifier/ Mix Mode	Description
MIXMODE_NORMAL	The normal mix mode.
MIXMODE_DARKEN	The darken mix mode.
MIXMODE_MULTIPLY	The multiply/mask mix mode.
MIXMODE_COLORBURN	The color burn mix mode.
MIXMODE_LINEARBURN	The linear burn mix mode.
MIXMODE_LIGHTEN	The lighten mix mode.
MIXMODE_SCREEN	The screen mix mode.
MIXMODE_COLORDODGE	The color dodge mix mode.
MIXMODE_LINEARDODGE	The linear dodge mix mode.
MIXMODE_OVERLAY	The overlay mix mode.
MIXMODE_SOFTLIGHT	The soft light mix mode.
MIXMODE_HARDLIGHT	The hard light mix mode.
MIXMODE_VIVIDLIGHT	The vivid light mix mode.
MIXMODE_LINEARLIGHT	The linear light mix mode.
MIXMODE_PINLIGHT	The pin light mix mode.
MIXMODE_HARDMIX	The hard mix mix mode.
MIXMODE_DIFFERENCE	The difference mix mode.
MIXMODE_EXCLUSION	The exclusion mix mode.
MIXMODE_AND	The AND mix mode.
MIXMODE_OR	The OR mix mode.
MIXMODE_XOR	The XOR mix mode.
MIXMODE_NAND	The NAND mix mode.
MIXMODE_NOR	The NOR mix mode.
MIXMODE_MASK	The mask mix mode.

2.4.3 Sound2Light & Music2Light

MADRIX can analyze music with regard to different musical parameters, like tones, bass, tonality, etc., using Sound2Light (S2L) or Music2Light (M2L) effects. This data is also available in MADRIX Script and may be used to create even more fantastic effects, controlled by music.

It is necessary to differentiate between music and sound data. The term "sound data" refers to data based on the frequency of the given input signal. A common effect based on sound data is the equalizer. The volume is also this kind of sound data.

The notion "music data" refers to the information about the music itself, known from actual music theory. Therefore, MADRIX identifies tonality, intervals or the current tone (or chord) itself. So, using the tone and the tonality you can say that e.g. c major or d minor are currently played. There are numerous examples.

As is the case with M2L or S2L effects, if a script uses any functionality that needs the audio analysis, it is automatically started in MADRIX.

2.4.3.1 Sound2Light (S2L)

Functionality

As described above, sound data refers to frequency-based data. First of all, the function *GetSoundLevel* retrieves the volume of the two audio channels (stereo sound). The value returned ranges from 0 to 255, the lowest and highest level possible, respectively.

Moreover, other available data is frequency values. They are stored in the fields `SOUND_DATA_LEFT` and `SOUND_DATA_RIGHT`, which are both of the data type *int*. There are up to 511 values and each describes the volume of a **well-defined frequency**.

The *length*-operator of the **fields** tells how much valid data they contain. A check may be necessary and then just take the values that are actually available/provided to have a proper effect. But it is secure to always assume 511 present values.

In contrast to other dynamic fields, in MADRIX Script those fields will not grow because their size is fixed. Trying to get an invalid element, always results in 0.

The First S2L Example

The following example first of all selects a color depending on the volume of the music, which is then used as background color. Lines will be drawn onto the matrix that indicate the average volume of each single frequency.

```
const color LEFT_CHANNEL = {255, 0, 0, 128};
const color RIGHT_CHANNEL= {0, 255, 0, 128};

void InitEffect()
{
}

int avgFrequ(int field[])
{
    int result;

    //to avoid division by zero later on
    if(field.length > 0)
    {
        for(int i = 0; i < field.length; i++)
            result += field[i];

        result /= field.length;
    }

    return(result);
}

void RenderEffect()
{
    int valL = GetSoundLevel(0); //left channel
    int valR = GetSoundLevel(1); //right channel
    color c = {valL, valR, (valL * valR) % 255, (valR + valL) / 2, 0};

    float iHL = (float)avgFrequ(SOUND_DATA_LEFT) / (float)MAX_FREQUENCY_VOLUME;
    float iHR = (float)avgFrequ(SOUND_DATA_RIGHT) / (float)MAX_FREQUENCY_VOLUME;

    Clear(c);
    DrawVectorLine(LEFT_CHANNEL, 0.0, iHL, 1.0, iHL);
    DrawVectorLine(LEFT_CHANNEL, iHL, 0.0, iHL, 1.0);

    DrawVectorLine(LEFT_CHANNEL, 0.0, 1.0-iHL, 1.0, 1.0-iHL);
    DrawVectorLine(LEFT_CHANNEL, 1.0-iHL, 0.0, 1.0-iHL, 1.0);

    DrawVectorLine(RIGHT_CHANNEL, 0.0, iHR, 1.0, iHR);
    DrawVectorLine(RIGHT_CHANNEL, iHR, 0.0, iHR, 1.0);

    DrawVectorLine(RIGHT_CHANNEL, 0.0, 1.0-iHR, 1.0, 1.0-iHR);
    DrawVectorLine(RIGHT_CHANNEL, 1.0-iHR, 0.0, 1.0-iHR, 1.0);
}
```


Explanation:

First, several constants are declared. They are used as colors to draw lines for the right and the left channel.

The function *avgFrequ* is an assistant function. It iterates through a given field of *integer* values and returns the average of all given values. If the field is empty, 0 is returned.

The function *RenderEffect* stores the volume of the left and the right sound channel. This is done for performance reasons. Calling a function needs more time than using a variable which is done in the third line. Here, a color is build up defined by the volume levels. This color is used as background color later on.

Afterwards, for both the left and the right sound channel the average volume is calculated. In the same step, the results are divided by the maximal possible value in order to break them down to a value between 0.0 and 1.0. Then, the matrix is cleared with the color calculated before and at last, horizontal and vertical lines are drawn onto the matrix. Hereby, the vector version of the draw function is used to draw lines across the whole matrix using the level described by the average volume level.

The Second S2L Example

Another nice effect based on the same data is done by the following *RenderEffect* function. It uses a black background and draws the lines with the calculated color.

```
const color LEFT_CHANNEL = {255, 0, 0, 128};
const color RIGHT_CHANNEL= {0, 255, 0, 128};

void InitEffect()
{
}

int avgFrequ(int field[])
{
    int result;

    //to avoid division by zero later on
    if(field.length > 0)
    {
        for(int i = 0; i < field.length; i++)
            result += field[i];
    }
}
```

```
        result /= field.length;
    }

    return(result);
}

void RenderEffect()
{
    int valL = GetSoundLevel(0);
    int valR = GetSoundLevel(1);
    color c = {valL, valR, (valL * valR) % 255, (valR + valL) / 2, 0};

    float iHL = (float)avgFrequ(SOUND_DATA_LEFT) / (float)MAX_FREQUENCY_VOLUME;
    float iHR = (float)avgFrequ(SOUND_DATA_RIGHT) / (float)MAX_FREQUENCY_VOLUME;

    Clear();

    DrawVectorLine(c, 0.0, iHL, 1.0, iHL);
    DrawVectorLine(c, iHL, 0.0, iHL, 1.0);

    DrawVectorLine(c, 0.0, 1.0-iHL, 1.0, 1.0-iHL);
    DrawVectorLine(c, 1.0-iHL, 0.0, 1.0-iHL, 1.0);

    DrawVectorLine(c, 0.0, iHR, 1.0, iHR);
    DrawVectorLine(c, iHR, 0.0, iHR, 1.0);

    DrawVectorLine(c, 0.0, 1.0-iHR, 1.0, 1.0-iHR);
    DrawVectorLine(c, 1.0-iHR, 0.0, 1.0-iHR, 1.0);
}
```

2.4.3.2 Music2Light (M2L)

Introduction

Sound2Light effects are really nice, but an even more interesting feature of MADRIX is the ability to analyze music regarding music theoretical aspects. Tonality, scale or intervals are only some examples of the data that can be retrieved by the sound analysis. This chapter describes how to retrieve the data provided by MADRIX. However, it does not describe any music theory.

Using Tonality Ánd Scale

Let us start with tonality and scale of a chord. The two be retrieved via the functions *GetTonality* and *GetToneScale*, respectively. The exemplary sample source code below uses tonality and scale to select a color and the alpha value will be the background color. Please remember: an audio input signal is needed.

```
const color g_colorTable[] = {
    {255, 0, 0, 0},          //C
    {255, 128, 128, 128},   //C#
    {0, 255, 0, 0},         //D
    {128, 255,128,128},     //D#
    {0, 0, 255,0},          //E
    {255, 255, 0, 0},       //F
    {255, 255, 128, 128},   //F#
    {255, 0, 255, 0},       //G
    {255, 128,255,128},     //G#
    {255, 128, 0, 0},       //A
    {255, 100, 100, 100},   //A#
    {255, 255, 255, 128}   //H
};

void InitEffect()
{
}

void RenderEffect()
{
    int idx=GetTonality();
    if (idx>=0 && idx<=11) ClearColor(g_colorTable[idx]);
    else ClearColor(BLACK);
    int alpha = (255 / (1 + GetToneScale()));
    ClearAlpha(alpha);
}
```

Explanation:

The first thing is to create a color table in which each entry equals a tonality. If the tonality is undetermined, the function *GetTonality* results in -1. Therefore, we have to check if the value is a valid field index and eventually draw a black matrix. You could use the function *IsTonality* to check if the tonality was set or not.

Using Notes

MADRIX is able to identify the notes played in a song. The lowest note that can be evaluated is the C with 8.25 Hz. The highest note is an A with 14.08 kHz. There are two functions available to get information about the identified notes. *GetNoteValue* retrieves the volume of the given note and *IsNote* returns *true* if the given note has been detected, otherwise *false*. There is the function *GetAllNoteValues*, which fills an field with the volume of each note. Using this method for a lot of notes is much faster then calling *GetNoteValue*. An overview over which index corresponds to which note is given by the [list of notes](#).

The next example uses played notes to fill the matrix with different colors. It uses only the values of three frequencies of C.

```
void InitEffect()
{
}

void RenderEffect()
{
    float val[];
    //C with 528Hz
    val[0] = (float)GetNoteValue(72) / 127.0;
    //C with 1.056KHz
    val[1] = (float)GetNoteValue(84) / 127.0;
    //C with 2.112KHz
    val[2] = (float)GetNoteValue(96) / 127.0;

    color c;
    c.r = (int)(255.0 * val[0]);
    c.g = (int)(255.0 * val[1]);
    c.b = (int)(255.0 * val[2]);

    Clear(c);
}
```

Explanation:

First, the levels of the notes are retrieved and normalized to the range of 0.0 to 1.0. Hereby, only three C notes are used. Then, a color is initialized and the matrix is filled.

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Please note: The old functions *GetValue* and *GetAllValues* still exist, but have been declared as deprecated and should not be used anymore. They may be removed in coming versions of MADRIX Script. Use *GetNoteValue* and *GetAllNoteValues* instead.

Using Intervals

There are similar functions to get information about the intervals indexed from 0 (small second interval) to 10 (large seventh interval). The function *IsInterval* returns *true* if the specified interval could be analyzed, otherwise *false*. Again, the function *GetAllIntervals* fills an field rapidly, each element with either *true* (interval was analyzed) or *false* (interval was not analyzed). The following short example clarifies the usage of *GetAllIntervals*:

```
const int middle=GetMatrixHeight()/2;
int buf[];
int xStep;

void InitEffect()
{
    GetAllIntervals(buf);
    xStep=max(GetMatrixWidth()/buf.length,1);
}

void RenderEffect()
{
    ClearAlpha(255);
    GetAllIntervals(buf);
    for (int i=0;i<buf.length;i++)
    {
        DrawPixelLine(WHITE,i*xStep,!buf[i]*middle,i*xStep,(1+!buf[i])*middle);
    }
}
```

Explanation:

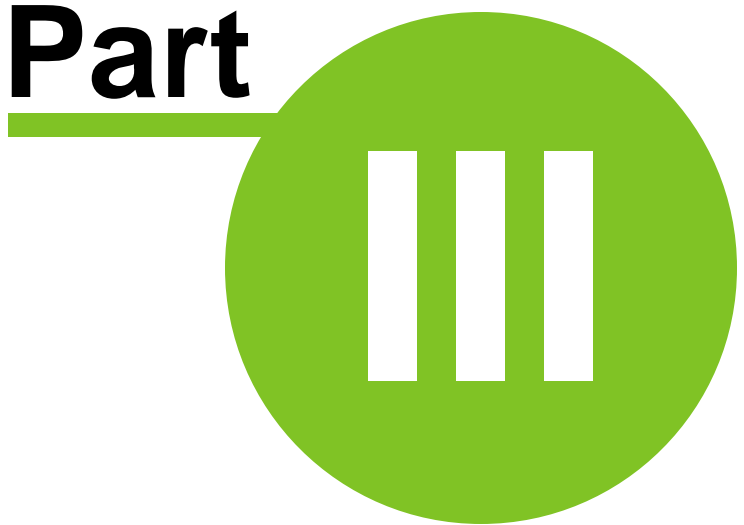
At first, in *InitEffect* the buffer *buf* is filled once, just in order to get the buffer length. With the buffer length, a horizontal distance *xStep* is calculated to separate some lines equally later on. Using the function *max*, the distance is at least 1.

Calling *RenderEffect* the buffer *buf* is filled with the current interval appearances. Then, a vertical line is drawn for every interval, either from the middle to the top of the matrix (if the interval was analyzed) or from middle to bottom (if not).

Using Other Tone Theoretical Parameters

There is lots of other data which may be used to create effects, e.g. the sound level or the note of the currently lowest note (bass tone). The handling is similar to the functions described above. Further details are given by the [List of Functions](#).

Part



3 MADRIX Script (Programming Language Overview)

3.1 Keyword Search

The following keywords are available in MADRIX Script:

- break (in 'switch' statements)
- break (in loops)
- case
- const
- continue
- default
- else
- false, FALSE
- for
- if
- persistent
- return
- switch
- true, TRUE
- while

3.2 List Of Functions (Alphabetical Order)

Overview

In addition to the specific functions of the MAS Script effect, Macros for effects, the Main Output Macro, and the Storage Place Macro ([all listed below](#)), the following table lists additional functions. A "+" symbol indicates, in which areas of MADRIX Script the functions can be used.

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Main Output Macro
float abs (float x)	Returns the absolute value of x.	+	+	+	+
void AddPixelTransposeEntry (int srcX, int srcY, int destX, int destY)	Adds one entry to the pixel transpose table and resizes the table if necessary. <i>srcX</i> and <i>srcY</i> are the x and y coordinates of the source. <i>destX</i> and <i>destY</i> are the destination coordinates. » Description	+	+	+	+

float arccos (float a) float arccosDeg (float a)	Returns the arc cosine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float arcsin (float a) float arcsinDeg (float a)	Returns the arc sine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float arctan (float a) float arctanDeg (float a)	Returns the arc tangent of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float ceil (float f)	Rounds up the given value to the next integer value. E. g. <code>ceil(2.00001) = 3.0</code>	+	+	+	+
void ChangeBrightness (color col)	Adds the values of the specified color to the current color of each pixel in the matrix. » Example	+	+	+	+
int CheckScriptEngineVersion (int major, int minor)	Checks the Script engine version in use and returns <i>1</i> if the version is equal or higher to the version specified with <i>major</i> and <i>minor</i> . Or else <i>0</i> is returned. The current Script Engine Version is 1.43. A useful function to check if the minimum requirements of your script are met. » Example	+	+	+	+
int CheckSoftwareVersion (int major, int minor, int subminor, int subsubminor)	Checks the MADRIX software version in use and returns <i>1</i> if the version is equal or higher to the version specified with <i>major</i> , <i>minor</i> , <i>subminor</i> , and <i>subsubminor</i> . Or else <i>0</i> is returned. The current MADRIX version is 2.14.8.0. You can check which version you are using by opening the Logfile in MADRIX (at the beginning of the file) or check the MADRIX.exe (perform a right-click->Properties->Version). A useful function to check if the minimum requirements of your script are met. » Example	+	+	+	+
void Clear () void Clear (color col)	Fills the whole matrix with the given color. The default color (no color parameter) is black.	+	+	+	+
void ClearAlpha (int alpha)	Sets the alpha value of each pixel in the matrix to <i>alpha</i> .	+	+	+	+
void ClearColor (color col)	Fills the whole matrix with the given color without changing the alpha value.	+	+	+	+
void ColorReplace (color oldCol, color newCol)	Replaces the given color <i>oldCol</i> with a new one (<i>newCol</i>).	+	+	+	+
float cos (float a) float cosDeg (float a)	Returns the cosine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float cosH (float a) float cosHDeg (float a)	Returns the hyperbolic cosine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
void CreatePixelTransposeTable (int size, int growsize)	Creates the pixel transpose table with the given <i>size</i> and <i>growsize</i> . » Description	+	+	+	+
float deg2rad (float a)	Converts the angle <i>a</i> from degrees to radian measure.	+	+	+	+
void Dim (float value)	Reduces the brightness of the complete virtual matrix. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+
void DimPixel (float value, int x, int y)	Reduces the brightness of an individual pixel. <i>x</i> and <i>y</i> are the coordinates of the pixel. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+

void DimPixelArea (float value, int x, int y, int width, int height)	Reduces the brightness of a certain area of the virtual matrix. <i>x</i> and <i>y</i> are the coordinates of the area (upper left corner). <i>width</i> and <i>height</i> specify the width and height of the area. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+
void DrawPixelArea (color matrix[[]], int xDst, int yDst, int w, int h, int xSrc, int ySrc, color filter)	Copies data from a 2-dimensional field of colors and renders it to the matrix. » Description	+	+	+	+
void DrawPixelCircle (color col, int x, int y, int rw, int rh)	Draws a circle at the center of a specified rectangle with the specified color. The rectangle is defined by the pixel coordinates (<i>x</i> , <i>y</i>) and the absolute width <i>rw</i> and height <i>rh</i> .	+	+	+	+
void DrawPixelCross (color col, int x, int y, int w, int h)	Draws a cross onto the matrix with the given width <i>w</i> and height <i>h</i> at the given position. Position and size are given as absolute pixel values.	+	+	+	+
void DrawPixelDiamond (color col, int x, int y, int w, int h)	Draws a diamond shape starting with pixel (<i>x</i> , <i>y</i>) with the absolute width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawPixelEllipse (color col, int mx, int my, int w, int h)	Draws an ellipse with the absolute midpoint position (<i>mx</i> , <i>my</i>) and the absolute width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawPixelLine (color col, int x1, int y1, int x2, int y2)	Draws a line from pixel (<i>x1</i> , <i>y1</i>) to (<i>x2</i> , <i>y2</i>) with the specified color. » Example	+	+	+	+
void DrawPixelRect (color col, int x, int y, int w, int h)	Draws a rectangle from pixel (<i>x</i> , <i>y</i>) with the absolute width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawPixelStar (color col, int x, int y, int w, int h)	Draws a star onto the matrix with the given width and height at the given position. Position and size are given as absolute pixel values.	+	+	+	+
void DrawPixelText (color c, font f, string t, int x, int y, int rotation)	Draws a text across the main output. <i>color c</i> and <i>font f</i> are structures. Valid values for <i>rotation</i> are ROTATION_TEXT_NONE , ROTATION_TEXT_90 , ROTATION_TEXT_180 , ROTATION_TEXT_270 . » Example 1 » Example 2 » Example 3 » Example 4	+	+	+	+
void DrawVectorCircle (color col, float x, float y, float rw, float rh)	Draws a circle at the center of a specified rectangle with the specified color. The rectangle is defined by the relative position (<i>x</i> , <i>y</i>) and the relative width <i>rw</i> and height <i>rh</i> .	+	+	+	+
void DrawVectorCross (color col, float x, float y, float w, float h)	Draws a cross onto the matrix with the given width and height at the given position. Both, position and size are given as relative values between 0.0 and 1.0.	+	+	+	+
void DrawVectorDiamond (color col, float x, float y, float w, float h)	Draws a diamond shape from the relative position (<i>x</i> , <i>y</i>) with the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawVectorEllipse (color col, float mx, float my, float w, float h)	Draws an ellipse with the relative midpoint position (<i>mx</i> , <i>my</i>) and the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawVectorLine (color col, float x1, float y1, float x2, float y2)	Draws a line from the relative position (<i>x1</i> , <i>y1</i>) to (<i>x2</i> , <i>y2</i>) with the specified color.	+	+	+	+

void DrawVectorRect (color col, float x, float y, float w, float h)	Draws a rectangle from the relative position (<i>x,y</i>) with the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void DrawVectorStar (color col, float x, float y, float w, float h)	Draws a star onto the matrix with the given width and height at the given position. Both, position and size are given as relative values between 0.0 and 1.0.	+	+	+	+
void DrawVectorText (color c, font f, string t, float x, float y, int rotation)	Draws a vector text across the main output. <i>color c</i> and <i>font f</i> are structures. Valid values for <i>rotation</i> are ROTATION_TEXT_NONE , ROTATION_TEXT_90 , ROTATION_TEXT_180 , ROTATION_TEXT_270 . Example: DrawPixelText(RED,f,"Hello",0,1,0);	+	+	+	+
void EnableFrameFade (int enable)	Enables or disables frame fade for the effect. If the value <i>enable</i> is set to <i>false</i> or <i>0</i> , frame fade will be disabled. Otherwise, it will be enabled.	+	+		
int endswith (string text, string substring)	This function checks if the string <i>text</i> ends with the given <i>substring</i> . If <i>text</i> ends with the specified <i>substring</i> , <i>true</i> is returned, otherwise <i>false</i> . » Description	+	+	+	+
void ExecutePixelTranspose (int clear)	Executes the pixel transposition by using the pixel transpose table. Using the value CLEAR will erase/overwrite all pixels that are not defined as destination with the the color black. Otherwise, NOCLEAR will keep all pixels that are not defined as original destination. » Description	+	+	+	+
float exp (float x)	Returns the result of e (2.7182...) to the power of <i>x</i> .	+	+	+	+
void FillPixelCircle (color col, int x, int y, int rw, int rh)	Fills a circle with the specified color at the center of the specified rectangle. The rectangle is defined by the starting pixel (<i>x,y</i>) and the absolute width <i>rw</i> and height <i>rh</i> .	+	+	+	+
void FillPixelDiamond (color col, int x, int y, int w, int h)	Fills a diamond starting from pixel (<i>x,y</i>) with the absolute width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void FillPixelEllipse (color col, int mx, int my, int w, int h)	Fills an ellipse with the absolute midpoint position (<i>mx, my</i>) and the absolute width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void FillPixelRect (color col, int x, int y, int w, int h)	Fills a rectangle starting from pixel (<i>x,y</i>) with the absolute width <i>w</i> and height <i>h</i> with the specified color. » Example	+	+	+	+
void FillVectorCircle (color col, float x, float y, float rw, float rh)	Fills a circle at the center of a specified rectangle in the specified color. The rectangle is defined by the relative position (<i>x,y</i>) and the relative width <i>rw</i> and height <i>rh</i> .	+	+	+	+
void FillVectorDiamond (color col, float x, float y, float w, float h)	Fills a diamond starting from the relative position (<i>x,y</i>) with the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void FillVectorEllipse (color col, float mx, float my, float w, float h)	Fills an ellipse with the relative midpoint position (<i>mx, my</i>) and the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void FillVectorRect (color col, float x, float y, float w, float h)	Fills a rectangle starting from the relative position (<i>x,y</i>) with the relative width <i>w</i> and height <i>h</i> with the specified color.	+	+	+	+
void Filter (int filter)	Renders a filter over the matrix. » Valid parameters (Filters) » Description	+	+	+	+

int findstring (int startIndex, string text, string substring)	Searches for the <i>substring</i> within <i>text</i> starting at <i>startIndex</i> . The function returns an index that describes the position at which the <i>substring</i> begins. » Description Example: substring(0, "Hallo Welt", "Welt") returns 6. If the <i>substring</i> is not found within <i>text</i> , -1 is returned.	+	+	+	+
float fmax (float x, float y)	Returns the maximum value of the floating point numbers <i>x</i> and <i>y</i> .	+	+	+	+
float fmin (float x, float y)	Returns the minimum value of <i>x</i> and <i>y</i> .	+	+	+	+
float fmod (float denominator, float divisor)	Calculates the remainder of the float division.	+	+	+	+
float frandom ()	Returns a random number within the the range of 0 to 1.	+	+	+	+
void GetAllIntervals (int buf[])	Fills the fiel <i>buf</i> with the occurrences of each interval (buf[0] ... buf[10]). <i>buf[index]</i> is <i>true</i> , if the specified interval was analyzed. » Example	+	+		
void GetAllNoteValues (int buf [])	Fills the field <i>buf</i> with the sound level values for each note (buf[0] ... buf[127]). <i>buf[index]</i> can differ from 0 to 127.	+	+		
string GetApplicationPath ()	Locates the MADRIX.exe on your harddisk and returns the path as a <i>string</i> . » Example	+	+	+	+
int GetBassTone ()	Returns a value, ranging from 0 to 127, representing the lowest tone (0 = C, 1 = C#, 2 = D, ...). The return value is -1, if the lowest tone could not be determined.	+	+		
int GetBassType ()	Returns a value, ranging from 0 to 11, representing the bass type (0 = base bass, 1,2 = small/large second bass, 3,4 = small/large third bass, ... , 10, 11 = small/large seventh bass). The return value is -1 if the bass type could not be determined.	+	+		
int GetBassValue ()	Returns a value, ranging from 0 to 127, representing the sound level of the bass tone.	+	+		
int GetColorDepth ()	Returns the color depth of the fixture.	+	+	+	+
string GetComputerName ()	Retrieves the name of the computer in use as defined in the Windows® operating system.	+	+	+	+
date GetDate ()	Returns a date structure with the current date. » Example	+	+	+	+
int GetDegreeMapRotation ()	Returns the current rotation value in degrees.	+	+		
void GetDmxIn (int DmxValues [], int startchannel, int channels, int universe)	Stores the DMX-IN status of several DMX channels in an array (<i>DmxValues[]</i>). To use the function, choose a DMX start channel and specify the number of channels that should be scanned. Valid values for <i>startchannel</i> range from 0 to 511. Valid values for <i>channels</i> range from 0 to 511. Please note that a maximum of 512 channels can be scanned. This means that the sum of <i>startchannel</i> + <i>channels</i> must be less or equal to 512. In addition, specify the universe that should be scanned. Valid values for <i>universe</i> range from 0 to 255. The default <i>universe</i> is 0, i.e. DMX Universe 1.	+	+	+	+

int GetDmxInChannel (int channel, int universe)	Returns the status of the DMX-IN functionality of a specified DMX channel (<i>channel</i>). Valid values for <i>channel</i> range from 0 to 511. In addition, specify the universe that should be scanned. Valid values for <i>universe</i> range from 0 to 255. The default value of <i>universe</i> is 0, i.e. DMX Universe 1.	+	+	+	+
int GetFilter ()	Returns which Filter Effect (FX) is applied to the matrix.	+	+	+	
float GetFrameCount ()	Retrieves the number of frames the effect produces before it gets repeated. This is the same value which was set by <i>SetFrameCount</i> . The initial value is 1000.0.	+	+		
float GetFrameId ()	Returns the ID of the current frame. »Description	+	+		
float GetFrameSteps ()	Returns the number of frames which are between this and the last call. »Description	+	+		
int GetLink ()	Returns 1 (i.e. true) if the option to link effect layers is enabled, otherwise 0 (false).	+	+		
int GetMapModeMirror ()	Returns the current mirror mode. »Description	+	+		
int GetMapModeTile ()	Returns the current tile mode. »Description	+	+		
int GetMapPixel (int map[])	Retrieves the current mapping of the effect. The coordinates and matrix dimension are returned in the given field using absolute pixel coordinates. The function returns <i>true</i> , if the effect matrix is mapped. <i>false</i> is returned if the effect is full-sized onto the matrix. »Description <ul style="list-style-type: none"> • map[0] = x coordinate • map[1] = y coordinate • map[2] = width • map[3] = height 	+	+		
int GetMapTileEffectPixel (int map[])	Retrieves the current map tile settings of the effect. Returns <i>true</i> if mapping is active, otherwise <i>false</i> . <i>map</i> retrieves the settings as follows: <ul style="list-style-type: none"> • map[0] = x - start coordinate • map[1] = y - start coordinate • map[2] = w - width of the tile • map[3] = h - height of the tile »Description	+	+		
int GetMapTileEffectVector (float map[])	Retrieves the current map tile settings of the effect. Returns <i>true</i> if mapping is active, otherwise <i>false</i> . <i>map</i> retrieves the settings as followed: <ul style="list-style-type: none"> • map[0] = x - start coordinate • map[1] = y - start coordinate • map[2] = w - width of the tile • map[3] = h - height of the tile »Description	+	+		
int GetMapVector (float map[])	Retrieves the current mapping of the effect. The coordinates and matrix dimension are returned in the given field using relative values between 0.0 and 1.0. But negative values are also possible. The function returns <i>true</i> , if the effect matrix is mapped. <i>false</i> is returned if the effect is full-sized onto the matrix. »Description <ul style="list-style-type: none"> • map[0] = x-coordiante • map[1] = y-coordiante • map[2] = width • map[3] = height 	+	+		

int GetMatrixHeight ()	Returns the vertical pixel count of the matrix.	+	+	+	+
int GetMatrixWidth ()	Returns the horizontal pixel count of the matrix.	+	+	+	+
void GetMidiInControl (int midivalues[], int startcontrol, int controlcount, int midichannel, int device)	Stores the MIDI values for multiple controls in an array (<i>midivalues[]</i>). To use the function, choose a start control (<i>startcontrol</i>) and the number of controls (<i>controlcount</i>), as well as the MIDI channel (<i>midichannel</i>) and the <i>device</i> . The default value of <i>startnote</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. The default and maximum value of <i>controlcount</i> is 128. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
int GetMidiInControlValue (int midicontrol, int midichannel, int device)	Returns the MIDI control value of the specified MIDI <i>control change</i> (<i>midicontrol</i>) and <i>midichannel</i> for the specified <i>device</i> . The default value of <i>control</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
void GetMidiInNote (int midivalues[], int startnote, int notecount, int midichannel, int device)	Stores the MIDI values for multiple notes in an array (<i>midivalues[]</i>). To use the function, choose a start note (<i>startnote</i>) and the number of notes (<i>notecount</i>), as well as the MIDI channel (<i>midichannel</i>) and the <i>device</i> . The default value of <i>startnote</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. The default and maximum value of <i>notecount</i> is 128. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
int GetMidiInNoteValue (int midinote, int midichannel, int device)	Returns the MIDI note value of the specified MIDI note (<i>midinote</i>) and <i>midichannel</i> for the specified <i>device</i> . The default value of <i>note</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first note, channel, and device. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
int GetMixMode ()	Returns the currently set mix mode. » Valid parameters (Mix modes) » Description	+	+		
int GetNoteValue (int note)	Returns a value, ranging from 0 to 127, representing the sound level of the specified note: 0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz. » Example	+	+		
int GetOpacity ()	Returns the currently set opacity value of the effect. The value ranges from 0 to 255.	+	+		
color GetPixel (int x, int y)	Returns the color of the pixel at position (x,y).	+	+	+	+
void GetPixelArea (color matrix[], int xSrc, int ySrc, int w, int h, int xDst, int yDst)	Retrieves data from the matrix and stores it into a 2-dimensional field of colors. » Description	+	+	+	+
string GetScriptEngineVersion ()	Returns the script engine version and returns the value as a <i>string</i> . » Example	+	+	+	+
string GetSoftwareVersion ()	Returns the MADRIX software version and returns the value as a <i>string</i> . » Example	+	+	+	+

int GetSoundLevel (int channel)	Returns a value, ranging from 0 to 255, representing the sound level of the stereo channel: 0 = left channel, 1 = right channel. » Example	+	+		
int GetSubMaster ()	Returns the currently set submaster value of the effect. The value ranges from 0 to 255.	+	+		
time GetTime ()	Returns a time structure with the current time. » Example	+	+	+	+
time GetTimeCode ()	Returns a time structure with the currently used Time Code. » Example	+	+	+	+
int GetTonality ()	Returns a value, ranging from 0 to 11, representing the tonality (0 = C, 1 = C#, 2 = D ...). The return value is -1, if the tonality could not be determined. » Example	+	+		
int GetToneScale ()	Returns a value, ranging from 0 to 2, representing the tone scale (0 = undetermined, 1 = major, 2 = minor). » Example	+	+		
string GetUserName ()	Retrieves the name of the current user as defined in the Windows® operating system.	+	+	+	+
string GetUserProfileDirectory ()	Locates the directory of the current user profile on the harddisk and returns the path as a <i>string</i> . » Example	+	+	+	+
float GetVectorMapRotation ()	Returns the current rotation value.	+	+		
color GetVectorPixel (float x, float y)	Returns the <i>color</i> of the pixel drawn at vector coordinates x and y.	+	+	+	+
void Greyscale ()	Converts the whole matrix to grey scaled values.	+	+	+	+
float hypot (float x, float y)	Calculates the length of the hypotenuse of an orthogonal triangle with x and y as the cathetus' lengths.	+	+	+	+
void InvertColor ()	Inverts the color of the entire matrix.	+	+	+	+
void InvertMatrix ()	Inverts the whole matrix. The top left pixel will be moved to the bottom right and the bottom left pixel to the top right.	+	+	+	+
int isalnum (string text)	Returns <i>true</i> if the given string <i>text</i> contains only characters and figures and if its length is greater than 0. Otherwise, <i>false</i> is returned. » Description	+	+	+	+
int isalpha (string text)	Returns <i>true</i> if the given string contains only characters and if its length is greater than 0. Otherwise, <i>false</i> is returned. » Description	+	+	+	+
int IsDmxInEnabled ()	Retrieves if DMX-IN is activated (1) or not (0).	+	+	+	+
int IsFrameFadeEnabled ()	Returns <i>true</i> if frame fade is currently enabled, or <i>false</i> if not.	+	+		
int IsInterval (int index)	Returns <i>true</i> if the specified interval (at the <i>index</i> position, ranging from 0 to 10) was analyzed.	+	+		
int IsMapped ()	Returns <i>true</i> if the current effect is mapped, otherwise <i>false</i> .	+	+		
int IsMapRotation ()	Returns the status of the map rotation. 0 means the rotation value is absolute (no animation) and 1 means the rotation value is relative (animation).	+	+		
int IsMidiInEnabled ()	Retrieves if MIDI-IN is activated (1) or not (0).	+	+	+	+

int IsNote (int note)	Returns <i>true</i> if the specified note was analyzed. (0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz)	+	+		
int isnum (string text)	Returns <i>true</i> if the given <i>text</i> represents a number. This may be an integer value or a floating point number, like 1.3. Otherwise it returns <i>false</i> . » Description	+	+	+	+
int IsTonality ()	Returns <i>true</i> if the tonality could be determined.	+	+		
float ln (float x)	Returns the result of the natural logarithm of <i>x</i> .	+	+	+	+
float log10 (float x)	Returns the result of the common logarithm of <i>x</i> .	+	+	+	+
void MapEffectPixel (int x, int y, int w, int h)	Sets the mapping coordinates for the current effect. Using this method the effect may be mapped to different positions of the matrix. <i>x</i> and <i>y</i> are the start coordinates using absolute pixel values. Negative values are also possible in order to move the effect matrix beyond the real matrix. <i>w</i> and <i>h</i> are the new width and height of the mapped matrix. » Description Please note: If the size of the virtual effect matrix is changed by this function, the effect will be restarted and <i>InitEffect</i> is called.	+	+		
void MapEffectVector (float x, float y, float w, float h)	Sets the mapping coordinates for the current effect. Using this method the effect may be mapped to different positions of the matrix. <i>x</i> and <i>y</i> are the start coordinates using relative values between 0.0 and 1.0. Negative values are also possible. <i>w</i> and <i>h</i> are the new width and height of the mapped matrix. » Description Please note: If the size of the virtual effect matrix is changed by this function, the effect will be restarted and <i>InitEffect</i> is called.	+	+		
void MapTileEffectPixel (int x, int y, int w, int h)	Sets the tile coordinates for the current effect using absolute values. <i>x</i> and <i>y</i> are the start coordinates of the tile. <i>w</i> and <i>h</i> are the width and the height.	+	+		
void MapTileEffectVector (float x, float y, float w, float h)	Sets the tile coordinates for the current effect using relative values. <i>x</i> and <i>y</i> are the start coordinates of the tile. <i>w</i> and <i>h</i> are the width and the height.	+	+		
int max (int x, int y)	Returns the maximum value of the integer numbers <i>x</i> and <i>y</i> .	+	+	+	+
int min (int x, int y)	Returns the minimum value of <i>x</i> and <i>y</i> .	+	+	+	+
void PixelFloodFill (color col, int x, int y)	Fills an single-colored area with the specified color <i>col</i> , starting at the given position <i>x</i> and <i>y</i> .	+	+	+	+
float pow (float x, float y)	Returns the result of the calculation of <i>x</i> to the power of <i>y</i> .	+	+	+	+
float rad2deg (float a)	Converts the angle <i>a</i> from radian measure to degrees.	+	+	+	+
int random (int min, int max)	Returns a random integer number in the value range of <i>min</i> to <i>max</i> .	+	+	+	+
int ReadAsync (string file, string txt)	Reads content as text from a <i>file</i> into the string <i>txt</i> . » Description & Examples	+	+	+	+
void replace (string src, string old, string new)	Replaces any appearances of <i>old</i> within <i>src</i> with <i>new</i> . » Description	+	+	+	+

int rfindstring (int startIndex, string text, string substring)	This function looks for the <i>substring</i> in the given <i>text</i> from its ending to the beginning. The search starts at the given <i>startIndex</i> , while the last character has the index 0. The function returns the index where the substring occurs for the first time after the specified <i>startIndex</i> . If the substring was not found, -1 is returned. » Description	+	+	+	+
float round (float f)	Rounds the given value to its next integer value. The value is rounded correctly, either up or down.	+	+	+	+
void SetDegreeMapRotation (int degree, int anim)	Sets the rotation value and animation status of the effect using absolute values. <i>degree</i> describes the degrees/angle. Valid values range from -720 to 720. Valid values for <i>anim</i> are MAP_ANIM_OFF and MAP_ANIM_ON.	+	+		
void SetFilter (int filter)	Applies a Filter Effect (FX) to the matrix. Valid values for <i>filter</i> are » Filters » Description	+	+	+	
void SetFrameId (float id)	Sets a new frame ID. If the given <i>id</i> is lower than 0, it is set to 0. » Description	+	+		
void SetInvalid ()	Use this function to set an already rendered frame/ effect as invalid. This means that the frame will be re-rendered, which is the normal case. This can be useful with dynamic effects since the content of the effect does change all the time. Re-rendering is the normal case as effects and output data are rendered with several frames per second (FPS). See also » SetValid	+	+		
void SetLink (int enable)	Enables or disables the option to link effect layers. Valid values range are 0 (false) or 1 (true).	+	+		
void SetMapModeMirror (mirrorMode)	Sets the mirror mode for effect tiling. For <i>mirrorMode</i> one of the MAP_MIRROR_ values must be used. » Description	+	+		
void SetMapModeTile (tileMode)	Sets the tile mode for effect tiling. For <i>tileMode</i> one of the MAP_TILE_ values must be used. » Description	+	+		
void SetMixMode (int mode)	Sets the mix mode of the effect. <i>mode</i> may be one of the values defined in Mix modes . If <i>mode</i> is invalid, nothing happens and a message is displayed in the Script output of the Script Editor. » Description	+	+		
void SetOpacity (int value)	Sets the opacity value of the effect. Valid values range from 0 to 255.	+	+		
void SetPixel (color col, int x, int y)	Sets the pixel at position (x,y) to the specified color. » Description & Examples	+	+	+	+
void SetPixelGreyscale (int x, int y)	Renders the selected pixel at position (x,y) in greyscale. » Description	+	+	+	+
void SetPixelTransposeEntry (int idx, int srcX, int srcY, int destX, int destY)	Adds one pixel transpose table entry defined by <i>idx</i> . <i>srcX</i> and <i>srcY</i> are the x and y coordinates of the source. <i>destX</i> and <i>destY</i> are the destination coordinates. » Description	+	+	+	+
int SetReadAsyncInterval (string file, int interval)	Sets the reading <i>interval</i> for a certain <i>file</i> . To be used in combination with ReadAsync . » Description & Examples	+	+	+	+
void SetSubMaster (int value)	Sets the submaster value of the effect. Valid values range from 0 to 255.	+	+		

void SetValid ()	Use this function to set an already rendered frame/ effect as valid. This means that the frame will not be re-rendered. This can be useful with static effects since the content of the effect does not change over time. Re-rendering would be the normal case as effects and output data are rendered with several frames per second (FPS). See also » SetInvalid	+	+		
void SetVectorMapRotation (float value, int anim)	Sets the rotation value and animation status of the effect using relative values. <i>degree</i> describes the degrees/angle. Valid values range from -200.0 to 200.0. Valid values for <i>anim</i> are MAP_ANIM_OFF und MAP_ANIM_ON.	+	+		
void SetVectorPixel (color, float x, float y)	Draws one pixel at the given variable vector coordinates. In addition to the coordinates, a <i>color</i> must be specified.	+	+	+	+
void ShiftPixelMatrix (int x, int y, int w, int h, int dir, int step)	Moves the area of the matrix (defined by <i>x</i> , <i>y</i> , <i>w</i> , and <i>h</i>) <i>step</i> pixel into the given direction. Exemplary values for <i>dir</i> are SHIFT_UP, SHIFT_DOWN, or SHIFT_UL. » List of Global Constants . Please note that after shifting the pixels, the data is lost. It is not possible to undo a shifting operation. <i>x</i> , <i>y</i> , <i>w</i> , <i>h</i> describe the rectangle that should be shifted. <i>x</i> and <i>y</i> describe the upper left corner. <i>w</i> and <i>h</i> are the width and height of the rectangle. The chapter about the function provides more details .	+	+	+	+
void ShiftVectorMatrix (float x, float y, float w, float h, int dir, float step)	Moves the area of the matrix (defined by <i>x</i> , <i>y</i> , <i>w</i> , and <i>h</i>) <i>step</i> pixel into the given direction. Exemplary values for <i>dir</i> are SHIFT_UP, SHIFT_DOWN, or SHIFT_UL. » List of Global Constants . Please note that after shifting the pixels, the data is lost. It is not possible to undo a shifting operation. <i>x</i> , <i>y</i> , <i>w</i> , <i>h</i> describe the rectangle that should be shifted. <i>x</i> and <i>y</i> describe the upper left corner. <i>w</i> and <i>h</i> are the width and height of the rectangle. The chapter about the function provides more details .	+	+	+	+
float sin (float a) float sinDeg (float a)	Returns the sine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float sinH (float a) float sinHDeg (float a)	Returns the hyperbolic sine of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float sqrt (float x)	Returns the square root of <i>x</i> .	+	+	+	+
int startswith (string text, string substring)	This function checks if the <i>text</i> string starts with the given <i>substring</i> . If <i>text</i> starts with <i>substring</i> , <i>true</i> is returned. Otherwise <i>false</i> is returned. » Description	+	+	+	+
int strcmp (string str1, string str2)	Compares the two given strings each other. If they are equal, <i>0</i> is returned. A value of <i>-1</i> is returned if <i>str1</i> is less than <i>str2</i> . A value of <i>1</i> is returned if <i>str1</i> is greater than <i>str2</i> . » Description	+	+	+	+
void strip (string text)	Removes leading and ending white spaces, such as space, tabulator, line feeds, etc., from the given string <i>text</i> . » Description	+	+	+	+
string substring (string text, int start, int count)	Extracts a certain number of characters (<i>count</i>) beginning with <i>start</i> from the given <i>text</i> . » Description Example: substring("Hello", 0, 2) returns "He".	+	+	+	+

float tan (float a) float tanDeg (float a)	Returns the tangent of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
float tanH (float a) float tanHDeg (float a)	Returns the hyperbolic tangent of the angle <i>a</i> in radian measure or degrees, respectively.	+	+	+	+
void tokenize (string src, string delimiter, string reslist[])	Strips the string <i>src</i> into smaller pieces delimited by characters within <i>delimiter</i> . The result is returned in <i>reslist</i> . » Description	+	+	+	+
void tolower (string text)	Converts each character of the given string <i>text</i> into a lowercase character. » Description	+	+	+	+
void toupper (string text)	Converts each character of the given string <i>text</i> into a uppercase characters. » Description	+	+	+	+
void TRACE (variable)	Writes the value of the specified variable into the Script output window. Fulfills the same function as WriteText .	+	+	+	+
float trunc (float f)	Truncates/cuts off the decimals from a floating point value.	+	+	+	+
void VectorFloodFill (color col, float x, float y)	Fills an single-colored area with the given color <i>col</i> , starting at the given position <i>x</i> and <i>y</i> . In contrast to <i>PixelFloodFill</i> , the coordinates are given as floating point values ranging from 0.0 to 1.0.	+	+	+	+
void WriteText (string s)	Writes the specified message into the Script output window. » Example	+	+	+	+

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void GetAllValues (int buf[])	Fills the field <i>buf</i> with the sound level values for each note (buf [0] ... buf[127]). <i>buf[index]</i> can differ from 0 to 127. Please note: This function is deprecated. Use GetAllNoteValues instead.
int GetValue (int note)	Returns a value, ranging from 0 to 127, representing the sound level of the specified note: 0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz. Please note: This function is deprecated. Use GetNoteValue instead.
int GetSubmasterValue ()	Returns the currently set submaster value of the effect. The value ranges from 0 to 255. Please note: This function is deprecated. Use GetSubMaster instead.
void SetSubmasterValue (int value)	Sets the submaster value of the effect. Valid values range from 0 to 255. Please note: This function is deprecated. Use SetSubMaster instead.

Specific Functions

Macro Functions Only Available For The MAS Script Effect

You can find a detailed description in the chapter [MAS Script Effect: Functions](#).

Here is an overview:

- **DoPreRender**
- **GetBpm**
- **GetFrameCount**
- **GetFrameId**
- **GetFrameSteps**
- **SetBpm**
- **SetFixedFrameRate**
- **SetFrameCount**
- **SetFrameId**
- **SetUseFloatFrames**

Functions Only Available For Macros For Effects

The standard functions of the MAS Script effect are also available for effect macros. Effects in MADRIX are grouped into three categories: SCE - Static Color effects, S2L - Sound2Light effects, and M2L - Music2Light effects.

For better readability, we have only listed the various effects. You can find more information in the chapter Macros for Effects: Functions or in the sub-chapters:

SCE - Static Color Effects S2L - Sound2Light Effects M2L - Music2Light Effects

SCE Color	S2L Equalizer	M2L Color Fade
SCE Bitmap	S2L EQ Drops	M2L Color Rings
SCE Bounce	S2L EQ Shapes	M2L Color Scroll
SCE Color Change	S2L EQ Tubes	M2L Interval Drops
SCE Color Fill	S2L Frequency Flash	M2L Interval Tubes
SCE Color Ramp	S2L Level Color	M2L Single Tone Shapes
SCE Color Scroll	S2L Level Meter	

SCE Drops	S2L Level Ring
SCE Explosions	S2L Waveform
SCE Fire	S2L Wavegraph
SCE Metaballs	
SCE Plasma	
SCE Pulse / Stroboscope	
SCE Radial	
SCE Shapes	
SCE Starfield	
SCE Ticker	
SCE Video	
SCE Wave	

Functions Only Available For The Main Output Macro

You can find a detailed description in the chapter [Main Output Macro: Functions](#).

Here is an overview:

- **CuelistBack**
- **CuelistCurrentCue**
- **CuelistGo**
- **CuelistGoto**
- **CuelistPlay**
- **CuelistStop**
- **Filter**
- **GetAudioFader**
- **GetFadeColor**
- **GetFadeTime**
- **GetFadeType**
- **GetFadeValue**
- **GetFilterColor**
- **GetFreeze**
- **GetMasterFader**
- **GetStorageFullState**
- **GetStoragePause**
- **GetStoragePlace**

- **GetStorageSpeedMaster**
- **GetStorageSubMaster**
- **SetAudioFader**
- **SetFadeColor**
- **SetFadeTime**
- **SetFadeType**
- **SetFadeValue**
- **SetFilterColor**
- **SetFreeze**
- **SetMasterFader**
- **SetStoragePause**
- **SetStoragePlace**
- **SetStorageSpeedMaster**
- **SetStorageSubMaster**
- **SetStorageFilter**
- **GetStorageFilter**
- **ImportStoragePlace**
- **ImportStorage**
- **ImportPatch**

Functions Only Available For The Storage Place Macro

You can find a detailed description in the chapter [Storage Place Macro: Functions](#).

Here is an overview:

- **EnableLayerFrameFade**
- **GetDescription**
- **GetLayerBlind**
- **GetLayerCount**
- **GetLayerDegreeMapRotation**
- **GetLayerFrameCount**
- **GetLayerFrameId**
- **GetLayerFrameSteps**
- **GetLayerLink**
- **GetLayerMapModeMirror**
- **GetLayerMapModeTile**
- **GetLayerMapPixel**

- **GetLayerMapTileEffectPixel**
- **GetLayerMapTileEffectVector**
- **GetLayerMapVector**
- **GetLayerMixMode**
- **GetLayerOpacity**
- **GetLayerPixelTileOffset**
- **GetLayerSolo**
- **GetLayerSubMaster**
- **GetLayerVectorMapRotation**
- **GetLayerVectorTileOffset**
- **GetPause**
- **GetSpeedMaster**
- **GetSubMaster**
- **IsLayerFrameFadeEnabled**
- **IsLayerMapped**
- **IsLayerMapRotation**
- **MapLayerEffectPixel**
- **MapLayerEffectVector**
- **MapLayerTileEffectPixel**
- **MapLayerTileEffectVector**
- **SetDescription**
- **SetLayerBlind**
- **SetLayerDegreeMapRotation**
- **SetLayerFrameId**
- **SetLayerLink**
- **SetLayerMapModeMirror**
- **SetLayerMapModeTile**
- **SetLayerMixMode**
- **SetLayerOpacity**
- **SetLayerPixelTileOffset**
- **SetLayerSolo**
- **SetLayerSubMaster**
- **SetLayerVectorMapRotation**
- **SetLayerVectorTileOffset**
- **SetPause**
- **SetSpeedMaster**
- **SetSubMaster**
- **SetFilter**

- **GetFilter**
- **SetLayerFilter**
- **GetLayerFilter**

3.3 List Of Functions (Grouped)

Overview

This chapter lists functions of the List Of Functions (Alphabetical Order) in groups of similar functions.

Further information is provided in the chapter [List Of Functions \(Alphabetical Order\)](#).

Function Header	Description	MAS Script	Macros For Effects	Storage Place Macro	Main Output Macro
Draw Functions					
void Clear () void Clear (color col)	Fills the whole matrix with the given color. The default color (no color parameter) is black.	+	+	+	+
void ClearAlpha (int alpha)	Sets the alpha channel of each pixel in the matrix to the specified <i>alpha</i> value.	+	+	+	+
void ClearColor (color col)	Fills the entire matrix with the given color without changing the alpha value.	+	+	+	+
void ChangeBrightness (color col)	Adds the values of the specified color to the current color of every pixel in the matrix.	+	+	+	+
void ColorReplace (color oldCol, color newCol)	Replaces the given color <i>oldCol</i> by another one.	+	+	+	+
void SetPixel (color col, int x, int y)	Sets the pixel at position (x,y) to the specified color. » Description & Examples	+	+	+	+
void SetPixelGreyscale (int x, int y)	Renders the selected pixel at position (x,y) in greyscale. » Description & Examples	+	+	+	+
color GetPixel (int x, int y)	Returns the color of the pixel at position (x,y).	+	+	+	+
void GetPixelArea (color matrix[][], int xSrc, int ySrc, int w, int h, int xDst, int yDst)	Retrieves data from the matrix and stores it into a 2-dimensional field of colors. » Description	+	+	+	+
void DrawPixelArea (color matrix[][], int xDst, int yDst, int w, int h, int xSrc, int ySrc, color filter)	Copies data from a 2-dimensional field of colors and renders it to the matrix. » Description	+	+	+	+
void DrawPixelLine (color col, int x1, int y1, int x2, int y2)	Draws a line from pixel (x1,y1) to (x2,y2) with the specified color.	+	+	+	+
void DrawVectorLine (color col, float x1, float y1, float x2, float y2)	Draws a line from the relative position (x1,y1) to (x2,y2) with the specified color.	+	+	+	+

void DrawPixelRect (color col, int x, int y, int w, int h) void FillPixelRect (color col, int x, int y, int w, int h)	Draws/Fills a rectangle starting from pixel (x,y) with the absolute width w and height h with the specified color.	+	+	+	+
void DrawVectorRect (color col, float x, float y, float w, float h) void FillVectorRect (color col, float x, float y, float w, float h)	Draws/Fills a rectangle from the relative position (x,y) with the relative width w and height h with the specified color.	+	+	+	+
void DrawPixelCircle (color col, int x, int y, int rw, int rh) void FillPixelCircle (color col, int x, int y, int rw, int rh)	Draws/Fills a circle in the center of a specified rectangle with the specified color. The rectangle is defined by the pixel (x,y) and the absolute width rw and height rh.	+	+	+	+
void DrawVectorCircle (color col, float x, float y, float rw, float rh) void FillVectorCircle (color col, float x, float y, float rw, float rh)	Draws/Fills a circle in the center of a specified rectangle with the specified color. The rectangle is defined by the relative position (x,y) and the relative width rw and height rh.	+	+	+	+
void DrawPixelDiamond (color col, int x, int y, int w, int h) void FillPixelDiamond (color col, int x, int y, int w, int h)	Draws/Fills a diamond starting from pixel (x,y) with the absolute width w and height h with the specified color.	+	+	+	+
void DrawVectorDiamond (color col, float x, float y, float w, float h) void FillVectorDiamond (color col, float x, float y, float w, float h)	Draws/Fills a diamond starting from the relative position (x,y) with the relative width w and height h with the specified color.	+	+	+	+
void DrawPixelEllipse (color col, int mx, int my, int w, int h) void FillPixelEllipse (color col, int mx, int my, int w, int h)	Draws/Fills an ellipse at the absolute midpoint position (mx,my) with the absolute width w, absolute height h, and the specified color.	+	+	+	+
void DrawVectorEllipse (color col, float mx, float my, float w, float h) void FillVectorEllipse (color col, float mx, float my, float w, float h)	Draws/Fills an ellipse at the relative midpoint position (mx,my) with the relative width w, relative height h, and the specified color.	+	+	+	+
void DrawPixelStar (color col, int x, int y, int w, int h)	Draws a star onto the matrix with the given width and height at the given position. Position and size are given as absolute pixel values.	+	+	+	+
void DrawVectorStar (color col, float x, float y, float w, float h)	Draws a star onto the matrix with the given width and height at the given position. Both, position and size are given as relative values between 0.0 and 1.0.	+	+	+	+
void DrawPixelCross (color col, int x, int y, int w, int h)	Draws a cross onto the matrix with the given width and height at the given position. Position and size are given as absolute pixel values.	+	+	+	+
void DrawVectorCross (color col, float x, float y, float w, float h)	Draws a cross onto the matrix with the given width and height at the given position. Both, position and size are given as relative values between 0.0 and 1.0.	+	+	+	+

void DrawPixelText (color c, font f, string t, int x, int y, int rotation)	Draws a text across the main output. <i>color c</i> and <i>font f</i> are structures. Valid values for <i>rotation</i> are ROTATION_TEXT_NONE , ROTATION_TEXT_90 , ROTATION_TEXT_180 , ROTATION_TEXT_270 . » Example 1 » Example 2 » Example 3 » Example 4	+	+	+	+
void DrawVectorText (color c, font f, string t, float x, float y, int rotation)	Draws a vector text across the main output. <i>color c</i> and <i>font f</i> are structures. Valid values for <i>rotation</i> are ROTATION_TEXT_NONE , ROTATION_TEXT_90 , ROTATION_TEXT_180 , ROTATION_TEXT_270 . Example: <code>DrawPixelText(RED,f,"Hello",0,1,0);</code>	+	+	+	+
void ShiftPixelMatrix (int x, int y, int w, int h, int dir, int step)	Moves the area of the matrix (defined by <i>x</i> , <i>y</i> , <i>w</i> , and <i>h</i>) <i>step</i> pixel into the given direction. Exemplary values for <i>dir</i> are SHIFT_UP , SHIFT_DOWN , or SHIFT_UL . » List of Global Constants . Please note that after shifting the pixels, the data is lost. It is not possible to undo a shifting operation. <i>x</i> , <i>y</i> , <i>w</i> , <i>h</i> describe the rectangle that should be shifted. <i>x</i> and <i>y</i> describe the upper left corner. <i>w</i> and <i>h</i> are the width and height of the rectangle. The chapter about the function provides more details .	+	+	+	+
void ShiftVectorMatrix (float x, float y, float w, float h, int dir, float step)	Moves the area of the matrix (defined by <i>x</i> , <i>y</i> , <i>w</i> , and <i>h</i>) <i>step</i> pixel into the given direction. Exemplary values for <i>dir</i> are SHIFT_UP , SHIFT_DOWN , or SHIFT_UL . » List of Global Constants . Please note that after shifting the pixels, the data is lost. It is not possible to undo a shifting operation. <i>x</i> , <i>y</i> , <i>w</i> , <i>h</i> describe the rectangle that should be shifted. <i>x</i> and <i>y</i> describe the upper left corner. <i>w</i> and <i>h</i> are the width and height of the rectangle. The chapter about the function provides more details .	+	+	+	+
void VectorFloodFill (color col, float x, float y)	Fills an single-colored area with the given color <i>col</i> , starting at the given position <i>x</i> and <i>y</i> . In contrast to <i>PixelFloodFill</i> , the coordinates are given as floating point values ranging from 0.0 to 1.0.	+	+	+	+
void PixelFloodFill (color col, int x, int y)	Fills an single-colored area with the specified color <i>col</i> , starting at the given position <i>x</i> and <i>y</i> .	+	+	+	+
void InvertColor ()	Inverts the color of the entire matrix.	+	+	+	+
void InvertMatrix ()	Inverts the whole matrix. The top left pixel will be moved to the bottom right and the bottom left pixel to the top right.	+	+	+	+
void Greyscale ()	Converts the whole matrix to grey scaled values.	+	+	+	+
void SetVectorPixel (color, float x, float y)	Draws one pixel at the given variable vector coordinates. In addition to the coordinates, a <i>color</i> must be specified.	+	+	+	+
color GetVectorPixel (float x, float y)	Returns the <i>color</i> of the pixel drawn at vector coordinates <i>x</i> and <i>y</i> .	+	+	+	+
void Filter (int filter)	Renders a filter over the matrix. » Valid parameters (Filters) » Description	+	+	+	+
void SetFilter (int filter)	Applies a Filter Effect (FX) to the matrix. Valid values for <i>filter</i> are » Filters » Description	+	+	+	
int GetFilter ()	Returns which Filter Effect (FX) is applied to the matrix.	+	+	+	

S2L Functions					
int GetSoundLevel (int channel)	Returns a value, ranging from 0 to 255, representing the sound level of the stereo channel: 0 = left channel, 1 = right channel.	+	+		
M2L Functions					
int GetNoteValue (int note)	Returns a value, ranging from 0 to 127, representing the sound level of the specified note: 0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz.	+	+		
int GetBassValue ()	Returns a value from 0 to 127 representing the sound level of the bass tone.	+	+		
void GetAllNoteValues (int buf[])	Fills the field <i>buf</i> with the sound level values for each note (buf[0] ... buf[127]). <i>buf[index]</i> can differ from 0 to 127.	+	+		
void GetAllIntervals (int buf[])	Fills the field <i>buf</i> with the occurrences of each interval (buf[0] ... buf[10]). <i>buf[index]</i> is <i>true</i> , if the specified interval was analyzed.	+	+		
int IsInterval (int index)	Returns <i>true</i> if the specified interval (at the <i>index</i> position, ranging from 0 to 10) was analyzed.	+	+		
int IsNote (int note)	Returns <i>true</i> if the specified note was analyzed. (0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz)	+	+		
int GetTonality ()	Returns a value, ranging from 0 to 11, representing the tonality (0 = C, 1 = C#, 2 = D ...). The return value is -1 if the tonality could not be determined.	+	+		
int GetBassTone ()	Returns a value, ranging from 0 to 127, representing the lowest tone (0 = C, 1 = C#, 2 = D, ...). The return value is -1 if the lowest tone could not be determined.	+	+		
int GetBassType ()	Returns a value, ranging from 0 to 11, representing the bass type (0 = base bass, 1,2 = small/large second bass, 3,4 = small/large third bass, ... , 10, 11 = small/large seventh bass). The return value is -1 if the bass type could not be determined.	+	+		
int GetToneScale ()	Returns a value, ranging from 0 to 2, representing the tone scale (0 = undetermined, 1 = major, 2 = minor).	+	+		
int IsTonality ()	Returns <i>true</i> if the tonality could be determined.	+	+		
Mathematical Functions					
int random (int min, int max)	Returns a random integer number in the value range of <i>min</i> to <i>max</i> .	+	+	+	+
float frandom ()	Returns a random float number in the value range of 0.0 to 1.0.	+	+	+	+
int min (int x, int y) float fmin (float x, float y)	Returns the minimum value of x and y.	+	+	+	+
int max (int x, int y) float fmax (float x, float y)	Returns the maximum value of x and y.	+	+	+	+
float abs (float x)	Returns the absolute value of x.	+	+	+	+
float sqrt (float x)	Returns the square root of x.	+	+	+	+
float pow (float x, float y)	Returns the result of the calculation of x to the power of y.	+	+	+	+

float exp (float x)	Returns the result of e (2.7182...) to the power of x.	+	+	+	+
float ln (float x)	Returns the result of the natural logarithm of x.	+	+	+	+
float log10 (float x)	Returns the result of the common logarithm of x.	+	+	+	+
float hypot (float x, float y)	Calculates the length of the hypotenuse of an orthogonal triangle with x and y as the cathetus' lengths.	+	+	+	+
float rad2deg (float a)	Converts the angle a from radian measure to degrees.	+	+	+	+
float deg2rad (float a)	Converts the angle a from degrees to radian measure.	+	+	+	+
float sin (float a) float sinDeg (float a)	Returns the sine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float arcsin (float a) float arcsinDeg (float a)	Returns the arc sine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float sinH (float a) float sinHDeg (float a)	Returns the hyperbolic sine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float cos (float a) float cosDeg (float a)	Returns the cosine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float arccos (float a) float arccosDeg (float a)	Returns the arc cosine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float cosH (float a) float cosHDeg (float a)	Returns the hyperbolic cosine of the angle a in radian measure or degrees, respectively.	+	+	+	+
float tan (float a) float tanDeg (float a)	Returns the tangent of the angle a in radian measure or degrees, respectively.	+	+	+	+
float arctan (float a) float arctanDeg (float a)	Returns the arc tangent of the angle a in radian measure or degrees, respectively.	+	+	+	+
float tanH (float a) float tanHDeg (float a)	Returns the hyperbolic tangent of the angle a in radian measure or degrees, respectively.	+	+	+	+
float round (float f)	Rounds the given value to its next integer value. The value is rounded correctly, either up or down.	+	+	+	+
float trunc (float f)	Truncates/cuts off the decimals from a floating point value.	+	+	+	+
float ceil (float f)	Rounds up the given value to the next integer value. E. g. <code>ceil(2.00001) = 3.0</code>	+	+	+	+
float fmod (float denominator, float divisor)	Calculates the remainder of the float division.	+	+	+	+
Mapping Functions					
int IsMapped ()	Returns <i>true</i> if the current effect is mapped, otherwise <i>false</i> .	+	+		
int GetMapPixel (int map[])	Retrieves the current mapping of the effect. The coordinates and matrix dimension are returned in the given field using absolute pixel coordinates. The function returns <i>true</i> if the effect matrix is mapped. <i>false</i> is returned if the effect is full-sized onto the matrix. <ul style="list-style-type: none"> map[0] = x coordinate map[1] = y coordinate map[2] = width map[3] = height 	+	+		

int GetMapVector (float map[])	Retrieves the current mapping of the effect. The coordinates and matrix dimension are returned in the given field using relative values between 0.0 and 1.0. But negative values are also possible. The function returns <i>true</i> if the effect matrix is mapped. <i>false</i> is returned if the effect is full-sized onto the matrix. <ul style="list-style-type: none"> • map[0] = x-coordiante • map[1] = y-coordiante • map[2] = width • map[3] = height 	+	+		
void MapEffectPixel (int x, int y, int w, int h)	Sets the mapping coordinates for the current effect. Using this method the effect may be mapped to different positions of the matrix. x and y are the start coordinates using absolute pixel values. Negative values are also possible in order to move the effect matrix beyond the real matrix. w and h are the new width and height of the mapped matrix. Please note: If the size of the virtual effect matrix is changed by this function, the effect will be restarted and <i>InitEffect</i> is called.	+	+		
void MapEffectVector (float x, float y, float w, float h)	Sets the mapping coordinates for the current effect. Using this method the effect may be mapped to different positions of the matrix. x and y are the start coordinates using relative values between 0.0 and 1.0. Negative values are also possible. w and h are the new width and height of the mapped matrix. Please note: If the size of the virtual effect matrix is changed by this function, the effect will be restarted and <i>InitEffect</i> is called.	+	+		
int GetMapTileEffectPixel (int map[])	Retrieves the current map tile settings of the effect. Returns <i>true</i> if mapping is active, otherwise <i>false</i> . <i>map</i> retrieves the settings as follows: <ul style="list-style-type: none"> • map[0] = x - start coordinate • map[1] = y - start coordinate • map[2] = w - width of the tile • map[3] = h - height of the tile 	+	+		
int GetMapTileEffectVector (float map[])	Retrieves the current map tile settings of the effect. Returns <i>true</i> if mapping is active, otherwise <i>false</i> . <i>map</i> retrieves the settings as followed: <ul style="list-style-type: none"> • map[0] = x - start coordinate • map[1] = y - start coordinate • map[2] = w - width of the tile • map[3] = h - height of the tile 	+	+		
void MapTileEffectPixel (int x, int y, int w, int h)	Sets the tile coordinates for the current effect using absolute values. x and y are the start coordinates of the tile. w and h are the width and the height.	+	+		
void MapTileEffectVector (float x, float y, float w, float h)	Sets the tile coordinates for the current effect using relative values. x and y are the start coordinates of the tile. w and h are the width and the height.	+	+		
int GetMapModeMirror ()	Returns the current mirror mode.	+	+		
int GetMapModeTile ()	Returns the current tile mode.	+	+		
void SetMapModeMirror (mirrorMode)	Sets the mirror mode for effect tiling. For <i>mirrorMode</i> one of the MAP_MIRROR_ values must be used.	+	+		
void SetMapModeTile (tileMode)	Sets the tile mode for effect tiling. For <i>tileMode</i> one of the MAP_TILE_ values must be used.	+	+		

void SetVectorMapRotation (float value, int anim)	Sets the absolute rotation value if <i>anim</i> is set to 0. Or the function sets the relative rotation value (meaning rotation animation) if <i>anim</i> is set to 1.	+	+		
float GetVectorMapRotation ()	Returns the current rotation value.	+	+		
void SetDegreeMapRotation (int degree, int anim)	Sets the absolute rotation value (int <i>degrees</i>) if <i>anim</i> is set to 0. Or the function sets the relative rotation value (int <i>degrees</i> ; meaning a rotation animation) if <i>anim</i> is set to 1.	+	+		
int GetDegreeMapRotation ()	Returns the current rotation value in degrees.	+	+		
int IsMapRotation ()	Returns the status of the map rotation. 0 means the rotation value is absolute (no animation) and 1 means the rotation value is relative (animation).	+	+		
String Functions					
int findstring (int startIndex, string text, string substring)	This functions looks for the <i>substring</i> in the given text. The search starts at the given <i>startIndex</i> . The first character has an index of 0. The function starts its search at a specified position of the entire <i>text</i> using <i>startIndex</i> and returns an index that describes the position at which the <i>substring</i> begins. If the <i>substring</i> is not found, -1 is returned.	+	+	+	+
string substring (string text, int startIndex, int count)	The function extracts <i>count</i> characters from the given <i>text</i> starting with <i>startIndex</i> . If <i>count</i> is -1, all characters of the string starting at <i>startIndex</i> are returned. » Example	+	+	+	+
int rfindstring (int startIndex, string text, string substring)	This functions looks for the <i>substring</i> in the given text from its end to the beginning. The function starts its search at a specified position of the entire <i>text</i> using <i>startIndex</i> and returns an index that describes the position at which the <i>substring</i> begins. If the substring was not found, -1 is returned.	+	+	+	+
int startswith (string text, string substring)	This function checks if the string <i>text</i> starts with the given <i>substring</i> . If <i>text</i> starts with <i>substring</i> , <i>true</i> is returned, otherwise <i>false</i> .	+	+	+	+
int endswith (string text, string substring)	This function checks if the string <i>text</i> ends with the given <i>substring</i> . If <i>text</i> ends with <i>substring</i> , <i>true</i> is returned, otherwise <i>false</i> .	+	+	+	+
int isalnum (string text)	Returns <i>true</i> if the given string contains only characters and figures and if its length is greater then 0. Otherwise, <i>false</i> is returned.	+	+	+	+
int isalpha (string text)	Returns <i>true</i> if the given string contains only characters and if its length is greater than 0, otherwise <i>false</i> is returned.	+	+	+	+
int isnum (string text)	Returns <i>true</i> if the given <i>text</i> represents a number. This may be an integer number or a floating point number (e.g. 1.3). Otherwise, it returns <i>false</i> .	+	+	+	+
void tolower (string text)	Converts each character of the given <i>string</i> into a lower-case character.	+	+	+	+
void toupper (string text)	Converts each character of the given <i>string</i> into an upper-case character.	+	+	+	+
void strip (string text)	Removes leading and ending white spaces, like space, tabulator, line feeds, etc. from the given <i>string</i> .	+	+	+	+

int strcmp (string str1, string str2)	Compares two given strings with each other. If they are equal, 0 is returned. -1 is returned if str1 is less than str2. A value of 1 is returned if str1 is bigger than str2.	+	+	+	+
void replace (string src, string old, string new)	Replaces any appearances of old within src with new.	+	+	+	+
void tokenize (string src, string delimiter, string reslist[])	Strips the src string into smaller pieces delimited by characters within delimiter. The result is returned in reslist. »Description	+	+	+	+
Effect-Controlling Functions					
void SetFrameId (float id)	Sets a new frame ID. If the given id is lower than 0, it is set to 0.	+	+		
float GetFrameId ()	Returns the ID of the current frame. »Description	+	+		
float GetFrameSteps ()	Returns the number of frames between this and the last call. »Description	+	+		
float GetFrameCount ()	Retrieves the number of frames the effect produces before it gets repeated. This is the same value which was set by SetFrameCount. The initial value is 1000.0.	+	+		
Other Functions					
void TRACE (variable)	Writes the value of the specified variable into the Script output window. Fulfills the same function as WriteText.	+	+	+	+
void WriteText (string s)	Writes the specified message into the Script output window.	+	+	+	+
int GetMatrixWidth ()	Returns the horizontal pixel count of the matrix.	+	+	+	+
int GetMatrixHeight ()	Returns the vertical pixel count of the matrix.	+	+	+	+
int GetColorDepth ()	Returns the color depth of the fixture.	+	+	+	+
date GetDate ()	Returns a date structure with the current date.	+	+	+	+
time GetTime ()	Returns a time structure with the current time.	+	+	+	+
int GetMixMode ()	Returns the currently set mix mode. »Valid parameters (Mix modes)	+	+		
void SetMixMode (int mode)	Sets the mix mode of the effect. mode may be one of the values defined in Mix modes. If mode is invalid, nothing happens and a message is displayed in the Script output of the Script Editor.	+	+		
int IsFrameFadeEnabled ()	Returns true if frame fade is currently enabled, or false if not.	+	+		
void EnableFrameFade (int enable)	Enables or disables frame fade for the effect. If the value enable is set to false or 0, frame fade will be disabled. Otherwise, it will be enabled.	+	+		
int GetSubMaster ()	Returns the currently set submaster value of the effect. The value ranges from 0 to 255.	+	+		
void SetSubMaster (int value)	Sets the submaster value of the effect. Valid values range from 0 to 255.	+	+		
int GetOpacity ()	Returns the currently set opacity value of the effect. The value ranges from 0 to 255.	+	+		

void SetOpacity (int value)	Sets the opacity value of the effect. Valid values range from 0 to 255.	+	+		
void SetLink (int enable)	Enables or disables the option to link effect layers. Valid values range are 0 (false) or 1 (true).	+	+		
int GetLink ()	Returns 1 (i.e. true) if the option to link effect layers is enabled, otherwise 0 (false).	+	+		
int ReadAsync (string file, string txt)	Reads content as text from a <i>file</i> into the string <i>txt</i> . » Description & Examples	+	+	+	+
int SetReadAsyncInterval (string file, int interval)	Sets the reading <i>interval</i> for a certain <i>file</i> . To be used in combination with ReadAsync . » Description & Examples	+	+	+	+
string GetApplicationPath ()	Locates the MADRIX.exe on your harddisk and returns the path as a <i>string</i> . » Example	+	+	+	+
string GetUserProfileDirectory ()	Locates the directory of the current user profile on the harddisk and returns the path as a <i>string</i> . » Example	+	+	+	+
void GetComputerName ()	Retrieves the name of the computer in use as defined in Windows.	+	+	+	+
void GetUserName ()	Retrieves the name of the current user as defined in Windows.	+	+	+	+
void SetInvalid ()	Use this function to set an already rendered frame/ effect as invalid. This means that the frame will be re-rendered, which is the normal case. This can be useful with dynamic effects since the content of the effect does change all the time. Re-rendering is the normal case as effects and output data are rendered with several frames per second (FPS). See also » SetValid	+	+		
void SetValid ()	Use this function to set an already rendered frame/ effect as valid. This means that the frame will not be re-rendered. This can be useful with static effects since the content of the effect does not change over time. Re-rendering would be the normal case as effects and output data are rendered with several frames per second (FPS). See also » SetInvalid	+	+		
time GetTimeCode ()	Returns a time structure with the currently used Time Code. » Example	+	+	+	+
Version Number Functions					
string GetScriptEngineVersion ()	Returns the script engine version and returns the value as a <i>string</i> . » Example	+	+	+	+
string GetSoftwareVersion ()	Returns the MADRIX software version and returns the value as a <i>string</i> . » Example	+	+	+	+
int CheckScriptEngineVersion (int major, int minor)	Checks the Script engine version in use and returns 1 if the version is equal or higher to the version specified with <i>major</i> and <i>minor</i> . Or else 0 is returned. The current Script Engine Version is 1.43. A useful function to check if the minimum requirements of your script are met. » Example	+	+	+	+

int CheckSoftwareVersion (int major, int minor, int subminor, int subsubminor)	Checks the MADRIX software version in use and returns 1 if the version is equal or higher to the version specified with <i>major</i> , <i>minor</i> , <i>subminor</i> , and <i>subsubminor</i> . Or else 0 is returned. The current MADRIX version is 2.14.8.0. You can check which version you are using by opening the Logfile in MADRIX (at the beginning of the file) or check the MADRIX.exe (perform a right-click->Properties->Version). A useful function to check if the minimum requirements of your script are met. » Example	+	+	+	+
DMX-IN Functions					
void GetDmxIn (int DmxValues [], int startchannel, int channels, int universe)	Stores the DMX-IN status of several DMX channels in an array (<i>DmxValues[]</i>). To use the function, choose a DMX start channel and specify the number of channels that should be scanned. Valid values for <i>startchannel</i> range from 0 to 511. Valid values for <i>channels</i> range from 0 to 511. Please note that a maximum of 512 channels can be scanned. This means that the sum of <i>startchannel</i> + <i>channels</i> must be less or equal to 512. In addition, specify the universe that should be scanned. Valid values for <i>universe</i> range from 0 to 255. The default <i>universe</i> is 0, i.e. DMX Universe 1.	+	+	+	+
int GetDmxInChannel (int channel, int universe)	Returns the status of the DMX-IN functionality of a specified DMX channel (<i>channel</i>). Valid values for <i>channel</i> range from 0 to 511. In addition, specify the universe that should be scanned. Valid values for <i>universe</i> range from 0 to 255. The default value of <i>universe</i> is 0, i.e. DMX Universe 1.	+	+	+	+
int IsDmxInEnabled ()	Retrieves if DMX-IN is activated (1) or not (0).	+	+	+	+
PixelTranspose Functions					
void CreatePixelTransposeTable (int size, int growsize)	Creates the pixel transpose table with the given <i>size</i> and <i>growsize</i> .	+	+	+	+
void SetPixelTransposeEntry (int idx, int srcX, int srcY, int destX, int destY)	Adds one pixel transpose table entry defined by <i>idx</i> . <i>srcX</i> and <i>srcY</i> are the x and y coordinates of the source. <i>destX</i> and <i>destY</i> are the destination coordinates.	+	+	+	+
void AddPixelTransposeEntry (int srcX, int srcY, int destX, int destY)	Adds one entry to the pixel transpose table and resizes the table if necessary. <i>srcX</i> and <i>srcY</i> are the x and y coordinates of the source. <i>destX</i> and <i>destY</i> are the destination coordinates.	+	+	+	+
void ExecutePixelTranspose (int clear)	Executes the pixel transposition by using the pixel transpose table. Using the value CLEAR will erase/overwrite all pixels that are not defined as destination with the color black. Otherwise, NOCLEAR will keep all pixels that are not defined as original destination.	+	+	+	+
MIDI-IN Functions					
int IsMidiInEnabled ()	Retrieves if MIDI-IN is activated (1) or not (0).	+	+	+	+

int GetMidiInNoteValue (int midinote, int midichannel, int device)	Returns the MIDI note value of the specified MIDI note (<i>midinote</i>) and <i>midichannel</i> for the specified <i>device</i> . The default value of <i>note</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first note, channel, and device. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
int GetMidiInControlValue (int midicontrol, int midichannel, int device)	Returns the MIDI control value of the specified MIDI <i>control change</i> (<i>midicontrol</i>) and <i>midichannel</i> for the specified <i>device</i> . The default value of <i>control</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
void GetMidiInNote (int midivalues[], int startnote, int notecount, int midichannel, int device)	Stores the MIDI values for multiple notes in an array (<i>midivalues[]</i>). To use the function, choose a start note (<i>startnote</i>) and the number of notes (<i>notecount</i>), as well as the MIDI channel (<i>midichannel</i>) and the <i>device</i> . The default value of <i>startnote</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. The default and maximum value of <i>notecount</i> is 128. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
void GetMidiInControl (int midivalues[], int startcontrol, int controlcount, int midichannel, int device)	Stores the MIDI values for multiple controls in an array (<i>midivalues[]</i>). To use the function, choose a start control (<i>startcontrol</i>) and the number of controls (<i>controlcount</i>), as well as the MIDI channel (<i>midichannel</i>) and the <i>device</i> . The default value of <i>startnote</i> , <i>midichannel</i> , and <i>device</i> is 0, i.e. the first control, channel, and device. The default and maximum value of <i>controlcount</i> is 128. (Please note: In MADRIX, the index for channels and devices starts with 1, while the index for MIDI notes starts with 0). » Example	+	+	+	+
Dimming Functions					
void Dim (float value)	Reduces the brightness of the complete virtual matrix. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+
void DimPixel (float value, int x, int y)	Reduces the brightness of an individual pixel. <i>x</i> and <i>y</i> are the coordinates of the pixel. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+
void DimPixelArea (float value, int x, int y, int width, int height)	Reduces the brightness of a certain area of the virtual matrix. <i>x</i> and <i>y</i> are the coordinates of the area (upper left corner). <i>width</i> and <i>height</i> specify the width and height of the area. Valid values for <i>value</i> range from 0.0 to 1.0.	+	+	+	+







Deprecated Functions

















Deprecated functions are outdated functions and should not be used anymore.





Function	Description
----------	-------------

void GetAllValues (int buf[])	Fills the field <i>buf</i> with the sound level values for each note (buf [0] ... buf[127]). <i>buf[index]</i> can differ from 0 to 127. Please note: This function is deprecated. Use GetAllNoteValues instead.
int GetValue (int note)	Returns a value, ranging from 0 to 127, representing the sound level of the specified note: 0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz. Please note: This function is deprecated. Use GetNoteValue instead.
int GetSubmasterValue ()	Returns the currently set submaster value of the effect. The value ranges from 0 to 255. Please note: This function is deprecated. Use GetSubMaster instead.
void SetSubmasterValue (int value)	Sets the submaster value of the effect. Valid values range from 0 to 255. Please note: This function is deprecated. Use SetSubMaster instead.

3.4 List Of Global Variables And Constants

Variable / Define	Description
Math	
float PI	Represents the number PI with a value of 3.141592.
S2L - Sound2Light	
int SOUND_DATA_LEFT[]	Contains the sound values of the left audio channel.
int SOUND_DATA_RIGHT[]	Contains the sound values of the right audio channel.
int MAX_FREQUENCY_VOLUME	Describes the maximum value of an entry in either SOUND_DATA_LEFT or SOUND_DATA_RIGHT.
Drawing	
color WHITE	 White color without alpha. {255,255,255,0,0}
color BLACK	 Black color without alpha. {0,0,0,0,0}
color RED	 Red color without alpha. {255,0,0,0,0}
color MAROON	 Maroon color without alpha. {128,0,0,0,0}
color GREEN	 Green color without alpha. {0,255,0,0,0}
color MADRIX_GREEN	 MADRIX green without alpha. {177,219,24,0,0}

Variable / Define	Description
color BLUE	 Blue color without alpha. {0,0,255,0,0}
color NAVY	 Navy color without alpha. {0,0,128,0,0}
color AQUA	 Aqua color without alpha. {0,255,255,0,0}
color CYAN	 Cyan color without alpha. {0,255,255,0,0}
color TURQUOISE	 Turquoise color without alpha. {0,255,255,0,0}
color TEAL	 Teal color without alpha. {0,128,128,0,0}
color FUCHSIA	 Fuchsia color without alpha. {255,0,255,0,0}
color PINK	 Pink color without alpha. {255,0,255,0,0}
color MAGENTA	 Magenta color without alpha. {255,0,255,0,0}
color PURPLE	 Purple color without alpha. {128,0,128,0,0}
color YELLOW	 Yellow color without alpha. {255,255,0,0,0}
color OLIVE	 Olive color without alpha. {128,128,0,0,0}
color LIGHT_GRAY	 Light gray color without alpha. {192,192,192,0,0}
color SILVER	 Silver color without alpha. {192,192,192,0,0}
color GRAY	 Gray color without alpha. {128,128,128,0,0}
color DARK_GRAY	 Dark gray color without alpha. {64,64,64,0,0}
color ORANGE	 Orange color without alpha. {255,128,0,0,0}
color BROWN	 Brown color without alpha. {139,69,19,0,0}

Variable / Define	Description
color SKY	 Sky color without alpha. {0,191,255,0,0}
color GOLD	 Gold color without alpha. {238,201,0,0,0}
color WHITE_ALPHA	 White color with alpha. {255,255,255,255,255}
color BLACK_ALPHA	 Black color with alpha. {0,0,0,0,255}
color COLOR	COLOR may be used as last parameter for drawing functions. The functions then draws the color channels red, green, blue, and white.
color ALPHA	ALPHA may be used as last parameter for drawing functions. The functions then draws only the alpha channel.
color COLOR_AND_ALPHA	COLOR_AND_ALPHA may be used as last parameter for drawing functions. The function then draws both the color and the alpha channel.
Mix Modes	
int MIXMODE_NORMAL	The normal mix mode.
int MIXMODE_DARKEN	The darken (LTP) mix mode.
int MIXMODE_MULTIPLY	The multiply/mask mix mode.
int MIXMODE_COLORBURN	The color burn mix mode.
int MIXMODE_LINEARBURN	The linear burn mix mode.
int MIXMODE_LIGHTEN	The lighten (HTP) mix mode.
int MIXMODE_SCREEN	The screen mix mode.
int MIXMODE_COLORDODGE	The color dodge mix mode.
int MIXMODE_LINEARODGE	The linear dodge mix mode.
int MIXMODE_OVERLAY	The overlay mix mode.
int MIXMODE_SOFTLIGHT	The softlight mix mode.
int MIXMODE_HARDLIGHT	The hard light mix mode.
int MIXMODE_VIVIDLIGHT	The vivid light mix mode.
int MIXMODE_LINEARLIGHT	The linear light mix mode.
int MIXMODE_PINLIGHT	The pin light mix mode.
int MIXMODE_HARDMIX	The hardmix mix mode.
int MIXMODE_DIFFERENCE	The difference mix mode.

Variable / Define	Description
int MIXMODE_EXCLUSION	The exclusion mix mode.
int MIXMODE_AND	The and mix mode.
int MIXMODE_OR	The or mix mode.
int MIXMODE_XOR	The xor mix mode.
int MIXMODE_NAND	The not and mix mode.
int MIXMODE_NOR	The not or mix mode.
int MIXMODE_MASK	The RGBW alpha mask mix mode.
Filters / Storage Place and Layer Effects (FX)	
General Filters	
int FILTER_NONE	Deactivates the filter.
Blur/Sharpen Filters	
int FILTER_BLUR	This filter blurs the output.
int FILTER_BLUR_BSPLINE	This filter blurs the output applying a B-spline.
int FILTER_BLUR_CATMULL_ROM	This filter blurs the output applying a Catmull-Rom spline.
int FILTER_BLUR_GAUSS	This filter blurs the output applying the Gaussian function.
int FILTER_BLUR_MITCHELL	This filter blurs the output applying the Mitchell-Netravali function.
int FILTER_SHARPE	This filter sharpens the output.
Color Correction Filters	
int FILTER_BRIGHTEN	The brighten filter to light up the whole matrix.
int FILTER_DARKEN	The darken filter to darken the whole matrix.
int FILTER_GREYSCALE	The greyscale filter to render the matrix greyscale, i.e. in grey colors.
int FILTER_INVERT_COLOR	The invert color filter to invert every color channel.
Color Mask Filters	
int FILTER_RED	The red filter to filter out every color except the red color channel.
int FILTER_GREEN	The green filter to filter out every color except the green color channel.
int FILTER_BLUE	The blue filter to filter out every color except the blue color channel.
int FILTER_WHITE	The white filter to filter out every color except the white color channel.
int FILTER_RED_GREEN	The red/green filter to filter out every color except the red and the green color channel.

Variable / Define	Description
int FILTER_RED_BLUE	The red/blue filter to filter out every color except the red and the blue color channel.
int FILTER_GREEN_BLUE	The green/blue filter to filter out every color except the green and the blue color channel.
int FILTER_RED_WHITE	The red/white filter to filter out every color except the red and the white color channel.
int FILTER_GREEN_WHITE	The green/white filter to filter out every color except the green and the white color channel.
int FILTER_BLUE_WHITE	The blue/white filter to filter out every color except the blue and the white color channel.
int FILTER_RED_GREEN_BLUE	The red/green/blue filter to filter out every color except the red, the green, and the blue color channel.
int FILTER_RED_GREEN_WHITE	The red/green/white filter to filter out every color except the red, the green, and the white color channel.
int FILTER_RED_BLUE_WHITE	The red/blue/white filter to filter out every color except the red, the blue, and the white color channel.
int FILTER_GREEN_BLUE_WHITE	The green/blue/white filter to filter out every color except the green, the blue, and the white color channel.
Style Filters	
int FILTER_EDGES	The edges filter to make the edges of objects/motifs stand out.
int FILTER_EDGES_POPUP	The edges popup filter to make the edges of objects/motifs stand out.
int FILTER_EMOSS	The emboss filter to create an image with just highlights and shadows.
int FILTER_EMOSS_POPUP	The emboss popup filter to create an image with just highlights and shadows depending on the motif.
Transformation Filters	
int FILTER_INVERT_H_MATRIX	The filter flips the matrix horizontally.
int FILTER_INVERT_V_MATRIX	The filter flips the matrix vertically.
int FILTER_INVERT_HV_MATRIX	The filter flips the matrix horizontally and vertically. Therefore it instantly rotates the matrix by 180°.
Shifting the Matrix	
int SHIFT_UP	Shifts the content of the defined area upwards.
int SHIFT_DOWN	Shifts the content of the defined area downwards.
int SHIFT_LEFT	Shifts the content of the defined area to the left.
int SHIFT_RIGHT	Shifts the content of the defined area to the right.
int SHIFT_UL	Shifts the content of the defined area to the upper left.
int SHIFT_UR	Shifts the content of the defined area to the upper right.
int SHIFT_DL	Shifts the content of the defined area to the bottom left.

Variable / Define	Description
int SHIFT_DR	Shifts the content of the defined area to the bottom right.
int SHIFT_H_IN_OUT	Shifts the content of the defined area from the middle of the the area outwards in a horizontal direction.
int SHIFT_H_OUT_IN	Shifts the content of the defined area from the outside of the the area to the middle in a horizontal direction.
int SHIFT_V_IN_OUT	Shifts the content of the defined area from the middle of the the area outwards in a vertical direction.
int SHIFT_V_OUT_IN	Shifts the content of the defined area from the outside of the the area to the middle in a vertical direction.
int SHIFT_C_IN_OUT	Shifts the content of the defined area from the middle of the the area outwards in a vertical and a horizontal direction.
int SHIFT_C_OUT_IN	Shifts the content of the defined area from the outside of the the area to the middle in vertical and horizontal direction.
Map Modes	
int MAP_MIRROR_NONE	Disables the mirror map mode.
int MAP_MIRROR_H	Mirrors the content of the matrix horizontally.
int MAP_MIRROR_V	Mirrors the content of the matrix vertically.
int MAP_MIRROR_HV	Mirrors the content of the matrix both vertically and horizontally.
int MAP_TILE_NONE	Disables mapping tile modes.
int MAP_TILE_REPEAT	Repeats tiles on the mapped matrix.
int MAP_TILE_MIRROR_H	Mirrors tiles horizontally.
int MAP_TILE_MIRROR_V	Mirrors tiles vertically.
int MAP_TILE_MIRROR_HV	Mirrors tiles both vertically and horizontally.
Directions	
int DIR_UP	Sets an upward direction.
int DIR_DOWN	Sets a downward direction.
int DIR_LEFT	Sets the direction to 'left'.
int DIR_RIGHT	Sets the direction to 'right'.
int DIR_UL	Sets the direction to the upper left corner.
int DIR_UR	Sets the direction to the upper right corner.
int DIR_DL	Sets the direction to the lower left corner.
int DIR_DR	Sets the direction to the lower right corner.
int DIR_NONE	Sets no direction. Usually this is the same as stopping the movement of an effect, if this is possible with the effect.
Others	

Variable / Define	Description
string WHITE_SPACES	A string which contains white spaces, like tab, new line, or space. This can be used as delimiter string for Tokenizing .

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Define	Description
int FILTER_INVERT_MATRIX	Please use int FILTER_INVERT_HV_MATRIX instead.

List Of Extra Script Information

Name of Value	Description
@author	The name of the author who wrote the script. » Description
@description	Any text that describes the script. » Description
@name	A name for the script. » Description
@version	The current version of this script, if there is any. » Description

3.5 List Of Operations

Arithmetical And Logical Operations

Operator	Operand/ Data Type	Results in Data Type	Description
++, --	int	int	Adds/subtracts 1 to/from the value of the operand. Superior expressions are evaluated before the value of the operand is changed.
+	int, float, bool, string	Depends on operand data types. It will be converted into the more precise data type.	Calculates the sum of the two operands or concatenates two character strings.

-, *, /	int, float, bool	Depends on operand data types. It will be converted into the more precise data type.	Calculates the difference/product/quotient of the two operands.
%	int	int	Calculates the remainder of an integer division.
<, <=, >, >=, ==, !=	int, float, bool, string	bool	Compares two operands.
!	bool	bool	This is the logical NOT. It negates the following operand.
	bool	bool	This is the logical OR. The result is true, if at least one operand is true. Both operands are evaluated in every single case.
&&	bool	bool	This is the logical AND. The result is true, if both operands are true. Both operands are evaluated in every single case.

Assignment Operations

Operator	Description
=	A simple assignment. The left operand gets the value of the right one.
+=	For example: i += 4 corresponds to i = i + 4 .
-=	For example: i -= 4 corresponds to i = i - 4 .
*=	For example: i *= 4 corresponds to i = i * 4 .
/=	For example: i /= 4 corresponds to i = i / 4 .
%=	For example: i %= 4 corresponds to i = i % 4 .

3.6 List Of Structures

Complex data types, so-called structures, consist of different elements. The elements of a structure are accessed by their names in the following way: *nameOfVariable.nameOfElement*. For example, *col.r*, if *col* is a variable of data type *color*. The following table is an overview of the structures MADRIX Script provides.

Structure	Elements	Description
color	<ul style="list-style-type: none"> int r int g int b int w 	<i>color</i> stores a color value. There are 5 channels (red, green, blue, white, alpha) with values between 0 and 255.

	<ul style="list-style-type: none"> • int a 	<p>Example: color c = {255, 255, 0, 0}; Members examples: c.r, c.g, c.b, c.w, c.a</p> <p>»Script Example</p>
date	<ul style="list-style-type: none"> • int day • int weekday • int month • int year 	<p><i>date</i> stores a date. Values for <i>day</i> include 1 to 31 for a single day of the month. Values for <i>weekday</i> include: 0 = Sunday, 1 = Monday, ..., 6 = Saturday. Values for <i>month</i> include 1 to 12 for every single month of the year. Vales for <i>year</i> include year dates.</p> <p>Example: date d = {24, 11, 1980}; Members examples: d.day, d.weekday, d.month, d.year</p> <p>»Script Example</p>
time	<ul style="list-style-type: none"> • int hour • int min • int sec 	<p><i>time</i> stores a certain time. Valid values are: hours: 0 .. 23, minutes: 0 .. 59, seconds: 0 .. 59.</p> <p>Example: time t = {12, 05, 00}; Members examples: t.hour, t.min, t.sec</p> <p>»Script Example</p>
font	<ul style="list-style-type: none"> • int height • int width • int escapement • int orientation • int weight • int italic • int underline • int strikeOut • int charset • int outprecision • int clipprecision • int quality • int pitch • int family • string fontname 	<p><i>height</i> specifies the size of the font and requires an <i>integer</i> value. <i>width</i> specifies the wideness of the font and requires an <i>integer</i> value. <i>escapement</i> specifies the desired rotation angle in tenths of a degree and requires an <i>integer</i> value. <i>orientation</i> should be set to the same value as <i>escapement</i> and requires an <i>integer</i> value.</p> <p><i>weight</i> specifies the weight of the font. valid values are FONT_WEIGHT_DONT CARE, FONT_WEIGHT_THIN, FONT_WEIGHT_EXTRALIGHT, FONT_WEIGHT_LIGHT, FONT_WEIGHT_NORMAL, FONT_WEIGHT_MEDIUM, FONT_WEIGHT_SEMIBOLD, FONT_WEIGHT_BOLD, FONT_WEIGHT_EXTRABOLD, FONT_WEIGHT_HEAVY</p> <p><i>italic</i> specifies the sloping of the font and requires an <i>integer</i> value: 0 (off) or 1 (on).</p> <p><i>underline</i> draws a line under the font and requires an <i>integer</i> value: 0 (off) or 1 (on).</p> <p><i>strikeOut</i> draws a line through the middle of the font and requires an <i>integer</i> value: 0 (off) or 1 (on).</p> <p><i>charset</i> specifies the character set of the font. Valid values are CHARSET_ANSI, CHARSET_DEFAULT, CHARSET_SYMBOL, CHARSET_SHIFTJIS, CHARSET_HANGEUL, CHARSET_HANGUL, CHARSET_GB2312, CHARSET_CHINESEBIG5, CHARSET_OEM, CHARSET_JOHAB, CHARSET_HEBREW, CHARSET_ARABIC, CHARSET_GREEK, CHARSET_TURKISH, CHARSET_VIETNAMESE, CHARSET_THAI, CHARSET_EASTEUROPE, CHARSET_RUSSIAN, CHARSET_MAC, CHARSET_BALTIC</p>

		<p><i>outprecision</i> specifies how closely the output must match the requested height, weight, and other attributes of a font. Valid values are PRECIS_OUT_DEFAULT, PRECIS_OUT_STRING, PRECIS_OUT_CHARACTER, PRECIS_OUT_STROKE, PRECIS_OUT_TT, PRECIS_OUT_DEVICE, PRECIS_OUT_RASTER, PRECIS_OUT_TT_ONLY, PRECIS_OUT_OUTLINE, PRECIS_OUT_SCREEN_OUTLINE, PRECIS_OUT_PS_ONLY</p> <p><i>clipprecision</i> specifies how to clip characters that are partially outside the clipping region. Valid values are PRECIS_CLIP_DEFAULT, PRECIS_CLIP_CHARACTER, PRECIS_CLIP_STROKE, PRECIS_CLIP_MASK, PRECIS_CLIP_LH_ANGLES, PRECIS_CLIP_TT_ALWAYS, PRECIS_CLIP_DFA_DISABLE (Windows Vista and up), PRECIS_CLIP_EMBEDDED</p> <p><i>quality</i> specifies the quality of the font. valid values are QUALITY_DEFAULT, QUALITY_DRAFT, QUALITY_PROOF, QUALITY_NONANTIALIASED, QUALITY_ANTIALIASED, QUALITY_CLEARTYPE (Windows XP and up), QUALITY_CLEARTYPE_NATURAL (Windows XP and up)</p> <p><i>pitch</i> specifies the pitch of the font. Valid values for are PITCH_DEFAULT, PITCH_FIXED, PITCH_VARIABLE, PITCH_MONO_FONT</p> <p><i>family</i> specifies the font family that describes the font in a general way. Valid values are FONT_FAMILY_DONTCARE, FONT_FAMILY_ROMAN, FONT_FAMILY_SWISS, FONT_FAMILY_MODERN, FONT_FAMILY_SCRIPT, FONT_FAMILY_DECORATIVE</p> <p><i>fontname</i> requires a <i>string</i>. For example "Arial".</p> <p>»Script Example</p>
--	--	---

3.7 Table Of Frequencies

The two global fields SOUND_DATA_LEFT and SOUND_DATA_RIGHT hold the volume of the frequencies. The following table indicates which value defines which frequency.

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
0	5.38	171	1,841.09	341	5,733.22
1	10.77	172	1,857.24	342	5,765.52
2	16.15	173	1,873.39	343	5,792.43
3	21.53	174	1,894.92	344	5,819.35
4	26.92	175	1,911.07	345	5,851.65

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
5	32.30	176	1,927.22	346	5,878.56
6	37.68	177	1,943.37	347	5,905.48
7	43.07	178	1,959.52	348	5,937.78
8	48.45	179	1,975.67	349	5,964.70
9	53.83	180	1,991.82	350	5,991.61
10	59.22	181	2,013.35	351	6,023.91
11	64.60	182	2,029.50	352	6,056.21
12	69.98	183	2,045.65	353	6,088.51
13	75.37	184	2,067.19	354	6,115.43
14	80.75	185	2,083.34	355	6,142.35
15	86.13	186	2,099.49	356	6,174.65
16	91.52	187	2,121.02	357	6,206.95
17	96.90	188	2,137.17	358	6,239.25
18	102.28	189	2,153.32	359	6,266.16
19	107.67	190	2,174.85	360	6,293.08
20	113.05	191	2,191.00	361	6,325.38
21	123.82	192	2,207.15	362	6,357.68
22	129.20	193	2,228.69	363	6,389.98
23	134.58	194	2,244.84	364	6,416.89
24	139.97	195	2,260.99	365	6,443.81
25	145.35	196	2,282.52	366	6,476.11
26	150.73	197	2,304.05	367	6,508.41
27	156.12	198	2,320.20	368	6,540.71
28	166.88	199	2,336.35	369	6,573.01
29	172.27	200	2,357.89	370	6,605.31
30	177.65	201	2,379.42	371	6,637.61
31	188.42	202	2,400.95	372	6,664.53
32	193.80	203	2,417.10	373	6,691.44
33	199.18	204	2,433.25	374	6,723.74
34	209.95	205	2,454.79	375	6,756.04
35	215.33	206	2,476.32	376	6,788.34
36	220.72	207	2,492.47	377	6,820.64
37	226.10	208	2,508.62	378	6,852.94
38	231.48	209	2,530.15	379	6,885.24

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
39	242.25	210	2,551.68	380	6,917.54
40	247.63	211	2,573.22	381	6,949.84
41	253.02	212	2,594.75	382	6,982.14
42	263.78	213	2,610.90	383	7,014.44
43	274.55	214	2,627.05	384	7,046.74
44	279.93	215	2,648.58	385	7,079.04
45	285.31	216	2,670.12	386	7,111.34
46	296.08	217	2,691.65	387	7,143.64
47	306.85	218	2,713.18	388	7,175.94
48	312.23	219	2,734.72	389	7,208.24
49	317.61	220	2,750.87	390	7,240.54
50	328.38	221	2,767.02	391	7,272.84
51	339.15	222	2,788.55	392	7,305.14
52	344.53	223	2,810.08	393	7,337.44
53	349.91	224	2,831.62	394	7,369.74
54	360.68	225	2,853.15	395	7,402.04
55	371.45	226	2,874.68	396	7,434.34
56	382.21	227	2,896.22	397	7,466.64
57	387.60	228	2,917.75	398	7,498.94
58	392.98	229	2,939.28	399	7,531.24
59	403.75	230	2,960.82	400	7,563.54
60	414.51	231	2,982.35	401	7,595.84
61	425.28	232	3,003.88	402	7,628.14
62	436.05	233	3,025.42	403	7,660.44
63	441.43	234	3,046.95	404	7,698.12
64	446.81	235	3,063.10	405	7,735.80
65	457.58	236	3,079.25	406	7,768.10
66	468.35	237	3,100.78	407	7,800.40
67	479.11	238	3,122.31	408	7,832.70
68	489.88	239	3,149.23	409	7,865.00
69	495.26	240	3,176.15	410	7,897.30
70	500.65	241	3,197.68	411	7,934.99
71	511.41	242	3,219.21	412	7,972.67
72	522.18	243	3,240.75	413	8,004.97

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
73	532.95	244	3,262.28	414	8,037.27
74	543.71	245	3,283.81	415	8,069.57
75	554.48	246	3,305.35	416	8,101.87
76	565.25	247	3,326.88	417	8,139.55
77	576.01	248	3,348.41	418	8,177.23
78	586.78	249	3,369.95	419	8,209.53
79	597.55	250	3,391.48	420	8,241.83
80	608.31	251	3,413.01	421	8,274.13
81	619.08	252	3,434.55	422	8,311.82
82	629.85	253	3,456.08	423	8,349.50
83	640.61	254	3,483.00	424	8,381.80
84	651.38	255	3,509.91	425	8,414.10
85	662.15	256	3,531.45	426	8,451.78
86	672.91	257	3,552.98	427	8,489.46
87	683.68	258	3,574.51	428	8,521.77
88	694.45	259	3,596.04	429	8,554.06
89	705.21	260	3,617.58	430	8,591.75
90	715.98	261	3,644.49	431	8,629.43
91	726.75	262	3,671.41	432	8,661.73
92	737.51	263	3,692.94	433	8,699.41
93	748.28	264	3,714.48	434	8,737.10
94	759.05	265	3,736.01	435	8,769.40
95	769.81	266	3,757.54	436	8,807.08
96	780.58	267	3,784.46	437	8,844.76
97	791.35	268	3,811.38	438	8,877.06
98	802.11	269	3,832.91	439	8,914.75
99	812.88	270	3,854.44	440	8,952.43
100	823.65	271	3,881.36	441	8,984.73
101	834.41	272	3,908.28	442	9,022.41
102	850.56	273	3,929.81	443	9,060.10
103	866.71	274	3,951.34	444	9,092.39
104	877.48	275	3,972.88	445	9,130.08
105	888.24	276	3,999.79	446	9,167.76
106	899.01	277	4,026.71	447	9,200.06

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
107	909.78	278	4,048.24	448	9,237.74
108	920.54	279	4,069.78	449	9,275.43
109	931.31	280	4,096.69	450	9,313.11
110	942.08	281	4,123.61	451	9,350.79
111	958.23	282	4,145.14	452	9,388.48
112	974.38	283	4,172.06	453	9,426.16
113	985.14	284	4,198.97	454	9,463.84
114	995.91	285	4,220.51	455	9,501.53
115	1,006.68	286	4,247.42	456	9,533.83
116	1,022.83	287	4,274.34	457	9,571.51
117	1,038.98	288	4,295.87	458	9,609.19
118	1,049.74	289	4,322.79	459	9,646.88
119	1,060.51	290	4,349.71	460	9,684.56
120	1,076.66	291	4,371.24	461	9,722.24
121	1,092.81	292	4,398.16	462	9,759.92
122	1,103.58	293	4,425.07	463	9,797.61
123	1,114.34	294	4,446.61	464	9,835.29
124	1,125.11	295	4,473.52	465	9,872.97
125	1,141.26	296	4,500.44	466	9,910.66
126	1,157.41	297	4,527.36	467	9,948.34
127	1,168.18	298	4,554.27	468	9,986.02
128	1,184.33	299	4,581.19	469	10,023.71
129	1,200.48	300	4,608.11	470	10,061.39
130	1,211.24	301	4,629.64	471	10,099.07
131	1,222.01	302	4,656.56	472	10,142.14
132	1,238.16	303	4,683.47	473	10,179.82
133	1,254.31	304	4,710.39	474	10,217.50
134	1,265.08	305	4,737.30	475	10,255.19
135	1,281.23	306	4,758.84	476	10,292.87
136	1,297.38	307	4,785.75	477	10,330.55
137	1,308.14	308	4,812.67	478	10,368.24
138	1,324.29	309	4,839.59	479	10,411.30
139	1,340.44	310	4,866.50	480	10,448.99
140	1,351.21	311	4,893.42	481	10,486.67

Index	Spectrum in Hz	Index	Spectrum in Hz	Index	Spectrum in Hz
141	1,367.36	312	4,920.34	482	10,529.74
142	1,383.51	313	4,947.25	483	10,567.42
143	1,399.66	314	4,979.55	484	10,605.10
144	1,415.81	315	5,006.47	485	10,642.79
145	1,426.57	316	5,033.39	486	10,680.47
146	1,442.72	317	5,060.30	487	10,723.54
147	1,458.87	318	5,087.22	488	10,761.22
148	1,469.64	319	5,114.14	489	10,798.90
149	1,485.79	320	5,141.05	490	10,841.97
150	1,501.94	321	5,167.97	491	10,879.65
151	1,518.09	322	5,194.89	492	10,917.33
152	1,534.24	323	5,221.80	493	10,960.40
153	1,550.39	324	5,248.72	494	11,003.47
154	1,566.54	325	5,281.02	495	11,041.15
155	1,582.69	326	5,307.93	496	11,078.83
156	1,598.84	327	5,334.85	497	11,121.90
157	1,614.99	328	5,361.77	498	11,159.58
158	1,631.14	329	5,388.68	499	11,197.27
159	1,641.91	330	5,420.98	500	11,240.33
160	1,658.06	331	5,447.90	501	11,283.40
161	1,674.21	332	5,474.82	502	11,321.08
162	1,690.36	333	5,501.73	503	11,358.76
163	1,706.51	334	5,528.65	504	11,401.83
164	1,722.66	335	5,560.95	505	11,444.90
165	1,738.81	336	5,587.87	506	11,487.96
166	1,754.96	337	5,614.78	507	11,525.65
167	1,771.11	338	5,647.08	508	11,563.33
168	1,787.26	339	5,674.00	509	11,606.40
169	1,808.79	340	5,700.92	510	11,649.46
170	1,824.94				

3.8 Table Of Notes

The following table provides an overview about the notes MADRIX is able to recognize during the music analysis and the corresponding indices, which must be used for functions like *GetNoteValue* to retrieve the value.

Note	C	C#	D	D#	E	F	F#	G	G#	A	A#	H
Index	0	1	2	3	4	5	6	7	8	9	10	11
Freq. (Hz)	8.25	8.56	9.28	9.90	10.30	11.00	11.60	12.38	13.2	13.75	14.85	15.47
Index	12	13	14	15	16	17	18	19	20	21	22	23
Freq. (Hz)	16.50	17.19	18.60	19.80	20.63	22.00	23.20	24.75	26.40	27.50	29.70	30.94
Index	24	25	26	27	28	29	30	31	32	33	34	35
Freq. (Hz)	33.00	34.38	37.12 8	39.60	41.25	44.00	46.40	49.50	52.80	55.00	59.40	61.88
Index	36	37	38	39	40	41	42	43	44	45	46	47
Freq. (Hz)	66.00	68.75	74.25	79.20	82.50	88.00	92.80	99.00	105.6 0	110.0 0	118.8 0	123.7 5
Index	48	49	50	51	52	53	54	55	56	57	58	59
Freq. (Hz)	132.00	137.0 0	148.5 0	158.4 0	165.0 0	176.0 0	185.6 0	198.0 0	211.2 0	220.0 0	237.6 0	247.5 0
Index	60	61	62	63	64	65	66	67	68	69	70	71
Freq. (Hz)	264.00	275.0 0	297.0 0	316.8 0	330.0 0	352.0 0	371.2 5	396.0 0	422.4 0	440.0 0	475.2 0	495.0 0
Index	72	73	74	75	76	77	78	79	80	81	82	83
Freq. (Hz)	528.00	550.0 0	594.0 0	633.6 0	660.0 0	704.0 0	742.5 0	792.0 0	844.8 0	880.0 0	950.4 0	990.0 0
Index	84	85	86	87	88	89	90	91	92	93	94	95
Freq. (Hz)	1,056	1,100	1,188	1,267 .2	1,320	1,408	1,485	1,584	1,689 .6	1,760	1,900 .8	1,980
Index	96	97	98	99	100	101	102	103	104	105	106	107
Freq. (Hz)	2,112	2,200	2,376	2,534 .4	2,640	2,816	2,970	3,168	3,379 .2	3,520	3,801 .6	3,960
Index	108	109	110	111	112	113	114	115	116	117	118	119
Freq. (Hz)	4,224	4,400	4,752	5,068 .8	5,280	5,632	5,940	6,336	6,758 .4	7,040	7,603 .2	7,920
Index	120	121	122	123	124	125	126	127				
Freq. (Hz)	8,558	8,800	9,504	10,13 7	10,56 0	11,26 4	11,88 0	12,67 2				

3.9 Examples

Script Examples for Specific Functions

GetApplicationPath

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example GetApplicationPath";
@author=" ";
@version="MADRIX 2.8a";
@description="Display the application path,
running from script engine version 1.25 and MADRIX 2.8a";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    WriteText(GetApplicationPath());
}
```

(Description)

GetUserProfileDirectory

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example GetUserProfileDirectory";
@author=" ";
@version="MADRIX 2.8a";
@description="Display the user profile directory,
running from script engine version 1.25 and MADRIX 2.8a";

void InitEffect()
{

}

void PreRenderEffect()
```

```
{  
}  
  
void PostRenderEffect()  
{  
    WriteText(GetUserProfileDirectory());  
}
```

(Description)

CheckScriptEngineVersion

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example CheckScriptEngineVersion";  
@author="";  
@version="MADRIX 2.8a";  
@description="Check script engine version number,  
running from script engine version 1.25 and MADRIX 2.8a";  
  
void InitEffect()  
{  
}  
void PreRenderEffect()  
{  
}  
void PostRenderEffect()  
{  
    if(CheckScriptEngineVersion(1,25)>0)  
        WriteText("Script engine version ok");  
    else  
        WriteText("Script engine version too old");  
}
```

(Description)

CheckSoftwareVersion

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example CheckSoftwareVersion";  
@author="";  
@version="MADRIX 2.8a";  
@description="Check software version number,  
running from script engine version 1.25 and MADRIX 2.8a";  
  
void InitEffect()  
{  
}  
void PreRenderEffect()  
{  
}  
void PostRenderEffect()
```

```
{
    if(CheckSoftwareVersion(2,8,1,0)>0)
        WriteText("Madrix version ok");
    else
        WriteText("Madrix version too old");
}
```

(Description)

GetScriptEngineVersion

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example GetScriptEngineVersion";
@author="";
@version="MADRIX 2.8a";
@description="Get MADRIX version,
running from script engine version 1.25 and MADRIX 2.8a";

void InitEffect()
{
}
void PreRenderEffect()
{
}
void PostRenderEffect()
{
    WriteText("ScriptEngine " +GetScriptEngineVersion());
}
```

(Description)

GetSoftwareVersion

Copy and paste the script below and monitor the 'Script output' to see the result.

```
@scriptname="Example GetSoftwareVersion";
@author="";
@version="MADRIX 2.8a";
@description="Get MADRIX version,
running from script engine version 1.25 and MADRIX 2.8a";

void InitEffect()
{
}
void PreRenderEffect()
{
}
void PostRenderEffect()
{
    WriteText("MADRIX " +GetSoftwareVersion());
}
```

(Description)

SetText (SCE Ticker) and GetTime

Copy and paste the script into the Macro Editor of the effect SCE Ticker and monitor the Preview Windows to see the result.

```
@scriptname="local ticker time";
@author="jky";
@version="MADRIX 2.10";
@description="Set time in SCE_Ticker text with GetTime and pm & am";

void InitEffect()
{
}

void PreRenderEffect()
{
    time t=GetTime();
    string m,s;
    if(t.min<10)
        m="0"+(string)t.min;
    else
        m=(string)t.min;
    if(t.sec<10)
        s="0"+(string)t.sec;
    else
        s=(string)t.sec;
    if(t.hour>12)
        SetText((string)(t.hour-12)+":"+m+": "+s+" pm");
    else
        SetText((string) t.hour +":"+m+": "+s+" am");
}

void PostRenderEffect()
{
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

DrawPixelText - Font Size

Draws the text "Hello", which constantly increases over and over again.

```
@scriptname="";
@author="";
@version="";
@description="";

font f={10,
        0,
        0,
        0,
        FONT_WEIGHT_BOLD,
        0,
        0,
        0,
        CHARSET_DEFAULT,
        PRECIS_OUT_DEFAULT,
        PRECIS_CLIP_DEFAULT,
        QUALITY_DEFAULT,
        PITCH_DEFAULT,
        FONT_FAMILY_SWISS,
        "Arial"};

int i=0;

void InitEffect()
{
    i=0;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
    f.height=i%40;
    DrawPixelText(WHITE,f,"Hello",0,0,ROTATION_TEXT_NONE);
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

DrawPixelText - Font Color

Draws the text "Hello" and changes its color.

```
@scriptname="";
@author="";
@version="";
@description="";

font f={20,
        0,
        0,
        0,
        FONT_WEIGHT_BOLD,
        0,
        0,
        0,
        CHARSET_DEFAULT,
        PRECIS_OUT_DEFAULT,
        PRECIS_CLIP_DEFAULT,
        QUALITY_DEFAULT,
        PITCH_DEFAULT,
        FONT_FAMILY_SWISS,
        "Arial"};

int i=0;
color col;
void InitEffect()
{
    i=0;
    col=WHITE;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
    col.r=i%255;
    col.g=(i%512)/2;
    col.b=(i%767)/3;
    DrawPixelText(col,f,"Hello",0,0,ROTATION_TEXT_NONE);
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

DrawPixelText - Moving Text

Draws the text "Hello", which moves from the upper left to the lower right.

```
@scriptname="";
@author="";
@version="";
@description="";

font f={20,
        0,
        0,
        0,
        FONT_WEIGHT_BOLD,
        0,
        0,
        0,
        CHARSET_DEFAULT,
        PRECIS_OUT_DEFAULT,
        PRECIS_CLIP_DEFAULT,
        QUALITY_DEFAULT,
        PITCH_DEFAULT,
        FONT_FAMILY_SWISS,
        "Arial"};

int i=0;

void InitEffect()
{
    i=0;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
    i=i%50;
    DrawPixelText(RED,f,"Hello",i,i,ROTATION_TEXT_NONE);
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

DrawPixelText - Rotating Text

Draws the text "Hello", which rotates by 90°, 180°, and 270°.

```
@scriptname="";
@author="";
@version="";
@description="";

font f={20,
        0,
        0,
        0,
        FONT_WEIGHT_BOLD,
        0,
        0,
        0,
        CHARSET_DEFAULT,
        PRECIS_OUT_DEFAULT,
        PRECIS_CLIP_DEFAULT,
        QUALITY_DEFAULT,
        PITCH_DEFAULT,
        FONT_FAMILY_SWISS,
        "Arial"};

int i=0;

void InitEffect()
{
    i=0;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
    i=i%400;
    switch(i/100)
    {
        case 0:DrawPixelText(RED,f,"Hello",25,25,ROTATION_TEXT_NONE);break;
        case 1:DrawPixelText(RED,f,"Hello",25,25,ROTATION_TEXT_90);break;
        case 2:DrawPixelText(RED,f,"Hello",25,25,ROTATION_TEXT_180);break;
        case 3:DrawPixelText(RED,f,"Hello",25,25,ROTATION_TEXT_270);break;
    }
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

GetTimeCode

Retrieves the currently used Time Code. This Script works in all four Script locations.

```
@scriptname="GetTimeCode";
@author="jky";
@version="2.14a";
@description="Returns the currently used Time Code";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    time TimeCode=GetTimeCode();
    WriteText("Timecode: "+(string)TimeCode.hour+": "+(string)TimeCode.min+": "
+(string)TimeCode.sec);
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

GetDmxIn

Uses incoming DMX-IN data to show colors on the LED matrix. This Script works in all four Script locations.

```
@scriptname="DmxInToColor";
@author="jky";
@version="1.0";
@description="Read DMX-IN data and makes to matrix color";

const int CHANNEL_START=0; // start by channel 1
const int CHANNEL_COUNT=3; // use this number of channels
const int UNIVERSE=0; // use this universe for DMX-IN data
int DmxValues[]; // field of DMX Universe
color col;
```

```

void InitEffect()
{
    col=BLACK;
    if(IsDmxInEnabled()==0)// if DMX-In is enabled
        WriteText("DMX-IN is disabled!");
}

void RenderEffect()
{
    if(IsDmxInEnabled()==1)// if DMX-In enabled?
    {
        // Get the DMX values from selected Universe
        GetDmxIn(DmxValues,CHANNEL_START,CHANNEL_COUNT,UNIVERSE);
        // set dmx value to color
        col.r= DmxValues[0]; // channel 1 to red
        col.g= DmxValues[1]; // channel 2 to green
        col.b= DmxValues[2]; // channel 3 to blue
    }
    else
        col=BLACK;// no DMX then no color
    Clear(col);// set complete matrix with color

    //Information for help
    //WriteText("Set color with Value Red="+(string)col.r+", Green="+(string)col.g+",
    Blue="+(string)col.b);
}

void MatrixSizeChanged()
{
    InitEffect();
}

```

(Description)

GetMidiInNoteValue and GetMidiInControlValue

Uses incoming MIDI-IN data to control the Master Fader. This Script works in the Main Output Macro.

```

@scriptname="MIDItoMaster";
@author="jky";
@version="2.14b";
@description="Uses incoming MIDI to control the Master Fader";

const int NOTE=0; // MIDI Note for control
const int CHANNEL=0; // MIDI Channel for control
const int DEVICE_ID=0; // MIDI Device for control

void InitEffect()
{

```

```
}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    if(IsMidiInEnabled()==1)
    {
        const float Value = (float)GetMidiInNoteValue(NOTE,CHANNEL,DEVICE_ID)/127.0;
        // MIDI NOTES 0x9, 0x8
        //const float Value = (float)GetMidiInControlValue(NOTE,CHANNEL,DEVICE_ID)/127.0;
        // MIDI CONTROLLER 0xb
        SetMasterFader(Value*255);
    }
}

void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

GetMidiInNote and GetMidiInControl

Uses incoming MIDI-IN data to control the Master Fader and the Audio Level. This Script works in the Main Output Macro.

```
@scriptname="MIDItoMasterandAudio";
@author="jky";
@version="2.14b";
@description="Uses incoming MIDI of 2 channels to control the Master Fader and Audio Level";

const int NOTE=0; // MIDI Note for control
const int NOTE_COUNT=2; // MIDI Note Count for control
const int CHANNEL=0; // MIDI Channel for control
const int DEVICE_ID=0; // MIDI Device for control

int MidiData[];

void InitEffect()
{

}

void PreRenderEffect()
{
```

```
}

void PostRenderEffect()
{
    if(IsMidiInEnabled()==1)
    {
        GetMidiInNote(MidiData, NOTE, NOTE_COUNT, CHANNEL, DEVICE_ID);
        // MIDI NOTES 0x9, 0x8
        //GetMidiInControl(MidiData, NOTE, NOTE_COUNT, CHANNEL, DEVICE_ID);
        // MIDI NOTES 0xb
        SetMasterFader(MidiData[0]*255/127);
        SetAudioFader(MidiData[1]*255/127);
    }
}

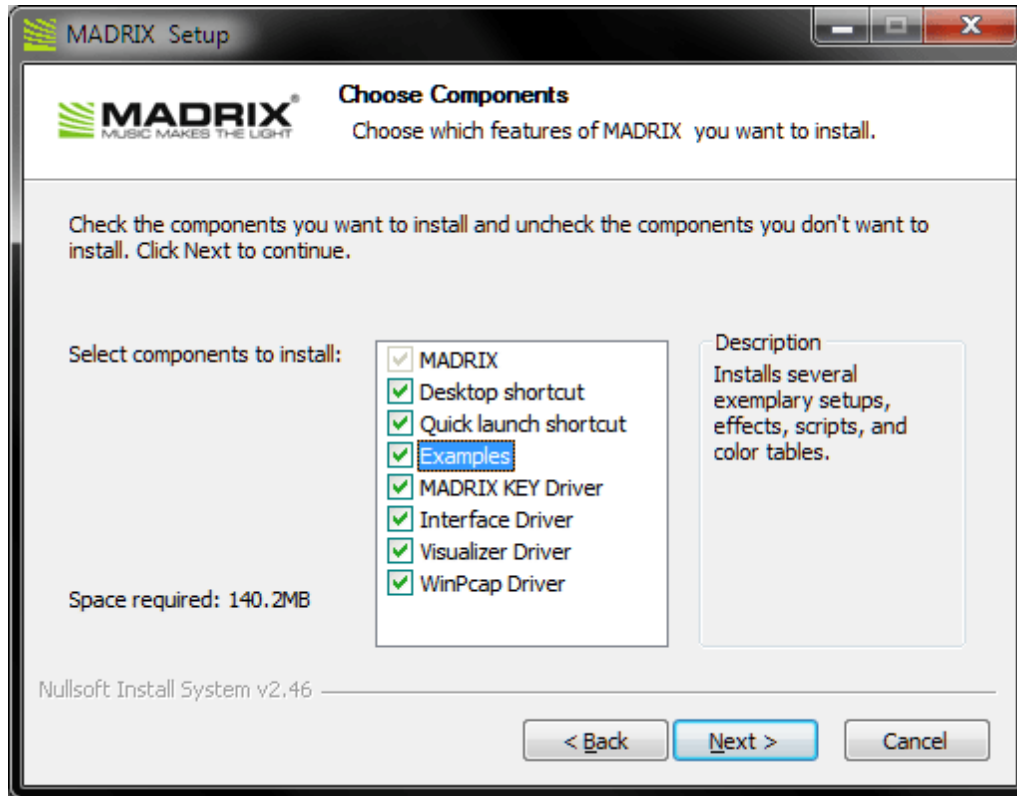
void MatrixSizeChanged()
{
    InitEffect();
}
```

(Description)

Installed Examples

Throughout this MADRIX Script Help and Manual a lot of practical script examples are already given.

If you would like to see some more examples, several exemplary scripts are already installed on your PC if you have enabled this option during the installation process.



You can find the examples on your hard disk. Please navigate to C:
\\Users\\USERNAME\\Documents\\MADRIX\\scripts.

(Please choose your Windows username for USERNAME).

The folder contains a link "Samples". This link will take you to another folder that contains various examples.

Part

IV

4 MAS Script Effect

4.1 Overview

Introduction

The MAS Script effect provides the potential to create your own light effects. If you want to create a completely new effect MADRIX does not offer, this is the right place for you offering numerous **functions**. Basically, the MAS Script effect is an effect like any other effect in MADRIX, except that it interprets a script written in MADRIX Script to calculate the effect.

In this way, the MAS Script effect has full control over the effect matrix. It is called continuously to render the effect onto the matrix. A script of the MAS script effect can also manipulate the effect speed in order to let the effect run slower or faster, for example.

It was already mentioned that scripts are stored as part of the effect. This means that they are part of a stored effect or setup. Moreover, it is possible to save scripts as separate files. The extension of a script for the MAS script effect is **.mas*. The extension of a compiled script is **.macs*.

Remember the MAS Script effect is an usual effect of MADRIX. For that reason, its result can be controlled and manipulated by a **macro** like all the other effects.



The MAS Script Effect is a normal effect of MADRIX and can be selected from the effect list like all the other effects.

Functions Called By MADRIX

There are several functions called by MADRIX in order to let the script react to different events.

- `void InitEffect()`
- `void RenderEffect()`
- `void MatrixSizeChanged()`

If a function is not needed by a script, it is not necessary to implement it. Regarding *InitEffect* and *RenderEffect* a message is printed out if one of them is missing. This is not an error, but only an information for the developer of the script.

InitEffect

(automatically included in a new script)

InitEffect is called by MADRIX whenever the script needs to be initialized. This is the case after compiling and starting a new script or when the user pressed the **"Start"** button of the **Script Editor**. A script can assume that any global variable is initialized with 0 and that any global field is empty as long as it has not been initialized with any value.

This function is the right place to initialize global variables, reset any fields, set the speed of an effect, or whatever is necessary to (re)start the script.

RenderEffect

(automatically included in a new script)

This function is called whenever the effect needs to be rendered. The number of calls per second depends on the currently set speed of the effect. It can be received with the help of the function *GetSpeed()* and set with the function *SetSpeed*. This is the right place to calculate the effect and draw it onto the matrix.

MatrixSizeChanged

(automatically included in a new script)

MatrixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings or because a new map setting was set, e.g. caused by the call of a map function.

Standard Outline

A newly created macro for an effect will look like this:

```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{

}

void RenderEffect()
{

}

void MatrixSizeChanged()
{
    InitEffect();
}
```

4.2 Functions

Specific Resources

- [Functions called by MADRIX](#)
- [MAS Script Effect: Available Functions](#)
- [Deprecated Functions](#)
- [Controlling the Tempo](#)

General Resources

- [Keyword Search](#)
- [List Of Functions \(Alphabetical Order\)](#)
- [List Of Functions \(Grouped\)](#)
- [List Of Global Variables and Constants](#)
- [List Of Operations](#)
- [List Of Structures](#)
- [Table Of Frequencies](#)
- [Table Of Notes](#)

Available Functions

It is useful to consult the [List of Functions \(Alphabetical Order\)](#) for non-specific functions.

In addition to that, there are several other functions available which are not available by macros.

MAS Script-Specific Functions

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999.
int GetBpm ()	Returns the speed of the effect in BPM.
void DoPreRender ()	Tells the effect that the script function <i>PreRenderEffect</i> is to be called before the next call of <i>RenderEffect</i> . See above for further details.
void SetFrameId (float id)	Sets a new frame <i>id</i> . If the given <i>id</i> is less than 0, it is set to 0. See below for further details.
float GetFrameId ()	Returns the ID of the current frame. See below for further details.

float GetFrameSteps ()	Returns the number of frames which are between this and the last call. See below for further details.
void SetFrameCount (float fc)	Set the number of frames the effect produces before it gets repeated. See below for further details.
float GetFrameCount ()	Get the number of frames the effect produces before it gets repeated. This is the same value which was set by <i>SetFrameCount</i> . The initial value is 1000.0. See below for further details.
void SetFixedFrameRate (int bpm, int enable)	Tells the effect to use a fixed frame rate which is independent of the speed set by <i>SetBpm</i> or by the slider of the GUI. <i>bpm</i> tells the effect with how much BPM it should render. Valid values range from 0 to 3000, that is between 0Hz and 50Hz. If <i>enable</i> is <i>true</i> , fixed frame rate rendering will be enabled, otherwise it will be disabled. If <i>enable</i> is <i>false</i> , the <i>bpm</i> value is ignored and the current set beats per minute will determine the speed.
void SetUseFloatFrames (int enable)	Enables usage of frames as floating-point frames. This enables rational values for the function <i>GetFrameSteps</i> , instead of integer values only. In case of having a fixed frame rate, float frames are always enabled and cannot be disabled.

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (float speed)	Sets the speed of the effect in FPS. Valid values are between inclusive 0 and 166. Please note: This function is deprecated and may be removed in one of the next releases. Use <i>SetBpm</i> instead.
float GetSpeed ()	Returns the speed of the effect in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use <i>GetBpm</i> instead.

Using 'FrameCount' And 'FrameId'

Please note: The script of the MAS Script effect itself is able to control the Frame ID and the frame count. Setting the frame count or Frame ID within both a macro and a script may result in undesired behavior. More information can be found in the chapter [Controlling a Script per 'FrameId'](#).

4.3 Using GUI Elements (User Interaction)

Introduction

This chapter is about interaction with the graphical user interface of MADRIX in addition to just writing a source code. In this way, working with elements of the GUI (graphical user interface) expands the options of the MAS Script effect widely. You will have the possibility to set values a script should use, such as a color or the text to write. MADRIX Script provides several GUI elements, which can be placed on the effects dialog. The picture below exemplarily shows the available graphical elements.



Exemplary GUI Elements of the MAS Script Effect

As you can see, it is possible to create 4 additional elements. There is a text field to enter text, a color control to select a color, a slider, and a button. Each different element will always be placed in a new line.

Creating GUI Elements

Creating a GUI element within MADRIX Script is as simple as declaring a variable. The corresponding element is created automatically. You just have to declare a type of element and a variable/name for it. Here is how it generally looks like:

```
ctrledit myText;  
ctrlcolor myColor;  
ctrlslider mySlider;  
ctrlbutton myButton;
```

It is as simple as that. MADRIX Script will create the elements on the effect dialog and they can be utilized right away. Since GUI elements should last as long as the script is loaded, those variables need to be global.

Please note: You cannot create local variables of those types. GUI elements have to be global! Furthermore, it is not possible to declare such variables as *persistent*. But their values will always be stored and reloaded automatically. It is also not allowed to copy or assign variables of those types.

Within MADRIX Script those data types are simple structures. Hence, you can initialize them like any other structures. The particular structures are described below. Here is an example which creates the GUI shown above:

```
ctrledit myText = {"Text", EDIT_ALIGN_LEFT, "MADRIX"};
ctrlcolor myCol = {"Color", MADRIX_GREEN};
ctrlslider mySlider = {"Range", 100, 150, 200};
ctrlbutton myButton = {"Button", "OnButton"};
```


Explanation:


This source code creates each GUI elements and initializes them with specific values. The first value always represents the description that is used to label the corresponding element. Then, several values may follow which are different for each structure. For example, the slider is initialized with a range of 100 to 200 and a starting value of 150.


Using GUI Elements


After creating an element, it can be used like controls of the same kind in MADRIX. A slider can be moved by using the mouse or by writing a value into the edit field next to it, etc. The script can access these values via the different elements of the corresponding structure. The table below shows those structures in detail:



List Of Available Elements



Structure	Entries			GUI Element/ Description
ctrlbutton 	string	label	The label for the button.	<i>ctrlbutton</i> provides a button. A function must be assigned that is called whenever the button was pressed. This is done by assigning the name of the function to the <i>proc</i> element of a <i>ctrlbutton</i> variable. The set function must be of the type void function() . The <i>tooltip</i> member holds the tooltip for the button, which is shown when the user holds the mouse over the button for some seconds. »Descriptive Example »Example 1 »Example 2 »Example 3
	string	proc	The name of the function which is called after the button has been pressed.	
	string	tooltip	The tooltip for the button.	


Structure	Entries			GUI Element/ Description
ctrlbutton2 	string	label1	The label for the first button.	<i>ctrlbutton2</i> provides two buttons in one line. For each button a function must be assigned, which is called whenever the corresponding button has been pressed. This is done by assigning the name of the function to the <i>proc[n]</i> element of a <i>ctrlbutton</i> variable. The function set must be of the type void function() . The <i>tooltip</i> members hold the tooltip for the buttons, which is shown when the user holds the mouse over a button for some seconds. »Description »Example 1 »Example 2 »Example 3
	string	proc1	Function handler for the first button.	
	string	label2	Label for the second button.	
	string	proc2	Function handler for the second button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	


Structure	Entries			GUI Element/ Description
ctrlbutton3 	string	label1	The label for the first button.	<i>ctrlbutton3</i> provides three buttons in one line. For each button a function must be assigned, which is called whenever the corresponding button has been pressed. This is done by assigning the name of the function to the <i>proc[n]</i> element of a <i>ctrlbutton</i> variable. The function set must be of the type void function() . The <i>tooltip</i> members hold the tooltip for each button, which are shown when the user holds the mouse over a button for some seconds. »Description »Example 1 »Example 2 »Example 3
	string	proc1	Function handler for the first button.	
	string	label2	Label for the second button.	
	string	proc2	Function handler for the second button.	
	string	label3	Label for the third button.	
	string	proc3	Function handler for the third button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	

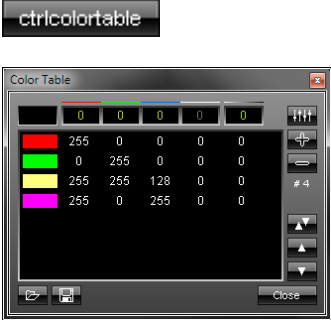
Structure	Entries	GUI Element/ Description
ctrlbutton4 	<div>string label1 The label for the first button.</div> <div>string proc1 Function handler for the first button.</div> <div>string label2 Label for the second button.</div> <div>string proc2 Function handler for the second button.</div> <div>string label3 Label for the third button.</div> <div>string proc3 Function handler for the third button.</div> <div>string label4 Label for the fourth button.</div> <div>string proc4 Function handler for the fourth button.</div> <div>string tooltip1 Tooltip for the first button.</div> <div>string tooltip2 Tooltip for the second button.</div> <div>string tooltip3 Tooltip for the third button.</div> <div>string tooltip4 Tooltip for the fourth button.</div>	<p><i>ctrlbutton4</i> provides four buttons in one line. For each button a function must be assigned, which is called whenever the corresponding button has been pressed. This is done by assigning the name of the function to the <i>proc[n]</i> element of a <i>ctrlbutton</i> variable. The function set must be of type void function().</p> <p>The <i>tooltip</i> members hold the tooltip for each button, which are shown when the user holds the mouse over a button for some seconds.</p> <p>»Description »Example 1 »Example 2 »Example 3</p>

Structure	Entries			GUI Element/ Description
ctrlcolor 	string	label	The label for the element .	<i>ctrlcolor</i> provides a color input control with the well-known Color Picker and five edit fields to enter the red, green, blue, white and alpha values. The alpha value will not be provided/set by the Color Picker. »Example
	color	value	The value set by the element .	
ctrledit 	string	label	The label for the element .	<i>ctrledit</i> provides an edit control which allows to enter a text. The variable <i>align</i> can be set to one of the following values: EDIT_ALIGN_LEFT, EDIT_ALIGN_CENTER, or EDIT_ALIGN_RIGHT. »Example
	int	align	Alignment of the edit field.	
	string	value	The text which has been entered in the edit field	
	string	tooltip	Tooltip for the edit control.	

Structure	Entries	GUI Element/ Description
ctrledit2 	<div>string label The label for the element .</div> <div>int align Alignment of the edit fields</div> <div>string value1 The first value.</div> <div>string value2 The second value.</div> <div>string tooltip1 Tooltip for the first edit control.</div> <div>string tooltip2 Tooltip for the second edit control.</div>	<p><i>ctrledit2</i> provides two edit controls in one line to enter text.</p> <p>The variable <i>align</i> can be set to one of the following values: EDIT_ALIGN_LEFT, EDIT_ALIGN_CENTER, or EDIT_ALIGN_RIGHT.</p> <p>Example: see ctrledit4</p>
ctrledit3 	<div>string label The label for the element .</div> <div>int align Alignment of the edit fields</div> <div>string value1 The first value.</div> <div>string value2 The second value.</div> <div>string value3 The third value.</div> <div>string tooltip1 Tooltip for the first edit control.</div> <div>string tooltip2 Tooltip for the second edit control.</div> <div>string tooltip3 Tooltip for the third edit control.</div>	<p><i>ctrledit3</i> provides three edit controls in one line to enter text.</p> <p>The variable <i>align</i> can be set to one of the following values: EDIT_ALIGN_LEFT, EDIT_ALIGN_CENTER, or EDIT_ALIGN_RIGHT.</p> <p>Example: see ctrledit4</p>

Structure	Entries			GUI Element/ Description
ctrledit4 	string	label	The label for the element.	<i>ctrledit4</i> provides four edit controls in one line to enter text. The variable <i>align</i> can be set to one of the following values: EDIT_ALIGN_LEFT, EDIT_ALIGN_CENTER, or EDIT_ALIGN_RIGHT. »Example
	int	align	Alignment of the edit fields.	
	string	value1	The first value.	
	string	value2	The second value.	
	string	value3	The third value.	
	string	value4	The fourth value.	
	string	tooltip1	Tooltip for the first edit control.	
	string	tooltip2	Tooltip for the second edit control.	
	string	tooltip3	Tooltip for the third edit control.	
	string	tooltip4	Tooltip for the fourth edit control.	

Structure	Entries			GUI Element/ Description
ctrlslider 	string	label	The label for the element	<p><i>ctrlslider</i> provides a slider with a predefined range. Like shown in MADRIX, a slider is represented by an edit control to enter a value as well as a slider control.</p> <p>You can set a range for the slider. The minimum value (<i>rangeMin</i>) has to be greater or equal to 0, while the maximum value (<i>rangeMax</i>) must be greater than the minimum value. If <i>value</i> itself is set to a value outside the range, it is automatically set to the minimum or maximum value.</p> <p>The <i>tooltip</i> is shown when the user leaves the mouse over the slider for some seconds.</p> <p>»Example</p>
	int	rangeMin	The minimum value the slider provides. This value must be greater than or equal to 0.	
	int	value	The start value set by the slider.	
	int	rangeMax	The maximum value the slider provides. This value must be greater than the minimum value.	
	string	tooltip	Holds the tooltip for the slider.	

Structure	Entries	GUI Element/Description
ctrlcolortable 	<div> <div>stringlabel</div> <div>Label of the button.</div> </div> <div> <div>color[]value</div> <div>A field of values which holds the entries of the color table.</div> </div> <div> <div>stringtooltip</div> <div>The tooltip for the button to open the color table.</div> </div>	<p><i>ctrlcolortable</i> provides a color table dialog to set a number of colors. On the GUI of the effect, a button is provided to open the color table dialog. The label description is used to mark the button.</p> <p>The <i>tooltip</i> is shown when the user holds the mouse over the button that opens the color table for some seconds.</p> <p>»Example</p>

Retrieving Values From GUI Elements

In order to receive values from a GUI element, it is simply necessary to read the appropriate value from the corresponding variable. The following example demonstrates that:

```

@author="";
@version="";
@description="";

ctrlcolor col = {"Color",AQUA}; //create color control
ctrlslider chgB = {"Fade", 0, 5, 150, "The value used to dim the lines"};
//create slider

void InitEffect()
{
}

void RenderEffect()
{
    //create a color structure for ChangeBrightness with the value set
    //set by the slider
    color chg = {-chgB.value, -chgB.value, -chgB.value, -chgB.value, -chgB.value};
    ChangeBrightness(chg);

    float f = frandom();

```



```
        //draw a line with the color set by the color control
        DrawVectorLine(col.value, f, 0.0, f, 1.0);
    }
```

(Description)

Explanation:

This exemplary effect randomly draws lines onto the matrix and a fade out is added.

First, one color and one slider control are created in addition to the standard speed control. The color control is used to set the color of the lines. The slider is used to determine the fading time.

In the function *RenderEffect*, first, the variable *chg* of the type *color* is created. It is initialized using the negative value of the slider control. Then, *ChangeBrightness* is called in order to fade out the current content of the matrix.

Finally, a line is drawn using the color value of the color control.

Setting Values Of GUI Elements

Just as it is possible to read the value, description, or range of a GUI element, it is also possible to set those values. In this way, it is possible to change the range of a slider, the text of an edit, or even their descriptions. You just have to set the corresponding value of the appropriate variable.

Here is an example:

```
@author="";
@version="";
@description="";

ctrlcolor col = {"Color", FUCHSIA};
ctrlslider t = {"Fade", 0, 5, 10};

void InitEffect()
{
}

void RenderEffect()
{
    col.value.a += t.value; //count up alpha value
    if(col.value.a > 255) //if necessary reset it
        col.value.a = 0;

    Clear(col.value);      //clear matrix
}
```

Explanation:

In this example the matrix is cleared with a predefined color and the alpha constantly changes in order to render the effect more and more transparent. A color control is created to change the color in use and a slider allows users to define how fast the color becomes transparent. When the color is fully transparent, it is reset to no transparency at all. In order to accomplish this effect, the script continually increases the alpha value set by the color control. You will see how the alpha value counts up by each frame and is reset to 0, after it has reached the maximum value of 255.

Using Buttons

Buttons are created like any other GUI element by declaring a global variable of one of the *ctrlbutton* elements. In order to use a button, it is necessary to assign a function which is called when the button has been pressed. Those functions must be of the following type:

```
void <function name>()
```

In order to assign such a handler function, the name of the according function must be assigned to the *proc*-element of the according variable. In theory it could look like this:

```
ctrlbutton btn;

void InitEffect()
{
    btn.proc = "OnBtn";
    btn.tooltip="A button";
}

void OnBtn()
{
    do something after button was pressed
}
```

If the assigned function does not exist or does not have the correct return type or parameter list, a warning is displayed in the Script output window of the Script Editor.

(Description)

Examples

Button Example 1 (*ctrlbutton*)

The following example creates four buttons. Pressing a button will activate the color indicated by the button (Red, Green, Blue, or White).

```
@scriptname="ColorTestWithButton";
@author="jky info@madrix.com";
@version="v1.0-2010/03/31";
@description="Create buttons that set the color of the matrix";

ctrlbutton ButtonRed = {"Red", "OnRed"};
ctrlbutton ButtonGreen= {"Green", "OnGreen"};
ctrlbutton ButtonBlue = {"Blue", "OnBlue"};
ctrlbutton ButtonWhite= {"White", "OnWhite"};

void InitEffect()
{
    ButtonRed.tooltip= "Press here to set color red";
    ButtonGreen.tooltip="Press here to set color green";
    ButtonBlue.tooltip= "Press here to set color blue";
    ButtonWhite.tooltip="Press here to set color white";
    Clear();
}

void RenderEffect()
{
    // do noting
}

void MatrixSizeChanged()
{
    InitEffect();
}

void OnRed()
{
    Clear(RED);
}

void OnGreen()
{
    Clear(GREEN);
}

void OnBlue()
{
    Clear(BLUE);
}

void OnWhite()
{
    Clear(WHITE);
}
```

```
}
```

(Description)

Button Example 2 (*ctrlbutton*)

The following example provides two buttons. By pressing the first button, blinking is started and stopped. Pressing the second button selects another color. This example changes the label and the handler function of the first button.

```
@scriptname="";
@author="";
@version="";
@description="";

ctrlbutton g_btnMode = {"Mode: blink", "OnBlinkOff",
"Enable or disable blinking mode"};
ctrlbutton g_btnColor= {"Change Color", "OnColor", "Select a new color"};

const int MODE_BLINK = 0;
const int MODE_STOP = 1;

int g_mode;
int g_counter;
color g_color = RED;

void InitEffect()
{

}

void RenderEffect()
{
    if(g_mode == MODE_BLINK)
    {
        if(g_counter++ == 10)
        {
            Clear(g_color);
            g_counter = 0;
        }
        else if(g_counter++ == 5)
        {
            Clear();
        }
    }
    else
    {
        Clear(g_color);
    }
}
```

```
void OnColor()
{
    g_color.r = random(0, 255);
    g_color.g = random(0, 255);
    g_color.b = random(0, 255);
    g_color.w = random(0, 255);
}

void OnBlinkOff()
{
    g_mode = MODE_STOP;
    g_btnMode.label = "Blinking Off";
    g_btnMode.proc = "OnBlinkOn";
    g_btnMode.tooltip = "Enable blinking mode";
    WriteText("Blink disabled");
}

void OnBlinkOn()
{
    g_mode = MODE_BLINK;
    g_btnMode.label = "Blinking On";
    g_btnMode.proc = "OnBlinkOff";
    g_btnMode.tooltip = "Disable blinking mode";
    WriteText("Blinking enabled");
}
```

(Description)

Button Example 3 (*ctrlbutton*)

The following example draws a continuing circles an provides 4 rows, each containing 1, 2, 3, or 4 buttons. The buttons are labeled with a color. Press the according button and the displayed circles will change their color.

```
@scriptname="buttons";
@author="jky";
@version="1.0";
@description="view to use button 1-4";

int height=0;
int width=0;
int i;
color col=WHITE;
ctrlbutton Button1={"Red","OnRed","set red color"};
ctrlbutton2 Button2={"Red","OnRed","Green","OnGreen","set red color", "set green color"};
ctrlbutton3 Button3={"Red","OnRed","Green","OnGreen","Blue","OnBlue","set red color", "set green color"};
ctrlbutton4 Button4={"Red","OnRed","Green","OnGreen","Blue","OnBlue","White","OnWhite","set red color"};

void InitEffect()
{

```

```
    height = GetMatrixHeight();
    width = GetMatrixWidth();
    Clear();
    i=0;
    SetBpm(100);
}

void RenderEffect()
{
    i++;
    if(i>width || i>height)
    {
        i=0;
        Clear();
    }

    DrawPixelCircle(col, width/2-i, height/2-i, i*2,i*2);
}

void MatrixSizeChanged()
{
    InitEffect();
}

void OnRed()
{
    col=RED;
}

void OnGreen()
{
    col=GREEN;
}

void OnBlue()
{
    col=BLUE;
}

void OnWhite()
{
    col=WHITE;
}
```

(Description)

Color Table (*ctrlcolortable*)

The following example clears the matrix with several predefined colors. The effect provides a color table to set up these colors. This example shows how to initialize and to use a color table within a script.

```
@author="";
@version="";
@description="";

ctrlcolortable col = {"Color Table", {RED, GREEN, {255, 255, 128}, FUCHSIA},
"Colors used to fill the matrix"};

void InitEffect()
{
    SetBpm(60);
}

void RenderEffect()
{
    if(col.value.length > 0)
    {
        Clear(col.value[random(0, col.value.length - 1)]);
    }
}
```

(Description)

Edit Field (*ctrledit*)

The following example clears the matrix with the color white. It also creates 1 edit fields to enter the color values for the red color channel. You can enter different values than 255, which is the default value of the edit field in this script. The green and blue color channel are automatically set to 255.

```
@scriptname="ctrledit1 sample";
@author="jky";
@version="MADRIX 2.10";
@description="color edit via script gui";

color col=WHITE;
// create 1x edit
ctrledit editColor={"Color",EDIT_ALIGN_RIGHT,"255","color red"};

void InitEffect()
{
}
```

```
void RenderEffect()  
{  
    col.r=(int)editColor.value;// value1 to color red via cast (int)  
    Clear(col);// Set color to complete matrix  
}  
  
void MatrixSizeChanged()  
{  
    InitEffect();  
}
```

(Description)

Edit Fields (*ctrledit4*)

The following example clears the matrix with the color white. It also creates 4 edit fields to enter color values for the red, green, blue, and white color channel. You can enter different values than 255, which is the default value of the edit fields in this script.

```
@scriptname="ctrledit4 sample";  
@author="jky";  
@version="MADRIX 2.10";  
@description="color edit via script gui";  
  
color col=WHITE;  
// create 4x edit  
ctrledit4 editColor={"Color",EDIT_ALIGN_RIGHT,"255","255","255","255",  
"color red","color green","color blue","color white"};  
  
void InitEffect()  
{  
}  
  
void RenderEffect()  
{  
    col.r=(int)editColor.value1;// value1 to color red   via cast (int)  
    col.g=(int)editColor.value2;// value2 to color green via cast (int)  
    col.b=(int)editColor.value3;// value3 to color blue  via cast (int)  
    col.w=(int)editColor.value4;// value4 to color white via cast (int)  
    Clear(col);// Set color to complete matrix  
}  
  
void MatrixSizeChanged()  
{  
    InitEffect();  
}
```

(Description)

4.4 Controlling The Tempo

Since the introduction of BPM for setting the speed of an effect, it is possible to have very fast effects in MADRIX. Furthermore, using the Speed Master it is possible to increase the speed even more. And some effects can even be played backwards. Several static effects, like SCE ColorScroll, may also be controlled via their frame ID. It enables the user to "scroll" the effect to a well defined "position".

All those things can also be done with an effect written in MADRIX Script. But the script author should take some precautions. This chapter describes what is necessary in order to have a script that can be run very fast or even backwards due to the Speed Master. A deeper insight into the topic *FrameId* is provided as well.

All the relationships will be illustrated with the help of a simple script, which moves a rectangle from left to right, or backwards.

4.4.1 Increasing The Speed Of Effects

BPM vs. FPS

The speed of the MAS Script Effect is controlled by the BPM slider (and edit field) which has a range between 0 and 9999. With regard to this, 0 implies that the effect is stopped and will not be rendered any more. Transferred to a script, it means that *RenderEffect* is not called anymore. Consequently, the effect stops automatically. 9999 is the maximum value and would mean that the effect runs with round about 166 FPS. Here is how BPM and FPS values are related to each other:

$$\begin{aligned}\text{BPM} &= \text{FPS} * 60 \\ \text{FPS} &= \text{BPM} / 60\end{aligned}$$

Since DMX maximally supports a data frequency of 43Hz (or FPS), MADRIX renders effects with a maximum of 50Hz (or FPS), which equals 3000 BPM. Any render frequency that is higher would not make any sense. That means that a script is not called more often than 50 times per second, no matter if the BPM slider shows 3000 or 9000 BPM.

To illustrate this fact, copy and paste the following script. It makes a rectangle move from the left to the right side of the matrix.

```
int g_pos[];
int g_rectSize[] = {4, 4}; //size of the rect

void MatrixSizeChanged()
{
    g_pos[1] = GetMatrixHeight() / 2 - g_rectSize[1] / 2;;
}

void InitEffect()
{
    MatrixSizeChanged();
}

void RenderEffect()
{
    Clear();
    FillPixelRect(WHITE, g_pos[0], g_pos[1], g_rectSize[0], g_rectSize[1]);
    //move the rectangle
    g_pos[0] += 1;
    //check if the rectangle has left the matrix and set it back
    if(g_pos[0] > GetMatrixWidth() + g_rectSize[0])
        g_pos[0] = -g_rectSize[0];
}
```

Subsequently, set the matrix to a width of 50. Use the BPM slider to increase or decrease the speed. You should be able to recognize that the rectangle always needs one second to move across the whole matrix, whether you select 3000 BPM or 9000 BPM. This is due to two facts. First, the rectangle is moved exactly one pixel per call. Second, the maximal render frequency of an effect, and therefore also for a script, is 50 FPS. So, the rectangle will maximally be moved 50 times per second by exactly one pixel.

Please note: A script is called a maximum of 50 times per second.

Creating Faster Effects

Now we know that we can not increase the maximal render frequency. If we cannot render the rectangle more often, we need to move the rectangle a little bit more than one pixel per call. The solution is *framesteps*, a value which can be determined by the function:

```
float GetFrameSteps()
```

This function tells us, how many frames would have been left since the last call of our *RenderEffect* function. E.g. if we have a speed of 6000 BPM, *GetFrameSteps* results in a value of 2.0 (6000 BPM / 2.0 = 3000 BPM or 50Hz). To use this in our script, we need to change one line:

```
g_pos[0] += 1;
to
g_pos[0] += (int)GetFrameSteps();
```

In the end, the script looks like this:

```
int g_pos[];
int g_rectSize[] = {4, 4}; //size of the rect

void MatrixSizeChanged()
{
    g_pos[1] = GetMatrixHeight() / 2 - g_rectSize[1] / 2;;
}

void InitEffect()
{
    MatrixSizeChanged();
}

void RenderEffect()
{
    Clear();
    FillPixelRect(WHITE, g_pos[0], g_pos[1], g_rectSize[0], g_rectSize[1]);

    //move the rectangle
    g_pos[0] += (int)GetFrameSteps();
    //check if the rectangle has left the matrix and set it to the other side
    if(g_pos[0] > GetMatrixWidth() + g_rectSize[0])
        g_pos[0] = -g_rectSize[0];
    else if(g_pos[0] < -g_rectSize[0])
        g_pos[0] = GetMatrixWidth()+g_rectSize[0];
}
```

Explanation:

This script is now able to play much faster than 3000 BPM and you can increase the speed even more using the Speed Master. Furthermore, if you set the Speed Master to its negative range of values, it will play backwards. This is due to the fact that if the Speed Master is negative, *framesteps* is also negative. Since we are making direct use of *FrameSteps*, the position will be decreased automatically.

Due to this fact, the script has not only to check if the rectangle has left the matrix to the right side (which we have done in the other script, too). It also needs to be checked if the rectangle has left the matrix to the left side. And if so, the rectangle needs to be set to the other side. This is done with the help of the last two lines that were added to the script.

Now we have a script which is able to run very fast and which can be controlled by the Speed Master as well.

Using Floating-point Values

Up to this point, *GetFrameSteps* always delivers integer values, like 1.0, 2.0, 3.0, and so on. The minimal value was 1.0. It is also possible, and sometimes necessary, to have values inbetween, like 1.5 for example. Especially if a fixed render frequency is used, this is absolutely necessary as you will see later on. Some effects require floating point values, or else the effect will not be rendered smoothly. You can enable floating point frame steps with the function:

```
void SetUseFloatFrames(int enable)
```

Add this line to the following script and start it. Now, if you move the BPM slider you will see that *GetFrameSteps* delivers values like 2.5, etc.

```
void InitEffect()  
{  
    SetUseFloatFrames(true);  
}  
  
void RenderEffect()  
{  
    WriteText((string)GetFrameSteps());  
}
```

Note: Even if a script enables floating point frame steps, after recompiling and starting another script it is disabled again.

4.4.2 Controlling A Script Via Frame ID

Using The Frame ID

The Frame ID is a special feature that is only supported by some effects. These effects are:

- SCE Color Change
- SCE Color Scroll
- SCE Plasma
- SCE Radial
- SCE Ticker
- SCE Wave
- MAS Script Effect

Using the Frame ID makes sense when the effect generates a sequence of visuals, which will repeat itself after a while. A user of MADRIX could forward or rewind effects. Hence, the Frame ID influences the position (and/or speed) of the listed effects. As mentioned above, this can also be used for effects, which are written as a Script in the MAS Script Effect.

You will not have access to the individual Frame IDs generated by MADRIX. Instead, you can simply add values to the current Frame ID, which is unknown to you.

Example:

Let's assume the current Frame ID is 0. Add 50 frames and the new result is 50.

Let's assume further that another Frame ID is 2500. Add 50 and the new result is 2550.

In both cases, adding 50 will have the same result: the position is skipped by 50.

For our script from the last chapter, the sequence of pictures was one complete movement of the rectangle over the whole matrix. Now, imagine an effect which clears/fills the matrix with different colors defined by a color table. This effect could use the Frame ID to index the color table in order to determine which color to use next.

In order to use Frame IDs, we need to know the number of frames an effect produces that will be repeated. The next step is to tell MADRIX this number of frames. MADRIX needs this number to be able to generate the Frame IDs (including 0), but excluding the given number of those frames. The last step is to determine the current Frame ID when the effect is rendered in order to draw the appropriate picture. The number of frames we need is set and received by:

- `void SetFrameCount(float framecount)`
- `float GetFrameCount()`

The current Frame ID can be retrieved by a call of:

- `float GetFrameId()`

And here is our script using Frame ID. As you can see, it is much simpler and shorter since we do not need to check if the rectangle has left the matrix. Furthermore, the rectangle will be moved almost automatically by MADRIX.

```
int g_ypos;
int g_rectSize[] = {4, 4}; //size of the rectangle

void MatrixSizeChanged()
{
    g_ypos = GetMatrixHeight() / 2 - g_rectSize[1] / 2; //the number of frames we have
    SetFrameCount((float)(GetMatrixWidth() + 2 * g_rectSize[0]));
}

void InitEffect()
{
    MatrixSizeChanged();
}

void RenderEffect()
{
    Clear();
    FillPixelRect(WHITE, (int)GetFrameId()-rectSize[0], g_ypos, g_rectSize[0], g_rectSize[1]);
}
```

Explanation:

This script uses the Frame ID as position x for the rectangle.

First, we need to determine and set the number of frames we produce. This is done in the function *MatrixSizeChanged*. Since we want to move the rectangle over the whole matrix and the rectangle shall move in and out of it, we calculate our number of frames as:

- `frame count = width of matrix + 2 * width of rectangle`

In order to let it move into the matrix, we subtract the width of the rectangle from the Frame ID so that position x is:

- `position x = GetFrameId() - width of the cube`

This is done in the last line of the script.

Now, this script can be controlled per Frame ID. Since the calculation of the Frame ID includes the current Speed Master and the set speed, this script is able to run for- and backwards and very fast.

Setting The Frame ID

It is also possible to set a new Frame ID using the following function.

```
void SetFrameId(float id)
```

This may be useful to have more control over a running script.

Another Way To Use the Frame ID

The following script fills the matrix with the colors red, green, and blue. It uses the Frame ID to index the color table in order to determine which color should be used to fill the matrix. Simply run this script and move the Speed Master into both directions. You will be able to see that the order of the colors will change. You may see that the effect runs backwards if the Speed Master is set to negative values.

```
color g_colTab[] = {
    {255, 0, 0, 0}, //red
    { 0,255,0, 0}, //green
    { 0, 0,255,0} //blue
};

void InitEffect()
{
    SetFrameCount((float)(g_colTab.length));
    SetBpm(30);
}

void RenderEffect()
{
    Clear(g_colTab[(int)GetFrameId()]);
}
```

4.4.3 Using A Fixed Render Frequency

Up to now, we know that the maximal render frequency of an effect is 50 FPS, which translates to 3000 BPM. We also know how to manage that a script can act like it would run much faster. The last thing we want to introduce here is the possibility to have a fixed render frequency.

Imagine a script that needs a lot of processing time; much too much to render it at 50 FPS. You may want to decrease the render frequency while the user is still able to set the speed of the effect using the BPM slider or the Speed Master. We have done this before, but with the difference that MADRIX did set the maximal fixed render frequency.

We can tell MADRIX to render an effect with a fixed frame rate, for example 1500 BPM (or 25 FPS), using the function:

```
void SetFixedFrameRate(int bpm, int enable)
```

This function sets the render frequency to the given value in *bpm* if *enable* is set to *true*. If *enable* is *false*, the *bpm* value is ignored and the usage of a fixed render frequency is disabled again. The parameter *bpm* may have a value of 1 to 3000. If the render frequency is set to a fixed rate, you will need to use the function *framesteps* value to calculate the effect. If this function is not used, it will be rendered with the speed the user has set. The following two examples will explain this in more detail.

To demonstrate this effect, the first script of this chapter is used as a basis. The line *SetFixedFrameRate(1500, true)* simply needs to be added to *InitEffect*. Here is the complete source code:

```
int g_pos[];
int g_rectSize[] = {4, 4}; //size of the rect

void MatrixSizeChanged()
{
    g_pos[1] = GetMatrixHeight() / 2 - g_rectSize[1] / 2;;
}

void InitEffect()
{
    MatrixSizeChanged();
    SetFixedFrameRate(1500, true);
}

void RenderEffect()
{

```



```
Clear();
FillPixelRect(WHITE, g_pos[0], g_pos[1], g_rectSize[0], g_rectSize[1]);
//move the rectangle
g_pos[0] += 1;
//check if the rectangle has left the matrix and set it back
if(g_pos[0] > GetMatrixWidth() + g_rectSize[0])
    g_pos[0] = -g_rectSize[0];
}
```

Please load and start the script using the MAS Script effect. While it is running, try to move the BPM slider or the Speed Master and you will see that nothing happens. The rectangle moves exactly with 25 FPS from the left to the right side of the matrix. In order to get a faster or slower movement of the rectangle, two things are necessary: First, the function *framesteps* must be used to calculate the movement. Second, the position values need be changed from *integer* to *float* since the frame steps will be given as floating point values. Here is the new example:

```
float g_pos[];
int g_rectSize[] = {4, 4}; //size of the rect

void MatrixSizeChanged()
{
    g_pos[1] = (float)(GetMatrixHeight() / 2 - g_rectSize[1] / 2);
}

void InitEffect()
{
    MatrixSizeChanged();
    SetFixedFrameRate(1500, true);
}

void RenderEffect()
{
    Clear();
    FillPixelRect(WHITE, (int)g_pos[0], (int)g_pos[1], g_rectSize[0], g_rectSize[1]);

    //move the rectangle
    g_pos[0] += GetFrameSteps();
    //check if the rectangle has left the matrix and set it to the other side
    if((int)g_pos[0] > GetMatrixWidth() + g_rectSize[0])
        g_pos[0] = (float)-g_rectSize[0];
    else if((int)g_pos[0] < -g_rectSize[0])
        g_pos[0] = (float)(GetMatrixWidth()+g_rectSize[0]);
}
```

In the previous examples, *GetFrameSteps()* always resulted in *integer* values like 1.0, 2.0, etc. The minimum value was 1.0. Now the framesteps are given as floating point values and may also result in 0.5 or 1.73. *SetFixedFrameRate* requires *floating point* values to work accurately. Therefore, if *SetFixedFrameRate* is enabled, floating point frames will be enabled as well. Moreover, it is not possible to disable floating point frames when a fixed frame rate is used.

Please note: Even though a script may use a fixed render frequency, after recompiling and starting another script it will be disabled again.

Part



5 Macros For Effects

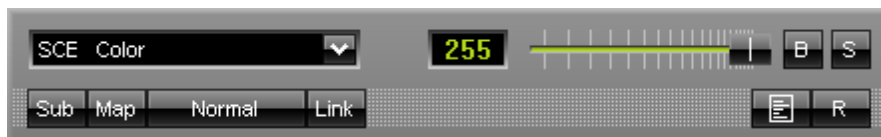
5.1 Overview

Introduction

Macros can be used to manipulate its result or to change its settings. They are bound to the particular effect. Macros are written in MADRIX Script. The main difference between a macro and a script of the MAS Script effect are **the functions MADRIX calls**. In addition, each MADRIX effect has its own functions that can be called from a macro to control it.

Macros are stored as a part of the effect. This means they are part of a stored effect or setup. Moreover, it is possible to save macros as separate files. The file extension of a macro is **.mms*. The extension of a compiled macro is **.mcm*.

Macros are used to manipulate the result of an effect. A macro is called before and after an effect has been rendered.



To run a macro for an effect, please select the **'Macro'** button in the effect area.

Functions Called By MADRIX

The following functions are called by MADRIX for each effect and can be implemented by a macro in order to react to different events:

- `void InitEffect()`
- `void PreRenderEffect`
- `void PostRenderEffect()`

- `void MatrixSizeChanged()`

InitEffect

(automatically included in a new macro)

InitEffect is called by MADRIX whenever the script needs to be initialized. This is the case after compiling and starting a new macro or when the user pressed the **"Start"** button of the **Script Editor**. A macro can assume that any global variable is initialized with 0 and that any global field is empty as long as it has not been initialized with any value.

PreRenderEffect

(automatically included in a new macro)

PreRenderEffect is called before the effect is going to be rendered. Changes done here affect the current frame, but may be overwritten by the effect itself. In most times, this function may be used to change the mapping of an effect, for example to move it around the matrix.

PostRenderEffect

(automatically included in a new macro)

This function is called immediately after the effect has been rendered. Here, the result of the effect can be manipulated. You could use a grey filter on it, for example.

Please note: The matrix, which the macro manipulates, is the same matrix that the effect uses to calculate its own effect. The effect may rely on the output being the input for the next frame with undefined behavior.

Please also note: Mapping operations done in *PostRenderEffect* will effect the next frame, but not the current one. To control the current frame, please use *PreRenderEffect*.

MatrixSizeChanged

(automatically included in a new macro)

MatixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings or because a new map setting was set, e.g. caused by the call of a map function.

Standard Outline

When you open the Effect Macro Editor, the empty standard macro will look like this:

```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

void MatrixSizeChanged()
{
    InitEffect();
}
```

5.2 Functions

Specific Resources

- [Functions called by MADRIX](#)
- [Macros for Effects: Available Functions](#)
- [Setting and Getting the Current Speed](#)
- [Using the Frame ID](#)
- [SCE - Static Light Effects](#)
- [S2L - Sound2Light Effects](#)
- [M2L - Music2Light](#)

General Resources

- [Keyword Search](#)
- [List Of Functions \(Alphabetical Order\)](#)
- [List Of Functions \(Grouped\)](#)
- [List Of Global Variables and Constants](#)
- [List Of Operations](#)
- [List Of Structures](#)
- [Table Of Frequencies](#)
- [Table Of Notes](#)

Available Functions

Standard Functions

It is useful to consult the [List of Functions \(Alphabetical Order\)](#) for non-specific functions.

Please note: The functions *GetSpeed* and *SetSpeed* are not available for every single effect.

Effect-Specific Functions Depending On The Effect

In addition to the standard functions of MADRIX Script, there are other functions available that are special to each effect.

Because of the quantity of different types of effects in MADRIX, each one has its own settings. Therefore, for each effect there are special commands available. Those commands are not working in other effects and a macro which uses such commands cannot be compiled in another effect. An example is the function *SetText* for the SCE Ticker effect. This function specifies the text of the Ticker. Since no other effect shows text, the function is not available for other effects.

The functions available for the each different effect and their meaning is described in the chapters below. There is one chapter for each effect ([SCE Static Color Effects](#), [S2L Sound2Light Effects](#), [M2L Music2Light Effects](#)).

Setting And Getting The Current Speed Of An Effect

Several effects allow to set their speed using the BPM slider. Examples for such effects include SCE Plasma or the S2L EQ effect. Until version 2.5 of MADRIX those effects had a speed-slider working with Hz and a range between 1 to 50 Hz. Since version 2.5 of MADRIX those effects have a BPM slider setting the speed of an effect using beats per minute with a range from 0 to 9999 BPM.

The macros of those effects, which support setting the speed, now have two additional functions:

- `void SetBpm(int bpm)`
- `int GetBpm()`

Setting BPM to 0 means that the effect will be stopped immediately. Furthermore, values of the function *SetSpeed* now range from 0 to 166, which represents a range of 0 to almost 9999 BPM.

Using The Frame ID

There are a lot of effects which use an internal frame ID to have more control. For example, the SCE Color Scroll effect uses the frame ID to control the speed of scrolling. This does not make sense for all effects. However, for each effect it is possible to set and get the current frame ID and frame count. The frame count identifies the maximal frame ID used by the effect. This may be utilized to speed up the effect. For example, the following source code increases the speed of an effect that is controlled by the frame ID.

```
void PreRenderEffect()  
{  
    SetFrameId(GetFrameId() + 2);  
}
```

However, it is not possible to set the frame count by a macro; except for a macro of the MAS Script effect.

Applying Filter Effects (FX)

You may quickly change the visual outcome of Effects using Filters (also called Filter Effects, FX).

- `void SetFilter(int filter)`
- `int GetFilter()`

Valid values for filter are the global variables »[Filters](#) . You can also find further explanations about the various filter types in this chapter.

Setting Blind Mode Or Solo Mode

- `void SetSolo(int)`
- `int GetSolo()`
- `void SetBlind(int)`
- `int GetBlind()`

Blind mode and solo mode are two options available for each layer. Blind mode will deactivate the current layer, while solo mode will only show this layer while disabling all other layers.

Valid values for *int* are *0 (Off)* or *1 (On)*.

5.3 SCE Static Color Effects

5.3.1 SCE Color

For the SCE Color effect it is possible to set the speed and the color to display.

Functions Provided By SCE Color

Getting And Setting The Speed

The speed is to be set in BPM with a range of 0 to 9999 BPM. The following functions are available:

- `void SetBpm(int bpm)`
- `int GetBpm()`

If *bpm* is lower than 0, it is reduced to 0 which means that the effect will be stopped. If *bpm* is higher than 9999, it is lowered to 9999.

Getting And Setting The Speed (Deprecated Functions)

The speed of the effect can be set in the range of 0 to 166 frames per seconds. The following functions are available:

- `void SetSpeed(int fps)`
- `int GetSpeed()`

If *FPS* is lower than 0 or greater than 166, then a warning is written to the Script Editor output. Additionally, it will be automatically to 0/166 FPS which represents a range of 0 to almost 9999 BPM.

Please note: These functions are deprecated and may be removed in some forthcoming release of MADRIX. Please use [SetBpm](#) and [GetBpm](#) instead. Deprecated functions are outdated functions and should not be used anymore.

Setting And Getting The Color

It is possible to set the color the effect uses with the following function:

- `void SetColor(color col)`
- `color GetColor()`

This Effect uses the Color Picker. Learn more about [Using Colors](#).

5.3.2 SCE Bitmap

Functions Provided By SCE Bitmap

The following table provides an overview of all functions the effect can use:

Function	Description
void SetDirection (int)	Sets the movement direction. Allowed are all directions described by the DIR_ constants . Use DIR_NONE , to stop the image movement.
int GetDirection ()	Retrieves the current movement direction. Returns DIR_NONE , if the movement has been stopped.
int GetPixelImagePositionX ()	Returns the current absolute x - pixel coordinate of the image.
int GetPixelImagePositionY ()	Returns the current absolute y - pixel coordinate of the image.
float GetVectorImagePositionX ()	Returns the current absolute x - coordinate of the image.
float GetVectorImagePositionY ()	Returns the current absolute y - coordinate of the image.
void SetPixelImagePosition (int x, int y)	Sets the position of the image with absolute pixel coordinates.
void SetVectorImagePosition (float x, float y)	Sets the position of the image using relative coordinates.
int GetPixelImageWidth ()	Returns the width of the currently displayed image as absolute pixel value.
int GetPixelImageHeight ()	Returns the height of the currently displayed image as absolute pixel value.
float GetVectorImageWidth ()	Returns the width of the currently displayed image relative to the settings of the current matrix.
float GetVectorImageHeight ()	Returns the height of the currently displayed image relative to the settings of the current matrix.
void SetStretch (int enable)	If <i>enable</i> is <i>false</i> , stretching will be disabled, otherwise it will be enabled.
int GetStretch ()	Returns <i>true</i> , if stretch is currently enabled, otherwise <i>false</i> .
void SetTile (int enable)	Disables tile mode of the bitmap effect if <i>enable</i> is <i>false</i> , otherwise the tile mode will be enabled.
int GetTile ()	Returns <i>true</i> if tile mode of the bitmap effect is currently enabled, otherwise <i>false</i> .
void SetGrey (int enable)	Disables grayscale mode if <i>enable</i> is <i>false</i> , otherwise it will be enabled.
int GetGrey ()	Returns <i>true</i> if grayscale mode is currently active, otherwise <i>false</i> .
void SetRgbToRgbw (int enable)	Disables the RGB-to-RGBW mode if <i>enable</i> is <i>false</i> , otherwise it will be enabled.
int GetRgbToRgbw ()	Returns <i>true</i> if RGB-to-RGBW mode is enabled, otherwise <i>false</i> .
void SetBpm (int bpm)	Sets the speed for the image movement. The value <i>bpm</i> must be within a range of 0 to 9999 BPM. E.g. a value of 60 means that the image moves one pixel per second into the given direction.

int GetBpm()	Returns the speed for the image movement in BPM.
void SetAnimationSpeed (float multiplier)	Sets the animation speed multiplier. See below for further details.
float GetAnimationSpeed ()	Returns the current animation speed multiplier. See below for further details.
void SetFilterColor (color col)	Sets the filter color for the effect. Please note that the alpha value of the color structure is not used by this effect.
color GetFilterColor ()	Returns the current filter color. Please note that the alpha value is not used and should be ignored.
void SetRotation (int angle)	Sets the rotation globally for all images of the image list. It is possible to rotate the images by multiples of 90°. This function is equal to the rotation button provided by the GUI of the effect. Valid values for <i>angle</i> are 0, 90, 180, and 270.
int GetRotation ()	Returns the current rotation which is set on the effect itself.
int GetImageCount ()	Returns the number of currently loaded images in the image table.
void SetCurrentImage (int idx)	Sets the image that should be displayed next. If <i>idx</i> is out of range, nothing happens.
int GetCurrentImage ()	Returns the index of the currently displayed image.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetMoveSpeed (float speed)	Sets the speed for the image movement. The speed must be given in frames per second. E.g. 2 means that the image will be moved 2 pixels per second. The available range can be set from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetMoveSpeed ()	Returns the speed for the image movement in frames per second. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Speed Settings Of The Bitmap Effect

The bitmap effect has two independent speed settings implemented. The first one is the movement speed of the picture moving over the matrix. The second one is the animation speed used to speed up or slow down the animation.

The following functions specify and retrieve the speed for the picture movement in frames per second:

- `void SetBpm(int bpm)`
- `int GetBpm()`

The next two functions control the animation speed:

- `void SetAnimationSpeed(float multiplier)`
- `float GetAnimationSpeed()`

The functions do not control the animation speed itself, but work with a multiplier. This means that if *multiplier* is set to 2 for example, the animation speed is doubled. A *multiplier* of 0.5 is setting the animation speed to half of the original one.

(Description)

Examples

Moving An Image

The following example moves the image or animation back and forth on the matrix.

```
void InitEffect()
{
}

void PreRenderEffect()
{
    if(GetDirection() == DIR_LEFT)
    {
        if(GetVectorImagePositionX() <= 0.0)
        {
            SetVectorImagePosition(0.0, GetVectorImagePositionY());
            SetDirection(DIR_RIGHT);
        }
    }
    else if(GetDirection() == DIR_RIGHT)
    {
        if(GetVectorImagePositionX() + GetVectorImageWidth() >= 1.0)
        {
            SetVectorImagePosition(1.0 - GetVectorImageWidth(), GetVectorImagePositionY());
            SetDirection(DIR_LEFT);
        }
    }
    else
    {
        SetDirection(DIR_LEFT);
    }
}

void PostRenderEffect()
{
}
```

Controlling The Animation

The following example stops the animation and selects the images to show.

```
@scriptname=" ";
@author=" ";
@version=" ";
@description=" ";

int g_img;
```

```

void InitEffect()
{
    WriteText(GetAnimationSpeed());
    SetAnimationSpeed(0.0);
}

void PreRenderEffect()
{
    SetCurrentImage(g_img);
    g_img = (g_img + 1) % GetImageCount();
}

void PostRenderEffect()
{
}

```

5.3.3 SCE Bounce

Functions Provided By SCE Bounce

The following table provides an overview over all functions the effect provides:

Function	Description
void SetBpm (int bpm)	Sets the speed for the movement. <i>bpm</i> must be within a range of 0 to 9999 BPM. E.g. a value of 60 means that it is moved one pixel per second into the given direction.
int GetBpm ()	Returns the current speed for the movement in BPM.
void SetObjects (int count)	Sets the number of objects.
int GetObjects ()	Returns the current number of objects.
void SetPoints (int count)	Sets the number of points per object. Only line and curve mode allow more than 1 point per object.
int GetPoints ()	Returns the current number of points per object.
void SetFadeOut (int value)	Sets the fade out value.
int GetFadeOut ()	Returns the current fade out value.
void SetSize (int size)	Sets the size of the shapes. This only works if points per object is set to 1.
int GetSize ()	Returns the current size of the shapes.
void SetShape (int shape)	Sets the shape to use. One of the defines described below must be used.
int GetShape ()	Returns the currently defined shape. See below for further details
void SetCollision (int state)	Sets the collision mode to on or off. A value of 1 stands for on and 0 for off.
int GetCollision ()	Returns the currently used collision mode.

void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color of the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Draw Modes

This effect uses various geometrical figures. With the function [SetShape](#) it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_CROSS	Selects the cross for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
DRAW_RECT	Selects the unfilled rectangle for drawing.
DRAW_STAR	Selects the star for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_LINE	Selects the line mode for drawing.
DRAW_CURVE	Selects the curve mode for drawing.
DRAW_DIAMOND	Selects the diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.

5.3.4 SCE Color Change

Functions Provided By SCE Color Change

The following table provides an overview of all functions the effect can use:

Function	Description
void SetChangeTime (float time)	Sets the fade time for the effect in seconds.
float GetChangeTime ()	Retrieves the fade time of the effect in seconds.
void SetBpm (int bpm)	Sets the speed of the color change in BPM. <i>bpm</i> is in the range of 0 to 9999.
int GetBpm ()	Gets the speed of the effect in BPM.
void SetFade (int enable)	Disables fade if <i>enable</i> is false, otherwise it will be enabled. Please note that this does not affect the effect's frame fade.
int GetFade ()	Returns <i>true</i> if fade is enabled, otherwise <i>false</i> .
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color of the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 2 entries.

This Effect uses the Color Table. Learn more about [Using Colors](#).

5.3.5 SCE Color Fill

The SCE Color Fill effect provides some functions to set the fill mode, the fill time, the delay time, and the mix color setting. Settings for the different fill modes cannot be set, yet.

Functions Provided By SCE Color Fill

The following table provides an overview of all functions the effect can use:

Function	Description
----------	-------------

void SetMixColor (int enable)	Disables the mix color mode if <i>enable</i> is <i>false</i> , otherwise it will be enabled.
int GetMixColor ()	Returns <i>true</i> if the mix color mode is enabled.
void SetFillMode (int mode)	Sets the fill mode. The table below describes the valid values for parameter <i>mode</i> .
int GetFillMode ()	Returns the current fill mode. See the table below for the meaning of the return values.
void SetFillTime (float filltime)	Sets the fill time in seconds. This is the time in which the effect has to fill the whole matrix with pixels.
float GetFillTime ()	Returns the fill time in seconds.
void SetDuration (float delaytime)	Sets the duration in seconds. This is the time the effect waits after it has filled the whole matrix.
float GetDuration ()	Returns the duration in seconds.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 2 entries.
void SetWidth (int size)	Sets the size of objects for the fill mode Random, Drops, Snake, and Tetris. Valid values range from 1 to double the size of the virtual matrix.
int GetWidth ()	Returns the size of objects for the fill mode Random, Drops, Snake, and Tetris.
void SetPitch (int pitch)	Sets the distance of objects for the fill mode Random, and Drops. Valid values range from 1 to double the size of the virtual matrix.
int GetPitch ()	Returns the distance of objects for the fill mode Random, and Drops.
void SetShape (int shape)	Sets the shape of objects for the fill mode Random, Drops, and Tetris. Applicable values can be found in the table below .
int GetShape ()	Returns the shape of objects for the fill mode Drops.
void SetMirror (int value)	Activates or deactivates mirror mode for the fill mode Snake. Applicable values are 0 (off) and 1 (on).
int GetMirror ()	Returns if mirror mode is activated or not.
void SetCircle (int value)	Activates or deactivates circle mode for the fill mode Snake. Applicable values are 0 (off) and 1 (on).
int GetCircle ()	Returns if circle mode is activated or not.
void SetCenter (int value)	Activates or deactivates center mode for the fill mode Snake. Applicable values are 0 (off) and 1 (on).

int GetCenter()	Returns if center mode is activated or not.
------------------------	---

This Effect uses the Color Table. Learn more about [Using Colors](#).

Setting The Fill Mode

By using the function *SetFillMode* it is possible to set the mode the effect uses to fill the matrix. The following constants must be used to set the different fill modes:

Value	Description
MODE_RANDOM	Sets the Random fill mode.
MODE_DROPS	Sets the Drops fill mode.
MODE_SNAKE	Sets the Snake fill mode.
MODE_FLAT	Sets the Flat fill mode.
MODE_COLLAPSE	Sets the Collapse fill mode.
MODE_TETRIS	Sets the Tetris fill mode.

Shapes for Fill Mode Drops

The fill mode Drops uses various geometrical figures. With the function *SetShape* it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_CROSS	Selects the cross for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_LINE	Selects the line mode for drawing.
DRAW_DIAMOND	Selects the unfilled diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

Full Example

The following example changes the fill mode after the matrix has been filled up completely.

```
int g_startTime;

void InitEffect()
{
    time t = GetTime();
    g_startTime = t.hour * 3600 + t.min * 60 + t.sec;
}

void PreRenderEffect()
{
    time t = GetTime();
    int t2 = t.hour * 3600 + t.min * 60 + t.sec;

    if(t2 - g_startTime > (int)GetFillTime())
    {
        g_startTime = t2;
        if(GetFillMode() == MODE_RANDOM)
            SetFillMode(MODE_DROPS);
        else if(GetFillMode() == MODE_DROPS)
            SetFillMode(MODE_SNAKE);
        else if(GetFillMode() == MODE_SNAKE)
            SetFillMode(MODE_FLAT);
        else if(GetFillMode() == MODE_FLAT)
```

```

        SetFillMode(MODE_COLLAPSE);
    else if(GetFillMode() == MODE_COLLAPSE)
        SetFillMode(MODE_TETRIS);
    else {
        SetFillMode(MODE_DROPS);
    }
}

void PostRenderEffect()
{
    if(GetFillTime() != 10)
        SetFillTime(10);
}

```

Explanation:

Since the effect cannot detect when the matrix has been filled up, it uses the fill time and changes the fill mode after the *fill time* has passed. In the *PostRenderEffect* function the time is corrected and set to 10, because each fill mode sets its own default time when activated.

5.3.6 SCE Color Ramp

Functions Provided By SCE Color Ramp

The following table provides an overview of all functions the effect can use:

Function	Description
void SetDirection (int direction)	Sets the direction of the Color Ramp. Please use a define as described below for <i>direction</i> .
int GetDirection ()	Returns the direction currently in use.
void SetCrossed (int enable)	Use <i>true</i> or <i>1</i> for <i>enable</i> to activate Cross Mode for the Direction. Use <i>false</i> or <i>0</i> to deactivate it.
int GetCrossed ()	Returns <i>true</i> if "Cross Mode" is enabled, otherwise <i>false</i> .
void SetAngle (int angle)	Sets the angle for the Color Ramp, type Radial.
int GetAngle ()	Gets the current angle.
void SetColor (int index, color c)	Sets the <i>color c</i> at the specified <i>index</i> in the Color Ramp. If <i>index</i> is out of range, nothing happens.
color GetColor (int index)	Returns the <i>color</i> with the specified <i>index</i> in the Color Ramp. If the <i>index</i> is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the amount of colors currently used by the Color Ramp.

void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the <i>index</i> is lower or equal to 0, the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are <i>1</i> (On) or <i>0</i> (Off).
void RemoveColor (int index)	Removes the color at the specified <i>index</i> . If the given <i>index</i> is out of range, nothing happens.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the color position of the given <i>index</i> .
void SetColorFade (int index, fade)	Sets the color fade option for the given <i>index</i> . Valid values for <i>fade</i> are <i>1</i> (On) or <i>0</i> (Off).
int GetColorFade (int index)	Returns the color fade option for the given <i>index</i> .
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.

This Effect uses the Color Ramp. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
int SetPosition (int index, float pos)	Use SetColorPosition() instead.
float GetPosition (int index)	Use GetColorPosition() instead.
void SetFade (int index)	Use SetColorFade() instead.
int GetFade (int index)	Use GetColorFade() instead.

Directions Of The Color Ramp

In addition to the [standard directions](#), there are five additional directions available:

Value / Define	Description
DIR_OUTWARDS	Sets the Color Ramp type to 'Box Ramp Explode'.
DIR_INWARDS	Sets the Color Ramp type to 'Box Ramp Implode'.
DIR_CIRCLE_OUTWARDS	Sets the Color Ramp type to 'Circular Ramp Explode'.
DIR_CIRCLE_INWARDS	Sets the Color Ramp type to 'Circular Ramp Implode'.
DIR_RADIAL	Sets the Color Ramp type to 'Radial Ramp'.

5.3.7 SCE Color Scroll

Functions Provided By SCE Color Scroll

The following table provides an overview of all functions the effect can use:

Function	Description
void SetDirection (int direction)	Sets the scroll direction. Allowed are all DIR_ constants for directions and there are some additional directions available, which are described below .
int GetDirection ()	Returns the current scroll direction.
void SetCrossed (int enable)	Disables "cross mode" if <i>enable</i> is <i>false</i> , otherwise it will be enabled.
int GetCrossed ()	Returns <i>true</i> if "cross mode" is enabled, otherwise <i>false</i> .
void SetBpm (int bpm)	Sets the <i>bpm</i> value of the effect. Valid values range from 0 to 9999.
int GetBpm ()	Returns the currently set BPM value.
void SetPixelStepWidth (int width)	Sets the step value, which is the number of pixels the effect should scroll per frame using absolute values.
int GetPixelStepWidth ()	Returns the step value.
void SetVectorStepWidth (float width)	Sets the step value, which is the number of pixels the effect should scroll per frame using relative values. The given value is relative to the current matrix size and the current direction. See below for further details.
float GetVectorStepWidth ()	Returns the step value relative to the matrix size and scroll direction. See below for further details.
void SetPixelOffset (int offset)	Sets the offset in pixels, which means the width of the single stripes.
int GetPixelOffset ()	Returns the offset value, the width of a stripe, in pixels.

void SetVectorOffset (float offset)	Sets the offset value relative to the matrix size and direction. See below for further details.
float GetVectorOffset ()	Returns the offset relative to the matrix size and direction. See below for further details.
void SetRandom (int enable)	Enables or disables random mode.
int GetRandom ()	Returns true if random mode is enabled, otherwise false.
void SetPixelRandValues (int min, int max)	Sets the random values in pixels.
int GetPixelMinRandValue ()	Returns the minimum random value in pixels.
int GetPixelMaxRandValue ()	Returns the maximum random value in pixels.
void SetVectorRandValues (float min, float max)	Sets the random values as values relative to the current matrix size and direction. See below for details.
float GetVectorMinRandValue ()	Returns the minimum random value as value relative to the matrix size and direction. See below for details.
float GetVectorMaxRandValue ()	Returns the maximum random value as value relative to the matrix size and direction. See below for details.
void SetPixelXOffset (int offset)	Sets the X-Offset in pixels.
int GetPixelXOffset ()	Returns the current X-Offset in pixels.
void SetVectorXOffset (float offset)	Sets the X-Offset relative to the matrix size and direction. See below for details.
float GetVectorXOffset ()	Retrieves the X-Offset relative to the matrix size and direction. See below for details.
void SetFade (int enable)	Enables the interpolation between two colors. If <i>enable</i> is <i>true</i> , interpolation will be enabled, otherwise disabled.
int GetFade ()	Returns whether interpolation between two colors is enabled, or not.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the current number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 2 entries.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Using Relative Values

The relative values mentioned above, e.g. for *SetVectorStepWidth*, relate to the current direction and matrix size. For vertical and horizontal scroll directions, it is simple to understand. For horizontal (left, right) movements, the values relate to the matrix width and for vertical movements (up, down), the values relate to the matrix height.

For diagonal movements, e.g. to the upper left corner, the values relate to the higher value of either matrix width or height. For example, the size of the matrix is 40 pixels in width and 55 pixels in its height. Then, given values relate to the height with 55 pixels. This is due to the fact that it fits better than using the extent of the matrix diagonal.

Additional Scroll Directions

As you can see, this effect provides additional directions. Regarding vertical and horizontal movements, this refers to outside movement to the center or and movement from the center to the outside. They can be set with the function *SetDirection* using the following constants:

Value	Description
DIR_OUTWARDS_V	Sets a vertical outward movement (Vertical Explode).
DIR_INWARDS_V	Sets a vertical inward movement (Vertical Implode).
DIR_OUTWARDS_H	Sets a horizontal outward movement (Horizontal Explode).
DIR_INWARDS_H	Sets a horizontal inward movement (Horizontal Implode).
DIR_OUTWARDS_RECT	Sets a rectangular outward movement (Rectangle Explode).
DIR_INWARDS_RECT	Sets a rectangular inward movement (Rectangle Implode).
DIR_OUTWARDS_SQUARE	Sets a quadratic outward movement (Square Explode).
DIR_INWARDS_SQUARE	Sets a quadratic inward movement (Square Implode).
DIR_OUTWARDS_CIRCLE	Sets a circled outward movement (Circle Explode).
DIR_INWARDS_CIRCLE	Sets a circled inward movement (Circle Implode).
DIR_OUTWARDS_ELLIPSE	Sets an elliptic outward movement (Ellipse Explode). This is different to a circle on rectangular matrices. On quadratic matrices the ellipse will also be a circle.
DIR_INWARDS_ELLIPSE	Sets an elliptic inward movement (Ellipse Implode). This is different to a circle on rectangular matrices. On quadratic matrices the ellipse will also be a circle.

5.3.8 SCE Drops

Functions Provided By SCE Drops

The following table provides an overview of all functions the effect can use:

Function	Description
void SetDirection (int dir)	Set the direction of the effect. Valid values are <i>DIR_UP</i> , <i>DIR_DOWN</i> , <i>DIR_LEFT</i> and <i>DIR_RIGHT</i> .
int GetDirection ()	Returns the current direction.

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetStep (int step)	Sets the step value, which stretches the drops. For example, a step value of 2 will render drops every second pixel. Valid values for <i>step</i> range from 1 to 100.
int GetStep ()	Returns the currently set step value.
void SetLength (int length)	Sets the length of the objects. Valid values range from 1 to 100.
int GetLength ()	Returns the current object length.
void SetCount (int count)	Sets the number of shapes which shall be created. Valid values range from 0 to 100.
int GetCount ()	Returns the number of shapes currently used.
void SetWidth (int width)	Sets the size of the objects in pixels. Valid values range from 1 to 100.
int GetWidth ()	Returns the size of the objects in pixels.
void SetPitch (int pitch)	Sets the pitch between two shapes in pixels. Valid values range from 1 to 100.
int GetPitch ()	Returns the currently set pitch in pixels.
void SetColorMode (int mode)	Sets the effect color mode. See the color mode table below for further details.
int GetColorMode ()	Returns the current color mode. See below for further details.
void SetShape (int mode)	Sets the shape the effect should use. See below for further details.
int GetShape ()	Returns the currently used shape. See below for further details.
void SetColor (int idx, color c)	Sets the color with the specified index to the given color value. If the index is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the current number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Color Modes

This effect uses various color modes. With the function *SetColorMode* it is possible to change them. The following values can be used as parameter:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

Shapes

This effect uses various geometrical figures. With the function *SetShape* it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_CROSS	Selects the cross for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_LINE	Selects the line mode for drawing.
DRAW_DIAMOND	Selects the unfilled diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

5.3.9 SCE Explosions

Functions Provided By SCE Explosions

The following table provides an overview of all functions the effect can use:

Function	Description
void SetDirection (int dir)	Set the direction of the effect. Valid values are DIR_UP , DIR_DOWN , DIR_LEFT and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999. The default value is 3000.
int GetBpm ()	Returns the current speed of the effect in BPM.
void SetExplosionSize (int size)	Sets the size of a single explosion. Valid values for <i>size</i> range from 1 to 1000. The default value is 400.
int GetExplosionSize ()	Returns the current size of a single explosion.
void SetShapeSize (int size)	Sets the size of single objects. Valid values for <i>size</i> range from 1 to 100. The default value is 8.
int GetShapeSize ()	Returns the current size of single objects.
void SetGravity (float gravity)	Sets the gravity value for the effect. Valid values range from 0.01 to 99.99. The default value is 0.25.
float GetGravity ()	Returns the currently set gravity.
void SetFadeOut (int fadeout)	Sets the size of a fade out tail. Valid values range from 0 to 1000. The default value is 20.
int GetFadeOut ()	Returns the currently set fade out value.
void SetShapeCount (int count)	Sets the number of objects within an explosion. Valid values range from 1 to 1000. The default value is 75.
int GetShapeCount ()	Returns the currently set number of objects of an explosion.
void SetRocketCount (int count)	Sets the number of rockets or explosions displayed by the effect at the same time in Fireworks mode and Explosion mode. Valid values range from 0 to 100. The default value is 1.
int GetRocketCount ()	Returns the currently set number of rockets or explosions.
void SetBlur (int enable)	Activates Blur mode for the effect. Valid values are 0 (off) or 1 (on).
int GetBlur ()	Returns the status of Blur mode.
void FireRocket (int posX, int posY, int explPosX, int explPosY, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates a firework. <i>posX</i> is the X-coordinate, <i>posY</i> is the Y-coordinate, <i>explPosX</i> is the end-coordinate of the fireworks explosion in X, <i>explPosY</i> is the end-coordinate of the fireworks explosion in Y, <i>ParticleCtn</i> is the number of objects, <i>explSize</i> is the size of the fireworks explosion, <i>explShape</i> is the type of the fireworks explosion, <i>drawShape</i> is the shape of the objects, <i>Col</i> is the color of the effect, <i>sparkleCol</i> is the color of the sparkle part.

void Detonate (int explPosX, int explPosY, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates an explosion. <i>explPosX</i> is the end-coordinate of the explosion in X, <i>explPosY</i> is the end-coordinate of the explosion in Y, <i>ParticleCtn</i> is the number of objects, <i>explSize</i> is the size of the explosion, <i>explShape</i> is the type of the explosion, <i>drawShape</i> is the shape of the objects, <i>Col</i> is the color of the effect, <i>sparkleCol</i> is the color of the sparkle part.
void SetExplosionMode (int mode)	Sets the explosion mode. See below for details.
int GetExplosionMode ()	Returns which effect mode is set.
void SetExplosionShape (int shape)	Sets the type of an explosion. See below for details.
int GetExplosionShape ()	Returns which type of explosion is selected.
void SetDrawShape (int shape)	Sets the type of objects of an explosion. See below for details.
int GetDrawShape ()	Returns which object shape is set.
void SetColorMode (int mode)	Sets the type of color mode of the Color Table. See below for details.
int GetColorMode ()	Returns which color mode is set for the Color Table.
int GetColorCount ()	Returns the number of colors in the Color Table.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.
void SetColor (int idx, color c)	Sets the color values for a certain, indexed color in the Color Table.
color GetColor (int idx)	Returns which color is set for a certain color at a certain position (index) in the Color Table.
void AddColor (int idx, color c)	Adds color <i>c</i> at position <i>idx</i> to the Color Table.
void SetSparkleColorMode (int mode)	Sets the type of color mode of the Color Table for the Sparkle. See below for details.
int GetSparkleColorMode ()	Returns which color mode is set for the Color Table of the Sparkle.
int GetSparkleColorCount ()	Returns the number of colors in the Color Table of the Sparkle.
void RemoveSparkleColor (int)	Removes a color from the Sparkle Color Table.
void SetSparkleColor (int idx, color c)	Sets the color values for a certain, indexed color in the Sparkle Color Table.
color GetSparkleColor (int idx)	Returns which color is set for a certain color at a certain position (index) in the Sparkle Color Table.
void AddSparkleColor (int idx, color c)	Adds color <i>c</i> at position <i>idx</i> to the Sparkle Color Table.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Explosion Modes

This effect uses various color modes. The function *SetExplosionMode* can be used to change them. The following values can be used as parameter:

Value	Description
MODE_EXPLOSIONS	Sets the Explosion mode.
MODE_FIREWORKS	Sets the Fireworks mode.

Explosion Shapes

This effect uses various color modes. The function *SetExplosionShape* can be used to change them. The following values can be used as parameter:

Value	Description
EXPLOSION_SHAPE_SPHERE	Sets the sphere type of explosion.
EXPLOSION_SHAPE_SPHERE_GLOW	Sets the glowing sphere type of explosion.
EXPLOSION_SHAPE_SPIRAL	Sets the spiral type of explosion.
EXPLOSION_SHAPE_RADIAL	Sets the radial type of explosion.
EXPLOSION_SHAPE_DIAMOND	Sets the diamond type of explosion.
EXPLOSION_SHAPE_STAR	Sets the star type of explosion.
EXPLOSION_SHAPE_RANDOM	Sets a random type of explosion.

Shapes

This effect uses various geometrical figures. The function *SetDrawShape* can be used to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_CROSS	Selects the cross for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_DIAMOND	Selects the unfilled diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

Color Modes

This effect uses various color modes. The function *SetColorMode* can be used to change them. The following values can be used as parameter:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

5.3.10 SCE Fire

Functions Provided By SCE Fire

The following table provides an overview of all functions the effect can use:

Function	Description
General	
void SetBpm (int bpm)	Set the speed of the effect in BPM. Valid values range from 0 to 9999.
int GetBpm ()	Returns the current effect speed in BPM.
void SetDirection (int direction)	Sets the direction of the fire. Valid values for <i>direction</i> are DIR_LEFT , DIR_RIGHT , DIR_UP , and DIR_DOWN .
int GetDirection ()	Returns the current direction.
void SetHeight (int height)	Sets the height of the fire. Valid values are include 1 up to 32.
int GetHeight ()	Returns the current height.
void SetMode (mode)	Sets the mode of the effect. Valid values for <i>mode</i> are described in the table below .
int GetMode ()	Returns the currently selected mode. See table below for returned values.
Mode "Fire"	
void SetColorMode (int mode)	Sets the color mode of the fire. Valid values for <i>mode</i> are described in the table below .
int GetColorMode ()	Returns the current color mode. See table below for returned values.
Mode "Flames"	
void SetFlameSize (int size)	Sets the size of the flames. Valid values for <i>size</i> range from 1 to 100.
int GetFlameSize ()	Returns the size of the flames.
void SetIntensity (int intensity)	Sets the intensity of the flames. Valid values for <i>intensity</i> range from 1 to 100.
int GetIntensity ()	Returns the currently used intensity for the flames.
void SetColor (int index, color c)	Sets the color at the specified index in the Color Ramp dialog. If <i>index</i> is out of range, nothing happens.
color GetColor (int index)	Returns the color with the specified index in the Color Ramp dialog. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors used in the Color Ramp.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from 0.01 to 0.99. The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the position of a color at the specified <i>index</i> .

void SetColorFade (int index, int mode)	Disables or enables color fade for the color at the specified <i>index</i> position by either using <i>0</i> or <i>1</i> for <i>mode</i> .
int GetColorFade (int index)	Returns if color fade is set for the color at the specified <i>index</i> .
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.
void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the <i>index</i> is lower or equal to <i>0</i> , the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are <i>1</i> (On) or <i>0</i> (Off).
void RemoveColor (int index)	Removes the color at the specified index. If the given index is out of range, nothing happens.

This Effect uses the Color Ramp dialog. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Color Modes

The SCE Fire effect supports different predefined color modes in the "Fire" mode. It is possible to set them within a macro using the **SetColorMode** function. Therefore the following values are defined and should be used as parameter:

Value	Description
COLOR_RG	Selects the red-green color mode.
COLOR_RB	Selects the red-blue color mode.
COLOR_GR	Selects the green-red color mode.
COLOR_GB	Selects the green-blue color mode.
COLOR_BR	Selects the blue-red color mode.
COLOR_BG	Selects the blue-green color mode.

Effect Modes

The SCE Fire effect supports two different modes: "Fire" and "Flames". It is possible to set them within a macro using the *SetMode* function. Therefore the following values are defined and should be used as parameter:

Value	Description
MODE_FIRE	Selects the fire mode.
MODE_FLAMES	Selects the flames mode.

Full Example

The following example combines functions of the fire effect with sound data analysis.

```
float new_width = 0.2;
float new_fact;
float old_fact;
int mh;

void InitEffect()
{
    SetBpm(400);
    SetDirection(DIR_UP);
    mh = GetMatrixHeight();
    SetColorMode(COLOR_RG);
}

void MatrixSizeChanged()
{
    InitEffect();
}

void PreRenderEffect()
{
    new_fact = ((float)(GetSoundLevel(0) + GetSoundLevel(1))) / 800.0 + 0.4;
    new_fact = new_fact * new_width + old_fact * (1.0 - new_width);
    SetHeight((int)((float)mh * new_fact));
    old_fact = new_fact;
}
```

Explanation:

The source code sets the height of the fire depending on the sound level. Furthermore, it stores the current matrix height in a global variable and therefore uses *MatrixSizeChanged* in order to set the variable equal to the matrix size. Works best with a matrix size of 50 x 50 pixels.

5.3.11 SCE Graph

Functions Provided By SCE Graph

The following table provides an overview over all functions the effect provides:

Function	Description
void SetBpm (int bpm)	Sets the speed for the movement. <i>bpm</i> must be within a range of 0 to 9999 BPM. E.g. a value of 60 means that it is moved one pixel per second into the given direction. The default value is 1200.
int GetBpm ()	Returns the current speed for the movement of objects in BPM.
void SetDirection (int direction)	Sets the direction of the movement. Valid values for <i>direction</i> are DIR_LEFT , DIR_RIGHT , DIR_UP , and DIR_DOWN .
int GetDirection ()	Returns the current effect direction.
void SetHeight (int value)	Sets the height (size) of elements. The default value is 10.
int GetHeight ()	Returns the current height (size) of elements.
void SetHeightMax (int value)	Sets the maximum height of elements.
int GetHeightMax (int value)	Returns the current maximum height of elements.
void SetWidth (int value)	Sets the width (size) of elements. The default value is 10.
int GetWidth ()	Returns the current width (size) of elements.
void SetWidthMax (int value)	Sets the maximum width of elements.
int GetWidthMax ()	Returns the current maximum width of elements.
void SetPitch (int value)	Sets the distance between elements. The default value is 10.
int GetPitch ()	Returns the currently used distance between elements.
void SetPitchMax (int value)	Sets the maximum distance between elements.
int GetPitchMax ()	Returns the current maximum distance between elements.
void SetFrequency (int index, float value)	Sets the rate of the graph function for mode 1, 2, or 3. Indexing (<i>index</i>) starts with 0. The default value is 1.
float GetFrequency (int index)	Returns the current frequency for mode 1, 2, or 3. Indexing (<i>index</i>) starts with 0.
void SetFrequencyMax (float value)	Sets the maximum frequency. A Distribution Mode should be activated first.
float GetFrequencyMax ()	Returns the maximum frequency.
void SetPeak (int index, int value)	Sets the Peak for the graph function for mode 1, 2, or 3. Indexing (<i>index</i>) starts with 0. The default value is 50.
int GetPeak (int index)	Returns the current Peak for mode 1, 2, or 3. Indexing (<i>index</i>) starts with 0.
void SetShape (int shape)	Sets the shape for elements. One of the shapes (Defines) described below must be used.

int GetShape ()	Returns the currently used shape for elements.
void SetColorMode (int mode)	Sets the color mode for the Color Table. One of the modes (Defines) described below must be used.
int GetColorMode ()	Returns the current color mode for the Color Table.
void SetModeHeight (int mode)	Sets the Distribution Mode for Height. One of the modes (Defines) described below must be used.
int GetModeHeight ()	Returns the current Distribution Mode for Height.
void SetModeWidth (int mode)	Sets the Distribution Mode for Width. One of the modes (Defines) described below must be used.
int GetModeWidth ()	Returns the current Distribution Mode for Width.
void SetModePitch (int mode)	Sets the Distribution Mode for Pitch. One of the modes (Defines) described below must be used.
int GetModePitch ()	Returns the current Distribution Mode for Pitch.
void SetModeFrequency (int mode)	Sets the Distribution Mode for Frequency. One of the modes (Defines) described below must be used.
int GetModeFrequency ()	Returns the current Distribution Mode for Frequency.
void SetText (string text)	Sets the text for the shape Text. Example: SetText("MADRIX").
string GetText ()	Returns the currently used text.
void SetRotation (int angle)	Rotates the text by multiples of 90°. Valid values for <i>angle</i> are 0, 90, 180, and 270.
int GetRotation ()	Returns the current rotation of text.
void SetFontWidth (int value)	Sets the width of the font.
int GetFontWidth ()	Returns the width of the currently used font.
void SetFontHeight (int value)	Sets the height of the font.
int GetFontHeight ()	Returns the height of currently used font.
void SetFontItalic (int value)	Sets the font in italics (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontItalic ()	Returns if the font used is in italics.
void SetFontUnderline (int value)	Sets underlining for the font (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontUnderline ()	Returns if the currently used font is underlined.
void SetFontStrikeOut (int value)	Sets strikeout for the font (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontStrikeOut ()	Returns if the a strikeout font is used.
void SetFontWeight (int value)	Sets the weight of the font. Valid values for <i>value</i> range from 0 to 1000.
int GetFontWeight ()	Returns the weight of the currently used font.
void SetFontFaceName (string name)	Sets which font to use. A maximum of 31 characters is allowed for <i>string</i> . Example: SetFontFaceName("Arial");
string GetFontFaceName ()	Returns the name of the font currently in use.
void SetMode (int mode)	Sets the text mode. One of the modes (Defines) described below must be used.
int GetMode ()	Returns the currently used text mode.

void SetContinuous (int enable)	Enables (<i>enable</i> = 1) or disables (<i>enable</i> = 0) continuous text ("Cont. Text") for the Text mode.
int GetContinuous ()	Returns if continuous text is enabled (1) or not (0).
void SetGraphMode (int index, int mode)	Sets the trigonometric functions for the graph for mode 1, 2, or 3. Valid values for <i>index</i> are 0, 1, and 2. Indexing (<i>index</i>) starts with 0. One of the modes (Defines) described below must be used.
int GetGraphMode (int index)	Returns the currently used trigonometric mode for mode 1, 2, or 3. Valid values for <i>index</i> are 0, 1, and 2. Indexing (<i>index</i>) starts with 0.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color of the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Trigonometric Functions

This effect uses four trigonometric functions. The function *SetGraphMode* can be used to change them. The following values can be used as parameter:

Value	Description
MODE_SINE	Selects the sine function.
MODE_COSINE	Selects the cosine function.
MODE_TRIANGLE	Selects the triangle function.
MODE_SQUARE	Selects the square function.
MODE_NONE	Deselects the trigonometric function.

Distribution Modes

This effect uses various Distribution Modes for several functions. The following values can be used as parameter:

Value	Description
MODE_UNIFORM	Selects a uniform distribution curve as function.
MODE_LINEAR	Selects a linear distribution curve as function.
MODE_QUADRATIC	Selects a quadratic distribution curve as function.
MODE_SQRT	Selects a square root distribution curve as function.
MODE_CUBIC	Selects a cubic distribution curve as function.
MODE_RANDOM	Selects a random distribution curve as function.

Shapes

This effect uses various geometrical figures. The function *SetShape* can be used to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_CROSS	Selects the cross for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_DIAMOND	Selects the unfilled diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.
DRAW_RANDOM	Selects random shapes for drawing.
DRAW_TEXT	Selects the text mode.

Color Modes

This effect uses various color modes in the Color Table. The function *SetColorMode* can be used to change them. The following values can be used as parameter:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

Text Modes

The Text shape of this mode supports different text modes. The given text can be interpreted as a whole sentence, as single words, or even single characters. With the function *SetMode* it is possible to set the mode with the following parameters:

Value	Description
MODE_SENTENCE	Sets the sentence mode.
MODE_WORD	Sets the word mode.
MODE_CHAR	Sets the character mode.

5.3.12 SCE Metaballs

Functions Provided By SCE Metaballs

The following table provides an overview over all functions the effect provides:

Function	Description
void SetBpm (int bpm)	Sets the speed for the movement. <i>bpm</i> must be within a range of 0 to 9999 BPM. E.g. a value of 60 means that it is moved one pixel per second into the given direction.
int GetBpm ()	Returns the current speed for the movement of objects in BPM.

void SetInnerGlow (int value)	Sets the value for the inner glow of objects. Valid values for <i>value</i> range from 0 to 100.
int GetInnerGlow ()	Returns the value of the inner glow of objects.
void SetOuterGlow (int value)	Sets the value for the inner glow of objects. Valid values for <i>value</i> range from 0 to 100.
int GetOuterGlow ()	Returns the value of the outer glow of objects.
void SetBorder (int size)	Sets the border size of objects. Valid values for <i>size</i> range from 1 to 100.
int GetBorder ()	Returns the border size of objects.
void SetMaxSize (int size)	Sets the maximum size of the objects. Valid values for <i>size</i> range from 1 to 100.
int GetMaxSize ()	Returns the current maximum size of the objects.
void SetCount (int number)	Sets the number of objects. Valid values for <i>number</i> range from 1 to 20.
int GetCount ()	Retrieves the current number of objects.
void SeedRandom ()	Instantly creates Metaballs with random sizes every time the function is called.
void SetShape (int shape)	Sets the shape to use. One of the defines described below must be used.
int GetShape ()	Returns the currently defined shape.
void SetMode (int mode)	Sets the distribution curve for the size of the Metaballs. One of the defines described below must be used.
int GetMode ()	Retrieves the currently used size distribution mode.
void SetScale (float value)	Sets the scale of the Metaballs in random mode using relative values ranging from 0.01 to 2.0.
float GetScale ()	Retrieves the currently set scale of the effect. Scale can only be used while in random mode.
void SetColorMode (int mode)	Sets the mode of the Color Table. See below for further details.
int GetColorMode ()	Returns the currently set mode of the Color Table.
int GetColorCount ()	Returns the number of colors in the Color Table.
void RemoveColor (int idx)	Removes a color from the Color Table.
void SetColor (int idx, color c)	Sets the color values for a certain, indexed color in the Color Table.
color GetColor (int idx)	Returns which color is set for a certain color at a certain position (index) in the Color Table.
void AddColor (int idx, color c)	Adds color c at position idx to the Color Table.
void SetColorMix (int mode)	Sets the second shape for the color mix. See below for further details.
int GetColorMix ()	Returns the currently set second shape for the color mix.
void SetSharpness (int mode)	Sets the type of color gradient. See below for further details.
int GetSharpness ()	Returns the currently set type of color gradient.

void SetColorMixLink (int value)	Sets if auto adjustment for the two shapes is activated (1) or not (0).
int GetColorMixLink ()	Returns if auto adjustment is enabled or disabled.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color of the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void setColor (color col)	Sets the color of the effect. Please note: This function is deprecated. Use the color functions described above instead.
color getColor ()	Returns the color of the effect. Please note: This function is deprecated. Use the color functions described above instead.

Shapes

This effect uses various geometrical figures. With the function *SetShape* it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_DIAMOND	Selects the diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.

Distribution Modes

This effect uses various size distribution modes. The function *SetMode* can be used to change them. The following values can be used as parameter:

Value	Description
MODE_UNIFORM	Selects a uniform size of the Metaballs.
MODE_LINEAR	Selects a linear distribution curve for the size of the Metaballs.
MODE_QUADRATIC	Selects a quadratic distribution curve for the size of the Metaballs.
MODE_SQRT	Selects a square root distribution curve for the size of the Metaballs.
MODE_CUBIC	Selects a cubic distribution curve for the size of the Metaballs.
MODE_RANDOM	Selects a random distribution curve for the size of the Metaballs.

Color Modes

This effect uses various color modes in the Color Table. The function *SetColorMode* can be used to change them. The following values can be used as parameter:

Value	Description
-------	-------------

CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

Modes

This effect uses a second type of shape and mixes colors of both. With the function *SetColorMix* it is possible to change them. The following values can be used as parameter:

Value	Description
MODE_CIRCLE	Selects the circle mode.
MODE_RECTANGLE	Selects the rectangle mode.
MODE_DIAMOND	Selects the diamond mode.

Sharpness

This effect uses various geometrical figures. With the function *SetSharpness* it is possible to change them. The following values can be used as parameter:

Value	Description
MODE_VERY_BLURRY	Selects the very blurry color gradient for Metaballs with several colors.
MODE_BLURRY	Selects the blurry color gradient for Metaballs with several colors.
MODE_SLIGHTLY_BLURRY	Selects the slightly blurry color gradient for Metaballs with several colors.
MODE_MEDIUM	Selects the medium color gradient for Metaballs with several colors.
MODE_SLIGHTLY_CLEAR	Selects the slightly clear color gradient for Metaballs with several colors.
MODE_CLEAR	Selects the clear color gradient for Metaballs with several colors.
MODE_VERY_CLEAR	Selects the very clear color gradient for Metaballs with several colors.

5.3.13 SCE Plasma

Functions Provided By SCE Plasma

The following table provides an overview of all functions the effect can use:

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Ramp. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the Color Ramp. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the amount of colors currently used by the Color Ramp.
void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the <i>index</i> is lower or equal to 0, the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are <i>1</i> (On).
void RemoveColor (int index)	Removes the color at the specified <i>index</i> . If the given <i>index</i> is out of range, nothing happens.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the color position of the given <i>index</i> .
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.

This Effect uses the Color Ramp. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

5.3.14 SCE Pulse / Stroboscope

Functions Provided By SCE Pulse / Stroboscope

Function	Description
void SetColor (color col)	Sets the color for the effect.
color GetColor ()	Returns the currently set color.
void SetFrequency (int frequency)	Sets the frequency for the pulse mode. Valid values range from 1 to 25.
int GetFrequency ()	Returns the frequency for the pulse mode.
void SetOnTime (float time)	Sets the time the flash is activated for the strobe mode. The given time must be greater 0.
float GetOnTime ()	Returns the current time the flash is activated for the strobe mode.
void SetOffTime (float time)	Sets the time the flash is off for the strobe mode. The given time must be greater 0.
float GetOffTime ()	Returns the current time the flash is off for the strobe mode.
void SetMode (int mode)	Sets the mode for the effect. This may be pulse or strobe mode. See table below for valid values.
int GetMode ()	Returns the current mode of the effect. See table below for values handed back.
void SetFade (int enable)	Disables fade if <i>enable</i> is <i>false</i> , otherwise it will be enabled.
int GetFade ()	Returns <i>true</i> if fade is enabled, otherwise <i>false</i> .

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Effect Modes

This effect supports the two modes "pulse" and "strobe". To change the mode, call the function *SetMode* using one of the following parameters.

Value	Description
MODE_PULSE	Activates pulse mode.
MODE_STROBO	Activates strobe mode.

5.3.15 SCE Radial

Functions Provided By SCE Radial

The following table provides an overview of all functions the effect can use:

Function	Description
void SetColor (color c)	Sets the effect color.
color GetColor ()	Returns the current effect color.
void SetBpm (int speed)	Set the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetLength (int length)	Sets the length of the effect. Valid values range from 0 to 100.
int GetLength ()	Returns the current length.
void SetCount (int number)	Sets the helix count, which means the number of helixes to draw. <i>number</i> must be greater than 0.
int GetCount ()	Returns the current helix count.
void SetPixelCenter (int x, int y)	Sets the center for the effect in pixel coordinates.
int GetPixelCenterX ()	Returns the current x coordinate of the center in pixels.
int GetPixelCenterY ()	Returns the current y coordinate of the center in pixels.
void SetVectorCenter (float x, float y)	Sets the center for the effect using relative coordinates.
float GetVectorCenterX ()	Returns the current x coordinate of the center using relative coordinates.
float GetVectorCenterY ()	Returns the current y coordinate of the center using relative coordinates.
void SetEffectMode (int mode)	Sets the effect mode. See below for further details.

int GetEffectMode ()	Returns the currently set effect mode. See below for further details.
void SetCurve (int curve)	Sets the curve mode. Valid values are 0 to 5 or the defines described below .
int GetCurve ()	Returns the currently used curve. See <i>SetCurve</i> for valid values.
void SetRotation (int direction)	Sets the direction of the rotation to clockwise or counter-clockwise. See table below for possible values of <i>direction</i> .
int GetRotation ()	Returns the current rotation direction. See table below for possible return values.
void SetDirection (int direction)	Sets the direction of the helix and the circle from center to the outside or vice versa. See below for further details.
int GetDirection ()	Returns the current direction. See below for further details.
void SetFactor (int factor)	Sets the current <i>factor</i> value. Valid values range from 0 to 100.
int GetFactor ()	Returns the current factor value.
void SetAmplitude (int amplitude)	Sets the current amplitude value. Valid values range from 0 to 100.
int GetAmplitude ()	Returns the current amplitude value.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Effect Modes

The radial effect supports the effect types "circle", "radar" and "helix". The function [SetEffectMode](#) can be used to set them. The following table provides an overview about the possible values as parameters for that function:

Value	Description
EFFECT_CIRCLE	Sets the "circle-mode".
EFFECT_RADAR	Sets the "radar-mode".
EFFECT_HELIX	Sets the "helix-mode".

Curve Modes

The radial effect supports different types of curves. The function [SetCurve](#) can be used to set them. The following table provides an overview about additional parameters:

Value	Description
CURVE_SIN	Sets a "sine" curve.
CURVE_PHASE	Sets the "phase" mode.
CURVE_ABSSIN	Uses an absolute value of sine for a curve.
CURVE_SAWTOOTH_DOWN	Sets the "sawtooth down" mode.
CURVE_SAWTOOTH_UP	Sets the "sawtooth up" mode.
CURVE_TRIANGLE	Uses a triangle curve.

Rotation Direction

The "radar"- and the "helix" - modes have two possible directions how to rotate: clockwise and anticlockwise. The function *SetRotation* may be used to set those directions, using the following values as parameter:

Value	Description
ROTATION_CW	Sets a clockwise rotation.
ROTATION_CCW	Sets a counter clockwise rotation.

Effect Direction

The "circle" mode supports outward and inward movements. The "helix" supports the same settings, but uses them in different ways. However, to set those directions, use the function *SetDirection* with the following values:

Value	Description
DIR_OUTWARDS	Sets direction to "outwards".
DIR_INWARDS	Sets direction to "inwards".

Full Example

The following example will move the center of the radial effect and change its colors as well as modes at the same time.

```
int g_dir;
float g_posX, g_posY;
const float g_speed = 1.25;

void InitEffect()
{

}

void PreRenderEffect()
{
    switch(g_dir)
    {
        case 0: //left
            if(g_posX <= 0.0) {
                g_dir++;
                g_posX = 0.0;
            } else {
                g_posX -= g_speed;
            }
            break;

        case 1: //up
            if(g_posY <= 0.0) {
                g_dir++;
                g_posY = 0.0;
                SetNextMode();
            } else {
                g_posY -= g_speed;
            }
            break;

        case 2: //right
            if(g_posX > (float)GetMatrixWidth()) {
                g_dir++;
                g_posX = (float)GetMatrixWidth();
            } else {
                g_posX += g_speed;
            }
            break;

        case 3: //down
        default:
            if(g_posY >= (float)GetMatrixHeight()) {
                g_dir = 0;
                g_posY = (float)GetMatrixHeight();
            } else {
                g_posY += g_speed;
            }
    }
} //switch
```

```

        SetPixelCenter((int)g_posX, (int)g_posY);
    }

    void SetNextMode()
    {
        if(GetEffectMode() == EFFECT_CIRCLE)
            SetEffectMode(EFFECT_RADAR);
        else if(GetEffectMode() == EFFECT_RADAR)
            SetEffectMode(EFFECT_HELIX);
        else {
            SetEffectMode(EFFECT_CIRCLE);
            if(GetCurve() == 1)
                SetCurve(CURVE_SIN);
            else
                SetCurve(CURVE_PHASE);
        }
    }

    void PostRenderEffect()
    {
    }

```

5.3.16 SCE Shapes

Functions Provided By SCE Shapes

Function	Description
void SetColorMode (color colmode)	Sets the effect color mode. See the color mode table below for further details.
color GetColorMode ()	Returns the current color mode. See the color mode table below for further details.
void SetShape (int shape)	Sets the shape of the objects. See below for details.
int GetShape ()	Returns the currently used shapes of the objects. See below for details.
void SetCount (int fade)	Sets the number of shapes which shall be created.
int GetCount ()	Returns the number of shapes currently used.
void SetVectorWidth (float vWidth)	Sets the size of the shapes in percent in relation to the matrix size. A value of 0.01 means a width of 1%, whereas 1.0 means 100%. Valid values range from 0.01 to 99.99. The given value is related to the smaller dimension of the matrix. So, if there is a matrix with a size of 5x20 pixels, 5 is used. And as you can see, it is possible to set a much greater value then the smaller dimension.
float GetVectorWidth ()	Returns the width of the shapes as percentage. See <i>SetVectorWidth</i> for further details.
void SetPixelWidth (int iWidth)	Sets the size of the objects in pixels. Valid values range from 1 to 9999.

int GetPixelWidth ()	Returns the size of the objects in pixels.
void SetVectorBorder (float vBorder)	Sets the border width of the shapes in percent. This value relates to the current width of the shapes. E.g. if the current width of the shapes is 10 pixels, a value of 0.5 would mean that the border gets a width of 5 pixels. Valid values range from 0.01 to 99.99. 1.0 represents 100% of the shapes' width.
float GetVectorBorder ()	Returns the border width of the shapes in percent relative to the shapes width. See <i>SetVectorBorder</i> for further details.
void SetPixelBorder (int iBorder)	Sets the current border thickness of the shapes in pixels.
int GetPixelBorder ()	Returns the current border thickness of the shapes in pixels.
void SetColor (int idx, color c)	Sets the color with the specified index to the given color value. If the index is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the current number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetDrawMode (int mode)	Sets the current draw mode the effect should use. Please note: This function is deprecated and may be removed in one of the next releases. Use SetShape instead.
int GetDrawMode ()	Returns the current draw mode. Please note: This function is deprecated and may be removed in one of the next releases. Use GetShape instead.

Color Modes

This effect uses various color modes. With the function **SetColorMode** it is possible to change them. The following values can be used as parameter:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

Shapes

This effect uses various geometrical figures. With the function **SetShape** it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_CROSS	Selects the cross for drawing.
DRAW_CROSS_IMPLode	Selects the imploding cross for drawing.
DRAW_CROSS_EXPLODE	Selects the exploding cross for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
DRAW_CIRCLE_IMPLode	Selects the imploding circle for drawing.
DRAW_CIRCLE_EXPLODE	Selects the exploding circles for drawing.
DRAW_RECT	Selects the unfilled rectangle for drawing.
DRAW_RECT_IMPLode	Selects the imploding rectangle for drawing.
DRAW_RECT_EXPLODE	Selects the exploding rectangle for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_STAR_IMPLode	Selects the imploding star for drawing.
DRAW_STAR_EXPLODE	Selects the exploding star for drawing.
DRAW_DIAMOND	Selects the diamond shape for drawing.
DRAW_DIAMOND_IMPLode	Selects the imploding diamond for drawing.
DRAW_DIAMOND_EXPLODE	Selects the exploding diamond for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
FILL_CIRCLE_IMPLode	Selects the filled imploding circle for drawing.
FILL_CIRCLE_EXPLODE	Selects the filled exploding circle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
FILL_RECT_IMPLode	Selects the filled imploding rectangle for drawing.
FILL_RECT_EXPLODE	Selects the filled exploding rectangle for drawing.
FILL_DIAMOND	Selects the filled diamond shape for drawing.
FILL_DIAMOND_IMPLode	Selects the filled imploding diamond for drawing.
FILL_DIAMOND_EXPLODE	Selects the filled exploding diamond shape for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

5.3.17 SCE Starfield

Functions Provided By SCE Starfield

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999. Default value is 1500.
int GetBpm ()	Returns the current speed of the effect in BPM.
void SetLength (int size)	Sets the length of the effect. Valid values range from 1 to 100. The default value is 20.
int GetLength ()	Returns the currently set length.
void SetCount (int count)	Sets the number of objects. Valid values range from 0 to 100. The default value is 50.
int GetCount ()	Returns the number of objects.
void SetWidth (int width)	Sets the size of the objects. Valid values range from 1 to 100. The default value is 10.
int GetWidth ()	Returns the size of the objects.
void SetDepth (int depth)	Sets the depth effect. Valid values range from 1 to 100. The default value is 80.
int GetDepth ()	Returns the currently set depth.
void SetRotation (int rotation)	Sets the amount of rotation. Valid values range from -100 to 100. The default value is 0.
int GetRotation ()	Returns the currently set amount of rotation.
void SetDirection (int dir)	Sets the direction of the effect. See below for details.
int GetDirection ()	Returns the currently set direction.
void SetColorMode (int mode)	Sets the color mode of the Color Table. See below for details.
int GetColorMode ()	Returns the currently set color mode.
void SetShape (int shape)	Sets the shape of the objects. See below for details.
int GetShape ()	Returns the currently set shape of objects.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color of the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Shapes

This effect uses various geometrical figures. The function [SetShape](#) can be used to change them. The following values can be used as parameter:

Value	Description
DRAW_RECT	Selects the unfilled rectangle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
DRAW_CROSS	Selects the cross for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_DIAMOND	Selects the unfilled diamond for drawing.
FILL_DIAMOND	Selects the filled diamond for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

Effect Direction

The "stars" can move outwards or inwards. To set those directions, use the function [SetDirection](#) with the following values:

Value	Description
DIR_OUTWARDS	Sets direction to "outwards".
DIR_INWARDS	Sets direction to "inwards".

Color Modes

This effect uses various color modes. The function *SetColorMode* can be used to change them. The following values can be used as parameter:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect loops through the colortable to select the colors for new shapes.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color for a new shape from the color table by random access.
CM_RANDOM	Sets the random mode. In this mode the colors for the shapes are generated by random but 10% of the colors will be generated by random access to the color table.

5.3.18 SCE Ticker

Functions Provided By SCE Ticker

Function	Description
void SetText (string text)	Sets the text the ticker shows.
string GetText ()	Retrieves the text shown by the ticker effect.
void SetDirection (int direction)	Sets the direction of the movement. Allowed are all <i>DIR_ values</i> described in the summary. Use DIR_NONE to stop the movement.
int GetDirection ()	Returns the currently set direction.
void SetPixelTextOffset (int x, int y)	Set the text offset in pixel coordinates.
int GetPixelTextOffsetX ()	Returns the x coordinate of the text offset using pixel coordinates.
int GetPixelTextOffsetY ()	Returns the y coordinate of the text offset using pixel coordinates.
void SetVectorTextOffset (float x, float y)	Sets text offset using relative coordinates.
float GetVectorTextOffsetX ()	Returns the x coordinate of the text offset using relative coordinates.
float GetVectorTextOffsetY ()	Returns the y coordinate of the text offset using relative coordinates.
int GetPixelTextWidth ()	Returns the width of the text using pixel coordinates.
int GetPixelTextHeight ()	Returns the height of the text using pixel coordinates.
float GetVectorTextWidth ()	Returns the width of the text using relative coordinates.
float GetVectorTextHeight ()	Returns the height of the text using relative coordinates.
int GetPixelTextPosX ()	Returns the current x coordinate of the position of the text using pixel coordinates.

int GetPixelTextPosY()	Returns the current y coordinate of the position of the text using pixel coordinates.
float GetVectorTextPosX()	Returns the current x coordinate of the position of the text using vector coordinates.
float GetVectorTextPosY()	Returns the current y coordinate of the position of the text using vector coordinates.
void SetTextColor (color col)	Sets the text color.
color GetTextColor()	Retrieves the text color.
void SetBpm (int bpm)	Sets the BPM value. Valid values range from 0 to 9999.
int GetBpm()	Returns the currently used BPM value.
void SetMode (int mode)	Sets the text mode. See below for further details.
int GetMode()	Returns the current text mode. See below for further details.
void SetReverseSentence (int enable)	Disables "Reverse Sentence" if <i>enable</i> is set to <i>false</i> . Otherwise, use <i>true</i> .
int GetReverseSentence()	Returns true if "Reverse Sentence" is active, otherwise <i>false</i> is returned.
void SetReverseWords (int enable)	Disables "Reverse Words" if <i>enable</i> is set to <i>false</i> . Otherwise, use <i>true</i> .
int GetReverseWords()	Returns <i>true</i> if "Reverse Words" is active, otherwise <i>false</i> is returned.
void SetRotation (int angle)	Rotates the text output by multiples of 90°. Valid values for <i>angle</i> are 0, 90, 180, and 270.
int GetRotation()	Returns the current rotation.
void SetContinuous (int enable)	Enables or disables "Cont. Text" mode. If <i>enable</i> is <i>false</i> , it will be disabled. Otherwise, use <i>true</i> .
int GetContinuous()	Returns <i>true</i> if "Cont. Text" mode is enabled, otherwise <i>false</i> .
void SetSmooth (int enable)	Enables or disables the "smooth" mode. If <i>enable</i> is <i>false</i> , it will be disabled. Otherwise, use <i>true</i> .
int GetSmooth()	Returns <i>true</i> if "smooth" mode is enabled, otherwise <i>false</i> .
int GetFontWidth()	Returns the width of the currently used font.
void SetFontWidth (int width)	Sets the width of the font.
int GetFontHeight()	Returns the height of currently used font.
void SetFontHeight (int height)	Sets the height of the font.
int GetFontItalic()	Returns if the font used is in italics.
void SetFontItalic (int value)	Sets the font in italics (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontUnderline()	Returns if the currently used font is underlined.
void SetFontUnderline (int value)	Sets underlining for the font (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontStrikeOut()	Returns if the a strikeout font is used.
void SetFontStrikeOut (int value)	Sets strikeout for the font (<i>value</i> = 1) or not (<i>value</i> = 0).
int GetFontWeight()	Returns the weight of the currently used font.

void SetFontWeight (int value)	Sets the weight of the font. Valid values for <i>value</i> range from 0 to 1000.
string GetFontFaceName ()	Returns the name of the font currently in use.
void SetFontFaceName (string)	Sets which font to use. A maximum of 31 characters is allowed for <i>string</i> . Example: SetFontFaceName("Arial");

Text Modes

The ticker effect supports different modes. The given text can be interpreted as a whole sentence, as single words, or even single characters. With the function *SetMode* it is possible to set the mode with the following parameters:

Value	Description
MODE_SENTENCE	Sets the sentence mode.
MODE_WORD	Sets the word mode.
MODE_CHAR	Sets the character mode.

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function/Define	Description
void SetTextMode (int mode)	Use SetMode instead.
void GetTextMode (int mode)	Use GetMode instead.
void SetContinuous (int enable)	Use SetContinuous instead.
int GetContinuous ()	Use GetContinuous instead.
void SetReverseWord (int enable)	Use SetReverseWords instead.
int GetReverseWord ()	Use GetReverseWords instead.
TM_SENTENCE	Use MODE_SENTENCE instead.
TM_WORD	Use MODE_WORD instead.
TM_CHAR	Use MODE_CHAR instead.

Full Example 1

The following example writes the current time onto the matrix. Furthermore, it moves the text from side to side.

```
time g_time = GetTime();

void InitEffect()
{
    SetText("Hallo Dresden");
    SetDirection(DIR_UP);
}

void PreRenderEffect()
{
    time t = GetTime();

    int diff = t.sec - g_time.sec;

    if(diff > 0 || diff < 0) {
        g_time = t;
        string s;
        if(g_time.hour < 10) {
            s += "0";
        }
        s += (string)g_time.hour;
        s += ":";

        if(g_time.min < 10)
            s += "0";
        s += (string)g_time.min;

        s += ".";

        if(g_time.sec < 10)
            s += "0";
        s += (string)g_time.sec;

        SetText(s);
    }

    if(GetDirection() == DIR_LEFT) {
        if(GetPixelTextPosX() < 0) {
            SetDirection(DIR_RIGHT);
        }
    } else if(GetDirection() == DIR_RIGHT) {
        if(GetPixelTextPosX() > GetMatrixWidth() - GetPixelTextWidth()) {
            SetDirection(DIR_LEFT);
        }
    } else {
        SetDirection(DIR_LEFT);
    }
}
```

```
void PostRenderEffect()  
{  
    int i = 128;  
    ClearAlpha(i);  
    WriteText(GetText());  
}  
  
int compareTimes(time t1, time t2)  
{  
    int result = 0;  
    if(t1.hour < t2.hour)  
        result = -1;  
    else if(t1.hour > t2.hour)  
        result = 1;  
    else {  
        if(t1.min < t2.min)  
            result = -1;  
        else if(t1.min > t2.min)  
            result = 1;  
        else {  
            if(t1.sec < t2.sec)  
                result = -1;  
            else if(t1.sec > t2.sec)  
                result = 1;  
        }  
    }  
  
    return(result);  
}
```

Full Example 2

This example includes SetText and GetTime.

» [Full Example 2](#)

Example 3

This example creates a digital clock and therefore displays the current time using the SCE Ticker Effect.

```
@scriptname="macro clock with sce_ticker";  
@author="sven";  
@version="1.0";  
@description="write the current time into the ticker text fields";  
  
void InitEffect()
```



```

{
}

void PreRenderEffect()
{
    time t = GetTime();
    SetText(ZeroString(t.hour) + ":" + ZeroString(t.min) + ":" + ZeroString(t.sec));
}

void PostRenderEffect()
{
}

void MatrixSizeChanged()
{
    InitEffect();
}

string ZeroString(int value)
{
    if(value<10)
        return "0" + (string)value;
    return (string)value;
}

```

5.3.19 SCE Video

Functions Provided By SCE Video

The following table provides an overview over all functions the effect provides.

Please note: Certain functions may not work due to restrictions and limitations of the video codec. (E.g. the Forward Backward Loop is only available for the QuickTime codec format.)

Function	Description
void SetDirection (int)	Sets the movement direction. Allowed are all directions described by the DIR_ constants . Set DIR_NONE to stop the movement.
int GetDirection ()	Gets the current movement direction. Returns DIR_NONE if the movement has been stopped.
int GetPixelImagePositionX ()	Returns the current absolute x pixel coordinate of the video.
int GetPixelImagePositionY ()	Returns the current absolute y pixel coordinate of the video.
float GetVectorImagePositionX ()	Returns the current relative x coordinate of the video.
float GetVectorImagePositionY ()	Returns the current relative y coordinate of the video.

void SetPixelImagePosition (int x, int y)	Sets the position of the video in absolute pixel coordinates.
void SetVectorImagePosition (float x, float y)	Sets the position of the video using relative coordinates.
int GetPixelImageWidth ()	Returns the width of the current displayed video as absolute pixel value.
int GetPixelImageHeight ()	Returns the height of the current displayed video as absolute pixel value.
float GetVectorImageWidth ()	Returns the width of the currently displayed video relative to the settings of the current matrix.
float GetVectorImageHeight ()	Returns the height of the currently displayed video relative to the settings of the current matrix.
void SetStretch (int enable)	If <i>enable</i> is <i>false</i> , stretch will be disabled. Otherwise, it will be enabled.
int GetStretch ()	Returns <i>true</i> if stretch is currently enabled, otherwise <i>false</i> .
void SetTile (int enable)	Disables tile mode of the bitmap effect if <i>enable</i> is set to <i>false</i> . Otherwise, tile mode will be enabled.
int GetTile ()	Returns <i>true</i> if the tile mode of the bitmap effect is currently enabled, otherwise <i>false</i> .
void SetGrey (int enable)	Disables grayscale mode if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetGrey ()	Returns <i>true</i> if grayscale mode is currently active, otherwise <i>false</i> .
void SetRgbToRgbw (int enable)	Disables the RGB-to-RGBW mode if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetRgbToRgbw ()	Returns <i>true</i> if the RGB-to-RGBW mode is enabled, otherwise <i>false</i> .
void SetBpm (int bpm)	Sets the speed for the image movement. <i>bpm</i> must be within a range of 0 to 9999 BPM. E.g. a value of 60 means that it is moved one pixel per second into the given direction.
int GetBpm ()	Returns the speed for the image movement in BPM.
void SetFilterColor (color col)	Sets the filter color for the effect. Please note that the alpha value of the color structure is not used by this effect.
color GetFilterColor ()	Returns the current filter color. Please note that the alpha value is not used and should be ignored.
void SetAutostart (int enable)	Disables auto start if <i>enable</i> is set to <i>false</i> . Otherwise, auto start is enabled.
int GetAutostart ()	Returns <i>true</i> if auto start is enabled, otherwise <i>false</i> .
void SetLoop (int mode)	Sets the video playback mode. Valid values are: NO_LOOP (for no loop), F_LOOP (for forward loop), and FB_LOOP (for forward and backward loop).
int GetLoop ()	Returns <i>true</i> if loop mode is enabled.
int GetVideoLoaded ()	Returns <i>true</i> if a video is already loaded.
int GetCapturing ()	Returns <i>true</i> if a capture device, e.g. a web-cam, has been selected for capturing.

int GetVideoRunning ()	Returns <i>true</i> if video or capture playback is active, otherwise <i>false</i> .
void StartVideo ()	Starts video playback. If a capture device has been selected, capturing is started.
void StopVideo ()	Stops video playing. If a capture device has been selected, capturing is stopped.
void SeekVideo (int secs)	Skips some seconds of the current video. <i>secs</i> may be a negative value in order to go back.
time GetVideoLength ()	Returns the length of the video as time structure. The structure is filled up with the hours, minutes, and seconds. If no video is loaded or capturing is active, 0 is returned.
void SetVideoTime (time t)	Sets the current time position of the video playback. If no video is loaded, nothing happens. See below for further details.
time GetVideoTime ()	Returns the current time of the video playback.
void SetRotation (int angle)	Rotates the video output by multiples of 90°. Valid values for <i>angle</i> are 0, 90, 180, and 270.
int GetRotation ()	Returns the current rotation of the video output.
void StartVideoBackward ()	Starts playing the video backwards. Only available with QuickTime video files. If a capture device has been selected, this function has no effect.
void SetVideoStartTime (time t)	Sets the start time of the video. The video starts playing from this position.
time GetVideoStartTime ()	Returns the position at which the video starts playing.
void SetVideoEndTime (time t)	Sets the end time of the video. The video ends playing at this position.
time GetVideoEndTime ()	Returns the position at which the video stops playing.
void SetVideoPlaySpeed (float speed)	Sets the playback speed of the video. Values between 0.1 and 2.0 are allowed. E.g. the value 2.0 means that the video will be running 2x faster than the original speed.
float GetVideoPlaySpeed ()	Returns the current video playback speed.
void SetVideoAspectRatio (int mode)	Sets the video aspect ratio. Valid values for <i>mode</i> are: ASPECT_ORIG (for the video original aspect ratio), ASPECT_MATRIX (for the current matrix aspect ratio), ASPECT_4_3 (for 4:3), and ASPECT_16_9 (for an aspect ratio of 16:9).

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetMoveSpeed (float speed)	Sets the speed for the video movement. The speed must be given in frames per second. E.g. 2 means that the video will be moved 2 pixels per second. The range is 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetMoveSpeed ()	Returns the speed for the video movement in frames per second. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Setting the Time for Video Playback

If a video is loaded that has a length of 2:45:00 the following source code would set the playback position to 1:25:30.

```
time t = {1, 25, 30};  
SetVideoTime(t);
```

Please note: If the given time is higher than the length of the video, the time is set to the end of the video.

(Description)

Full Example

The following example moves the video side to side on the matrix.

```
void InitEffect()  
{  
  
}  
  
void PreRenderEffect()  
{  
    if(GetDirection() == DIR_LEFT)  
    {  
        if(GetVectorImagePositionX() <= 0.0)  
        {  
            SetVectorImagePosition(0.0, GetVectorImagePositionY());  
            SetDirection(DIR_RIGHT);  
        }  
    }  
}
```

```

    }
    else if(GetDirection() == DIR_RIGHT)
    {
        if(GetVectorImagePositionX() + GetVectorImageWidth() >= 1.0)
        {
            SetVectorImagePosition(1.0 - GetVectorImageWidth(), GetVectorImagePositionY());
            SetDirection(DIR_LEFT);
        }
    }
    else
    {
        SetDirection(DIR_LEFT);
    }
}

void PostRenderEffect()
{
}

```

5.3.20 SCE Wave

Functions Provided By SCE Wave

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetColor (color col)	Sets the color for the effect.
color GetColor ()	Returns the current effect color.
void SetDirection (int dir)	Sets the direction for the effect. Allowed are all DIR_ values described in the table of constants, except DIR_NONE.
int GetDirection ()	Returns the current effect direction.
void SetPixelLength (int len)	Sets the wave length in pixels. Valid values range from 2 to 150.
int GetPixelLength ()	Returns the wave length in pixels.
void SetVectorLength (float len)	Sets the wave length relative to the current matrix size and effect direction. See below for further details.
float GetVectorLength ()	Returns the current wave length relative to the current matrix size and effect direction. See below for further details.
void SetEffectMode (int mode)	Sets the wave mode. See below for further details and valid values for <i>mode</i> .
int GetEffectMode ()	Returns the current wave mode. See below for further details.
void SetPeak (int peak)	Sets the peak value in triangle mode. Valid values range from 0 to 100.

Function	Description
int GetPeak ()	Returns the value set for the peak in triangle mode.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the effect speed. Valid values range from 0 to 166 FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
int GetSpeed ()	Returns the current effect speed in frames per second. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Using Relative Values For The Length

The relative values [SetVectorLength](#) uses relate to the actual direction and matrix size. Vertical and horizontal scroll directions are simple to understand. For horizontal (left, right) movements, the value relates to the matrix width. And for vertical movements (up, down), the value relates to the matrix height.

For diagonal movements, e.g. to the upper left corner, the value relates to the greater value of either matrix width or height. For example, the matrix has a width of 40 pixels and a height of 55 pixels. Then, the given value relates to the height of 55 pixels. This is due to the fact that it fits better than using the extent of the matrix diagonal.

Wave Modes

The wave effect provides different algorithms to calculate a wave, such as a sin curve or a sawtooth wave. Using the *SetEffectMode* it is possible to set them. The following table provides the possible parameters of this function:

Value	Description
EFFECT_ABS_SINE	Uses absolute values of a sin wave to describe a wave.
EFFECT_SINE_WAVE	Uses a sin wave.
EFFECT_SAWTOOTH_UP	Uses an upward sawtooth wave to describe the wave.
EFFECT_SAWTOOTH_DOWN	Uses a downward sawtooth wave to describe the wave.
EFFECT_TRIANGLE	Uses a triangle curve as to describe the wave.

5.4 S2L Sound2Light Effects

5.4.1 S2L Equalizer

Functions Provided By S2L Equalizer

The following table provides an overview over all functions the effect provides:

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetCrossed (int enable)	Enables cross-direction mode if <i>enable</i> is set to <i>true</i> , otherwise it is disabled.
int GetCrossed ()	Returns <i>true</i> if cross-direction mode is enabled, otherwise <i>false</i> is returned.
void SetDirection (int direction)	Sets the direction of the effect. Valid values for <i>direction</i> are <i>DIR_LEFT</i> , <i>DIR_RIGHT</i> , <i>DIR_UP</i> , and <i>DIR_DOWN</i> .
int GetDirection ()	Returns the current direction.
void SetFade (int fade)	Sets the fade value. Valid values range from 1 to 100.
int GetFade ()	Returns the current fade value.
void SetAmplify (int amp)	Sets the amplification value (Amplify). Valid values range from 0 to 100.

Function	Description
int GetAmplify ()	Returns the current amplify value.
void SetBandCount (int count)	Sets the number of equalizer bands to display. Valid values range from 1 to 511.
int GetBandCount ()	Returns the current number of equalizer bands.
void SetInvert (int enable)	Disables inversion mode ("Invert") if enable is set to <i>false</i> , otherwise it will be enabled.
int GetInvert ()	Returns <i>true</i> if "Invert" is enabled, otherwise <i>false</i> .
void SetMirror (int enable)	Disables "Mirror" mode if enable is set to <i>false</i> , otherwise it will be enabled.
int GetMirror ()	Returns <i>true</i> if "Mirror" mode is enabled, otherwise <i>false</i> .
void SetViewLog (int enable)	Disables "View Log" if <i>enable</i> is <i>false</i> . Otherwise, it will be enabled.
int GetViewLog ()	Returns <i>true</i> if "View Log" is enabled, otherwise <i>false</i> .
void SetPushHF (int enable)	Disables "Push HF" (amplification of high frequencies) if enable is set to <i>false</i> . Otherwise, it will be enabled.
int GetPushHF ()	Returns <i>true</i> if "Push HF" is enabled, otherwise <i>false</i> .
void SetDoubleInwards (int enable)	Disables the doubled, inward-oriented equalizer if enable is set to <i>false</i> . Otherwise, it will be enabled.
int GetDoubleInwards ()	Returns <i>true</i> if "Double Inwards" is enabled, otherwise <i>false</i> .
void SetDoubleOutwards (int enable)	Disables the doubled, outward-oriented equalizer if enable is set to <i>false</i> . Otherwise, it will be enabled.
int GetDoubleOutwards ()	Returns <i>true</i> if "Double Outwards" is enabled, otherwise <i>false</i> .
void SetMonochrome (int enable)	Enables or disables monochrome mode for the equalizer.
int GetMonochrome ()	Returns <i>true</i> if the equalizer is set to monochrome mode, otherwise <i>false</i> .
void SetDrawMode (int mode)	Sets the draw mode for the effect. See below for further details.
int GetDrawMode ()	Returns the currently selected draw mode. See below for further details.
void SetBandMode (int state)	Enables (<i>state</i> set to 1) or disables (<i>state</i> set to 0) band mode for the effect.
int GetBandMode ()	Retrieves if band mode is activated (1) or not (0).
void SetColor (int index, color c)	Sets the color at the specified <i>index</i> in the Color Ramp dialog. If <i>index</i> is out of range, nothing happens.
color GetColor (int index)	Returns the color with the specified <i>index</i> in the Color Ramp dialog. If <i>index</i> is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors used in the Color Ramp.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new <i>index</i> . Valid values for <i>position</i> range from 0.01 to 0.99. The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the position of a color at the specified <i>index</i> .
void SetColorFade (int index, int fade)	Disables or enables color fade for the color at the specified <i>index</i> by either using 0 (Off) or 1 (On) for <i>fade</i> .

Function	Description
int GetColorFade (int index)	Returns if color fade is set for the color at the specified <i>index</i> .
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.
void AddColor (color c, float position, int index)	Adds another color to the Color Ramp at the specified <i>position</i> . Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int index)	Removes the color at the specified index. If the given <i>index</i> is out of range, nothing happens.

This Effect uses the Color Ramp dialog. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Draw Modes

This effect uses various modes. With the function *SetDrawMode* it is possible to change them. The following values can be used as parameter:

Value	Description
MODE_BAR	Selects the normal bar mode.
MODE_RADIAL	Selects the unfilled radial mode.
MODE_RADIAL_LINE	Selects the filled radial mode without an outline.
MODE_RADIAL_OUTLINE	Selects the radial mode with an outline.
MODE_RADIAL_DOT	Selects the radial mode drawing a dotted outline.

Full Example

The following example combines Music2Light data with a Sound2Light effect. It sets the number of bands according to the current tonality.

```
int t = 10;

void InitEffect()
{
    SetBandCount(t);
    SetCrossed(true);
    SetDoubleInwards(true);
}

void PreRenderEffect()
{
    t = t / 2 + GetTonality() / 2;
    if (t > 1)
        SetBandCount(t * t);
}

void PostRenderEffect()
{
}
```

5.4.2 S2L EQ Drops

Functions Provided By S2L EQ Drops

The following table provides an overview over all functions the effect provides:

Function	Description
void SetDirection (int dir)	Set the direction of the effect. Valid values are DIR_UP , DIR_DOWN , DIR_LEFT and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetColor (color col)	Sets the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetLength (int len)	Sets the length of the drops. Valid values range from 1 to 100.
int GetLength ()	Returns the current length of drops.
void SetSens (int sens)	Sets the sensitivity value for the effect. Valid values range from 0 to 100. If "Use Level (db)" is enabled, this function has no effect because instead of the "Sensitivity" value, the "Amplify" value will be used.
int GetSens ()	Returns the current sensitivity value of the effect. If "Use Level (db)" is enabled, this function returns 0 because not the "Sensitivity" value is used by the effect, but the amplify value.
void SetAmplify (int amp)	Sets the amplification value of the effect. Valid values range from 0 to 100. This function has no effect if "Use Level (db)" is disabled. In that case the "Sensitivity" value is used by the effect.
int GetAmplify ()	Returns the currently set "Amplify" value. This function returns 0 if "Use Level (db)" is disabled. In that case the "Sensitivity" value is used by the effect.
void SetBandCount (int cnt)	Sets the number of "Bands" to be displayed. Valid values range from 1 to 511.
int GetBandCount ()	Returns the current number of "Bands".
void SetInvert (int enable)	Disables "Invert" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetInvert ()	Returns <i>true</i> if "Invert" is enabled, otherwise <i>false</i> .
void SetMirror (int enable)	Disables "Mirror" mode if <i>enable</i> is <i>false</i> . Otherwise it will be enabled.
int GetMirror ()	Returns <i>true</i> if "Mirror" is enabled, otherwise <i>false</i> .
void SetUseLevel (int enable)	Disables "Use Level (db)" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetUseLevel ()	Returns <i>true</i> if "Use Level (db)" is enabled, otherwise <i>false</i> .

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

5.4.3 S2L EQ Shapes

Functions Provided By S2L EQ Shapes

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColorMode (color colmode)	Sets the color mode for the effect. See the table below for further details.
color GetColorMode ()	Returns the currently set color mode. See the table below for further details.
void SetShape (int shape)	Sets the shape of the objects. See below for details.
int GetShape ()	Returns the currently used shape of the objects. See below for details.
void SetFade (int fade)	Sets the fade value in BPM. Valid values range from 1 to 3000 BPM.
int GetFade ()	Returns the currently used fade value.
void SetSens (int sens)	Set the "Sensitivity" value for the effect. Valid values range from 0 to 100.
int GetSens ()	Returns the current "Sensitivity" value of the effect.

void SetVectorWidth (float vWidth)	Sets the size of the shapes in percent referring to the matrix size. Valid values range from 0.01 to 99.99. The given value is related to the smaller dimension of the matrix. E.g. if there is a matrix with a size of 5x20 pixels, the given value means n% of 5. Whereas 1.0 means 100%. As you can see, it is possible to set a much greater value then the smaller dimension.
float GetVectorWidth ()	Returns the width of the shapes in percent. See <i>SetVectorWidth</i> for further details.
void SetPixelWidth (int iWidth)	Sets the size of the objects in pixels. Valid values range from 1 to 9999.
int GetPixelWidth ()	Returns the size of the objects in pixels.
void SetVectorPitch (float vPitch)	Sets the pitch of the shapes in percent in relation to the matrix size. Valid values range from 0.01 to 99.99. The given value is related to the smaller dimension of the matrix. E.g. if there is a matrix with a size of 5x20 pixels, the given value means n% of 5. Whereas 1.0 means 100%. As you can see, it is possible to set a much greater value then the smaller dimension.
float GetVectorPitch ()	Returns the pitch of the shapes in percent. See <i>SetVectorPitch</i> for further details.
void SetPixelPitch (int iPitch)	Sets the pitch between two shapes in pixels. Valid values range from 1 to 9999.
int GetPixelPitch ()	Returns the currently set pitch.
void SetVectorBorder (float vBorder)	Sets the border width of the shapes in percent. This value relates to the current width of the shapes. E.g. with a current width of 10 pixels, a value of 0.5 would mean that a border with 5 pixels will be used. Valid values range from 0.01 to 99.99. A value of 1.0 means 100% of the shapes' width.
float GetVectorBorder ()	Returns the border width of the shapes in percent relative to the shapes width. See <i>SetVectorBorder</i> for further details.
void SetPixelBorder (int iBorder)	Sets the current border thickness of the shapes in pixels.
int GetPixelBorder ()	Returns the current border thickness of the shapes in pixels.
void SetAmplify (int amp)	Sets the "Amplify" value. Valid values range from 0 to 100.
int GetAmplify ()	Returns the currently set amplify value.
void SetUseLevel (int enable)	Enables or disables usage of the sound level for the opacity.
int GetUseLevel ()	Returns <i>true</i> , if usage of the sound level for the opacity is enabled. Otherwise, <i>false</i> is returned.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetDrawMode (int mode)	Sets the current draw mode the effect should use. Please note: This function is deprecated and may be removed in one of the next releases. Use SetShape instead.
int GetDrawMode ()	Returns the current draw mode. Please note: This function is deprecated and may be removed in one of the next releases. Use GetShape instead.

Color Modes

The function [SetColorMode](#) can be used to set different coloring modes for the effect.

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect repetitively selects colors in color table for new shapes in a row.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color of a new shape from the color table randomly.
CM_RANDOM	Sets the random mode. In this mode the colors of the shapes are generated randomly. But 10% of the colors will be generated by random access to the color table.

Shapes

This effect uses different geometrical figures. With the function [SetShape](#) it is possible to change them. The following values can be used as parameters:

Value	Description
DRAW_CROSS	Selects the cross for drawing.
DRAW_CROSS_IMPLODE	Selects the imploding cross for drawing.

DRAW_CROSS_EXPLODE	Selects the exploding cross for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
DRAW_CIRCLE_IMPLODE	Selects the imploding circle for drawing.
DRAW_CIRCLE_EXPLODE	Selects the exploding circles for drawing.
DRAW_RECT	Selects the unfilled rectangle for drawing.
DRAW_RECT_IMPLODE	Selects the imploding rectangle for drawing.
DRAW_RECT_EXPLODE	Selects the exploding rectangle for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_STAR_IMPLODE	Selects the imploding star for drawing.
DRAW_STAR_EXPLODE	Selects the exploding star for drawing.
DRAW_DIAMOND	Selects the diamond shape for drawing.
DRAW_DIAMOND_IMPLODE	Selects the imploding diamond for drawing.
DRAW_DIAMOND_EXPLODE	Selects the exploding diamond for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
FILL_CIRCLE_IMPLODE	Selects the filled imploding circle for drawing.
FILL_CIRCLE_EXPLODE	Selects the filled exploding circle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
FILL_RECT_IMPLODE	Selects the filled imploding rectangle for drawing.
FILL_RECT_EXPLODE	Selects the filled exploding rectangle for drawing.
FILL_DIAMOND	Selects the filled diamond shape for drawing.
FILL_DIAMOND_IMPLODE	Selects the filled imploding diamond for drawing.
FILL_DIAMOND_EXPLODE	Selects the filled exploding diamond shape for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

5.4.4 S2L EQ Tubes

Functions Provided By S2L EQ Tubes

The following table provides an overview over all functions the effect provides:

Function	Description
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetColor (color col)	Sets the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetFade (int fade)	Set the "fade" value. Valid values range from 1 to 3000.
int GetFade ()	Returns the current "fade" value.
void SetSens (int sens)	Sets the "Sensitivity" value for the effect. Valid values range from 0 to 100. If "Use Level (db)" is enabled, this function has no effect because the "Amplify" value is used instead.
int GetSens ()	Returns the current "Sensitivity" value of the effect. If "Use Level (db)" is enabled, this function returns 0. Because in that case the "Amplify" value is used.
void SetAmplify (int amp)	Sets the "Amplify" value for the effect. Valid values range from 0 to 100. This function has no effect if "Use Level (db)" is disabled because then the "Sensitivity" value is used by the effect.
int GetAmplify ()	Returns the current "Amplify" value of the effect. This function returns 0 if "Use Level (db)" is disabled because then the "Sensitivity" value is used by the effect.
void SetBandCount (int cnt)	Sets the number of "bands" to be displayed. Valid values range from 1 to 511.
int GetBandCount ()	Returns the currently used number of "bands".
void SetInvert (int enable)	Disables "Invert" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetInvert ()	Returns <i>true</i> if "Invert" is enabled, otherwise <i>false</i> .
void SetMirror (int enable)	Disables "Mirror" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetMirror ()	Returns <i>true</i> if "Mirror" is enabled, otherwise <i>false</i> .
void SetUseLevel (int enable)	Disables "Use Level (db)" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetUseLevel ()	Returns <i>true</i> if "Use Level (db)" is enabled, otherwise <i>false</i> .
void SetDirection (int dir)	Sets the direction of the lines to horizontal or vertical. See below for details.
int GetDirection ()	Returns current direction of the lines. See below for details.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Setting The Direction

This effect supports two directions, vertical lines and horizontal lines. Using the [SetDirection](#) function, it is possible to set any of them with one of the following parameters:

Value	Description
DIR_HORIZONTAL	Sets horizontal lines to be used by the effect.
DIR_VERTICAL	Sets vertical lines to be used by the effect.
DIR_HV	Sets vertical and horizontal lines to be used by the effect.

5.4.5 S2L Frequency Flash

Functions provided By S2L Frequency Flash

The following table provides an overview over all functions the effect provides:

Function	Description
void SetFade (int fade)	Sets the "fade" value. Valid values range from 1 to 3000.
int GetFade ()	Returns the current "fade" value.

Function	Description
void SetColor (color col)	Sets the color of the effect. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetSens (int sens)	Set the "Sensitivity" value for the effect. Valid values range from 0 to 100.
int GetSens ()	Returns the current "Sensitivity" value of the effect.
void SetMinBand (int index)	Sets the minimum band that should be included in the effect calculation. The given value is an index, which describes the band that should be used. The table below provides an overview over available values and the appropriate band.
int GetMinBand ()	Returns the specifier of the current minimum band. See below for further details.
void SetMaxBand (int index)	Set the maximum band that should be included in the effect calculation. The given value is an index which describes the band that should be used. The table below provides an overview over available values and the appropriate band.
int GetMaxBand ()	Returns the specifier of the current maximum band. See below for further details.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Frequency Bands

This effect allows to select the frequency bands which should be used to calculate the flash. The functions [SetMinBand](#) and [SetMaxBand](#) may be used to set the minimum band and the maximum band. Both use an identifier value between 0 and 21 which describes one of the following frequency bands:

Value	Frequency Band in Hz
0	8
1	42
2	57
3	77
4	102
5	135
6	180
7	242
8	322
9	430
10	570
11	767
12	1020
13	1360
14	1830
15	2450
16	3270
17	4310
18	5780
19	7760
20	10.300
21	13.000

Example:

The source code below would select 77Hz as minimum band and 570Hz as maximum band:

```
SetMinBand(3);  
SetMaxBand(10);
```

5.4.6 S2L Level Color

Functions Provided By S2L Level Color

Function	Description
----------	-------------

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetColor (color col)	Sets the color of the effect. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetFade (int fade)	Sets the "fade" value. Valid values range from 1 to 3000.
int GetFade ()	Returns the current "fade".
void SetDirection (int dir)	Sets the direction of the effect. Valid values for <i>dir</i> are DIR_LEFT , DIR_RIGHT , DIR_UP and DIR_DOWN .
int GetDirection ()	Returns the current direction.
void SetMono (int enable)	Disables "mono" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetMono ()	Returns <i>true</i> if "mono" is enabled, otherwise <i>false</i> is returned.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

5.4.7 S2L Level Meter

Functions Provided By S2L Level Meter

The following table provides an overview over all functions the effect provides:

Function	Description
----------	-------------

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetFade (int fade)	Sets the "fade" value. Valid values range from 1 to 100.
int GetFade ()	Returns the current "fade" value.
void SetDirection (int dir)	Sets the direction of the effect. Valid values for <i>dir</i> are DIR_UP , DIR_DOWN , DIR_LEFT and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetCrossed (int enable)	Enables cross-direction mode if <i>enable</i> is set to <i>true</i> . Otherwise, it is disabled.
int GetCrossed ()	Returns <i>true</i> if cross-direction mode is enabled, otherwise <i>false</i> is returned.
void SetAmplify (int amp)	Sets the "Amplify" value. Valid values range from 0 to 100.
int GetAmplify ()	Returns the current "Amplify" value.
void SetBandCount (int cnt)	Sets the number of "bands" to be displayed. Valid values are 1 and 2.
int GetBandCount ()	Returns the current number of "bands".
int SetInvert (int enable)	Disables "Invert" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
void GetInvert ()	Returns <i>true</i> if "Invert" is enabled, otherwise <i>false</i> .
void SetMirror (int mirror)	Disables "Mirror" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetMirror ()	Returns <i>true</i> if "Mirror" is enabled, otherwise <i>false</i> .
void SetViewLog (int enable)	Disables "ViewLog" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetViewLog ()	Returns <i>true</i> if "ViewLog" is enabled, otherwise <i>false</i> .
void SetDoubleOutwards (int enable)	Disables "double outwards" if <i>enable</i> is <i>false</i> . Otherwise, it will be enabled.
int GetDoubleOutwards ()	Returns <i>true</i> if "double outwards" is enabled, otherwise <i>false</i> .
void SetDoubleInwards (int enable)	Disables "double inwards" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetDoubleInwards ()	Returns <i>true</i> if "double inwards" is enabled, otherwise <i>false</i> .
void SetMonochrome (int enable)	Enables or disables monochrome mode.
int GetMonochrome ()	Returns whether monochrome mode is enabled or not.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Ramp dialog. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the Color Ramp dialog. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors used in the Color Ramp.
int SetColorPosition (int idx, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from 0.01 to 0.99. The first and last color are not allowed to be moved!

float GetColorPosition (int idx)	Returns the position of a color at the specified index.
void SetColorFade (int idx, int fade)	Disables or enables color fade for the color at the specified position by either using <i>0</i> (Off) or <i>1</i> (On) for <i>fade</i> .
int GetColorFade (int idx)	Returns if color fade is set for the color at the specified index.
void FadeAllColors ()	Enables color fade for all colors in the Color Ramp.
void FadeNoneColors ()	Disables color fade for all colors in the Color Ramp.
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.
void AddColor (color c, float position, int idx)	Adds another color to the Color Ramp at the specified index <i>position</i> . Valid values for <i>position</i> range from <i>0.01</i> to <i>0.99</i> . If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color at the specified index. If the given index is out of range, nothing happens.

This Effect uses the Color Ramp dialog. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

5.4.8 S2L Level Ring

Functions Provided By S2L Level Ring

The following table provides an overview over all functions the effect provides:

Function	Description
----------	-------------

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetFade (int fade)	Sets the "fade" value. Valid values range from 1 to 100.
int GetFade ()	Returns the current "fade" value.
void SetAmplify (int amp)	Sets the "Amplify" value. Valid values range from 0 to 100.
int GetAmplify ()	Returns the current "Amplify" value.
void SetMono (int enable)	Disables "mono" if <i>enable</i> is set to <i>false</i> . Otherwise, use <i>true</i> .
int GetMono ()	Returns <i>true</i> if "mono" is enabled, otherwise <i>false</i> .
void SetViewLog (int enable)	Disables "ViewLog" if <i>enable</i> is set to <i>false</i> . Otherwise, use <i>true</i> .
int GetViewLog ()	Returns <i>true</i> if "ViewLog" is enabled, otherwise <i>false</i> .
void SetDirection (int dir)	Sets the direction of the effect. See below for details.
int GetDirection ()	Returns the current effect direction. See below for details.
void SetShape (int shape)	Sets the shape of the objects. See below for details.
int GetShape ()	Returns the currently used shape of the objects. See below for details.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Table dialog. If <i>idx</i> is out of range, nothing happens. The Color Table of this Effect always requires 3 entries.
color GetColor (int idx)	Returns the color with the specified index in the color table. If the index is out of range <i>black</i> is returned.
int GetColorCount ()	Returns the number of colors in the color table. The Color Table of this Effect always requires 3 entries.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

void SetDrawMode (int mode)	Sets the current draw mode the effect should use. Please note: This function is deprecated and may be removed in one of the next releases. Use SetShape instead.
int GetDrawMode ()	Returns the current draw mode. Please note: This function is deprecated and may be removed in one of the next releases. Use GetShape instead.

Setting The Direction

The directions for this effect are only "inward" or "outward". Therefore, **SetDirection** only gets the following values as valid parameters:

Value	Description
DIR_OUTWARDS	Sets the direction from the center to the outside.
DIR_INWARDS	Sets the direction from the outside to the center.

Shapes

This effect uses different geometrical figures. With the function **SetShape** it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_CROSS	Selects the cross for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
DRAW_RECT	Selects the unfilled rectangle for drawing.
DRAW_STAR	Selects the star for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.

5.4.9 S2L Waveform

Functions Provided By S2L Waveform

The following table provides an overview over all functions the effect provides:

Function	Description
void SetDirection (int direction)	Sets the direction for the effect. Valid values for <i>direction</i> are DIR_UP , DIR_DOWN , DIR_LEFT , and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetTimeSlot (float value)	Sets the time slot <i>value</i> . Valid values for <i>value</i> range from 0.01 to 9999.
float GetTimeSlot ()	Returns the currently set time slot.
void SetChannelView (int view)	Sets the channels that should be displayed by the effect. See below for further details.
int GetChannelView ()	Returns the currently set channel to be displayed by the effect. See below for further details.
void SetShiftOutwards (int enable)	Sets the direction to outwards if <i>enable</i> is set to <i>true</i> . Otherwise, it is disabled. See below for further details.
int GetShiftOutwards ()	Returns <i>true</i> if "Shift Outwards" is currently enabled, otherwise <i>false</i> . See below for further details.
void SetShiftInwards (int enable)	Sets the direction to inwards if <i>enable</i> is set to <i>true</i> , otherwise it is disabled. See below for further details.
int GetShiftInwards ()	Returns <i>true</i> if "Shift Inwards" is currently enabled, otherwise <i>false</i> . See below for further details.
void SetColor (int idx, color c)	Sets the color at the specified index in the Color Ramp. If <i>idx</i> is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the Color Ramp. If the index is out of range, <i>black</i> is returned.
int GetColorCount ()	Returns the amount of colors currently used by the Color Ramp.
void AddColor (color c, float position, int fade)	Adds another <i>color c</i> to the Color Ramp at the specified position. Valid values for <i>position</i> range from 0.01 to 0.99. If the <i>index</i> is lower or equal to 0, the new color is added to the first position. If <i>index</i> is greater than the current number of colors, the new color is added at the end. Valid values for <i>fade</i> are 1 (On).
void RemoveColor (int index)	Removes the color at the specified <i>index</i> . If the given <i>index</i> is out of range, nothing happens.
int SetColorPosition (int index, float position)	Sets the color of the given <i>index</i> to a new position and returns the new index. Valid values for <i>position</i> range from 0.01 to 0.99. The first and last color are not allowed to be moved!
float GetColorPosition (int index)	Returns the color position of the given <i>index</i> .
void SetUniformDistances ()	Sets uniform distances between each color in the Color Ramp.
void InvertColorPositions ()	Inverts the positions of the colors in the Color Ramp.
void InvertColors ()	Inverts every single color in the Color Ramp.

This Effect uses the Color Ramp. Learn more about [Using Colors](#).

Selecting The Sound Channels

The waveform effect is able to show the waveform of the two stereo sound channels. It is possible to select the waveform of the left or right channel. In addition, it is possible to show both channels at the same time (stereo). Selecting mono is also an option. The function [*SetChannelView*](#) enables a macro to set these settings. Therefore, the following values may be used as parameters:

Value	Description
CHM_MONO	Selects mono mode.
CHM_STEREO	Selects stereo mode, which displays both channels at the same time.
CHM_LEFT	Selects the left channel.
CHM_RIGHT	Selects the right channel.

Shifting The Direction

In addition to the directions [*DIR_UP*](#), [*DIR_DOWN*](#), [*DIR_LEFT*](#), and [*DIR_RIGHT*](#), the waveform can move inwards or outwards. With the two functions [*SetShiftOutwards*](#) and [*SetShiftInwards*](#), it is possible to control these directions. They both use *true* or *false* as parameters to disable or to enable the according movement.

Example:

- `SetShiftInwards(true);` `//activates inwards movement`
- `SetShiftInwards(false);` `//disables inwards movement`

- `SetShiftOutwards(true);` `//activates outwards movement`
- `SetShiftOutwards(false);` `//disables outwards movement`

5.4.10 S2L Wavegraph

Functions Provided By S2L Wavegraph

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColor (color col)	Sets the color for the effect.
color GetColor ()	Returns the current effect color.
void SetDirection (int dir)	Sets the direction for the effect. Valid values are DIR_UP , DIR_DOWN , DIR_LEFT , and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetTimeSlot (float timeSlot)	Sets the time slot value. Valid values range from 0.01 to 9999.
float GetTimeSlot ()	Returns the currently set time slot.
void SetChannelView (int view)	Sets the channels that should be displayed by the effect. See below for further details.
int GetChannelView ()	Returns the currently set channel to be displayed by the effect. See below for further details.
void SetShiftOutwards (int enable)	Sets the direction to outwards if <i>enable</i> is set to <i>true</i> . Otherwise, it is disabled. See below for further details.
int GetShiftOutwards ()	Returns <i>true</i> if "Shift Outwards" is currently enabled, otherwise <i>false</i> . See below for further details.
void SetShiftInwards (int enable)	Sets the direction to inwards if <i>enable</i> is set to <i>true</i> , otherwise it is disabled. See below for further details.
int GetShiftInwards ()	Returns <i>true</i> if "Shift Inwards" is currently enabled, otherwise <i>false</i> . See below for further details.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Selecting The Sound Channels

The wavegraph effect is able to show the wave graph of the two stereo sound channels. It is possible to select the wavegraph of the left or right channel. In addition, it is possible to show both channels at the same time (stereo). Selecting mono is also an option. The function *SetChannelView* enables a macro to set these settings. The following values may be used as parameters:

Value	Description
CHM_MONO	Selects mono mode.
CHM_STEREO	Selects stereo mode, which displays both channels at the same time.
CHM_LEFT	Selects the left channel.
CHM_RIGHT	Selects the right channel.

Shifting The Direction

In addition to the directions *DIR_UP*, *DIR_DOWN*, *DIR_LEFT*, and *DIR_RIGHT*, the wavegraph can move inwards or outwards. With the two functions *SetShiftOutwards* and *SetShiftInwards*, it is possible to control these directions. They both use *true* or *false* as parameters to disable or to enable the according movement.

Example:

- `SetShiftInwards(true);` `//activates inwards movement`
- `SetShiftInwards(false);` `//disables inwards movement`

- `SetShiftOutwards(true);` `//activates outwards movement`
- `SetShiftOutwards(false);` `//disables outwards movement`

5.5 M2L Music2Light Effects

5.5.1 M2L Color Fade

Functions Provided By M2L Color Fade

The following table provides an overview over all functions the effect provides:

Function	Description
----------	-------------

void SetColor (color col)	Sets the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetFade (int fade)	Sets the fade value. Valid values range from 1 to 100.
int GetFade ()	Returns the current fade value.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

5.5.2 M2L Color Rings

Functions Provided By M2L Color Rings

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColor (color col)	Sets the color of the effect. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetDirection (int dir)	Sets the direction of the effect. See below for further details.
int GetDirection ()	Returns the current direction. See below for further details.

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
----------	-------------

void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetSpeed instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetSpeed instead.

Setting the Direction

This effect draws rings depending on the music. Those rings are drawn around the matrix and get smaller with each call (inwards) or they are drawn in the center of the matrix and grow with each frame (outwards). Using the function **SetDirection** it is possible to set this direction.

Value	Description
DIR_INWARDS	Selects an inward direction. Large rings will be drawn around the matrix and shrink with each frame.
DIR_OUTWARDS	Selects an outward direction. Small rings will be drawn in the center of the matrix and grow with each frame.

5.5.3 M2L Color Scroll

Functions Provided By the M2L Color Scroll

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColor (color col)	Sets the color of the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetDirection (int dir)	Sets the direction of the effect. Allowed are all directions described in the general summary and some additional values which are described below .
int GetDirection ()	Returns the current direction. There are some effect specific directions which are described below .
void SetCrossed (enable)	Enables cross-direction mode if <i>enable</i> is set to <i>true</i> . Otherwise, it is disabled.
int GetCrossed ()	Returns <i>true</i> if the cross-direction mode is enabled, otherwise <i>false</i> .

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetStep (int step)	Sets the step width of the new colors. Valid values for <i>step</i> range from 1 to 9999.
int GetStep ()	Returns the current step width.
void SetXOffset (int offset)	Sets the x-offset value, which ranges from 1 to 9999.
int GetXOffset ()	Returns the current x-offset.
void SetFade (int enable)	Enables fading between the colors if <i>enable</i> is <i>true</i> . Otherwise it will be disabled.
int GetFade ()	Returns <i>true</i> if fading is enabled, otherwise <i>false</i> .

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Additional Directions

In addition to the standard directions **DIR_UP**, **DIR_DOWN**, **DIR_LEFT**, and **DIR_RIGHT**, four more directions are effect-specific. These are vertical and horizontal scrolling lines, both inwards and outwards. It is possible to set them using the function **SetDirection**. The following values must be used as parameters:

Value	Description
DIR_OUTWARDS_V	Sets a vertical outward movement (Vertical Explode).
DIR_INWARDS_V	Sets a vertical inward movement (Vertical Implode).
DIR_OUTWARDS_H	Sets a horizontal outward movement (Horizontal Explode).
DIR_INWARDS_H	Sets a horizontal inward movement (Horizontal Implode).
DIR_OUTWARDS_RECT	Sets a rectangular outward movement (Rectangle Explode).
DIR_INWARDS_RECT	Sets a rectangular inward movement (Rectangle Implode).
DIR_OUTWARDS_SQUARE	Sets a quadratic outward movement (Square Explode).
DIR_INWARDS_SQUARE	Sets a quadratic inward movement (Square Implode).
DIR_OUTWARDS_CIRCLE	Sets a circled outward movement (Circle Explode).
DIR_INWARDS_CIRCLE	Sets a circled inward movement (Circle Implode).
DIR_OUTWARDS_ELLIPSE	Sets an elliptic outward movement (Ellipse Explode). This is different to a circle on rectangular matrices. On quadratic matrices the ellipse will also be a circle.
DIR_INWARDS_ELLIPSE	Sets an elliptic inward movement (Ellipse Implode). This is different to a circle on rectangular matrices. On quadratic matrices the ellipse will also be a circle.

5.5.4 M2L Interval Drops

Functions Provided by M2L Interval Drops

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColor (color col)	Sets the color of the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.

void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetDirection (int dir)	Sets the direction of the effect. Valid values are DIR_UP , DIR_DOWN , DIR_LEFT and DIR_RIGHT .
int GetDirection ()	Returns the current direction.
void SetLength (int len)	Sets the length of the drops. Valid values range from 1 to 100.
int GetLength ()	Returns the current length of drops.
void SetPixelWidth (int w)	Sets the width of the drops in pixels. Valid values range from 1 to 9999.
int GetPixelWidth ()	Returns the width of the drops in pixels.
void SetVectorWidth (float w)	Sets the width of the drops using relative coordinates between 0 and 1 according to the current matrix size and direction. See below for further details.
float GetVectorWidth ()	Returns the width of the drops relative to the current matrix size and direction. See below for further details.
void SetUseBass (int enable)	Disables "use bass" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetUseBass ()	Returns <i>true</i> if "use bass" is enabled, otherwise <i>false</i> .

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Using a Relative Width of Drops

The width of the drops may be given in pixels or using relative values between 0 and 1. Please see the functions [SetPixelWidth](#) or [SetVectorWidth](#). A relative value of 0 represents the minimum size and will be increased to 1 pixel. And 1 represents the full size of the matrix.

Furthermore, these values depend on the current direction of the effect. If a horizontal movement is active (a movement to the left or right), the value is used in relation to the matrix height. If a vertical movement is active (up or down), the value is used in relation to the width of the matrix.

5.5.5 M2L Interval Tubes

Functions Provided By M2L Interval Tubes

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColor (color col)	Sets the color of the effect color. Please note that the alpha channel is not used by this effect.
color GetColor ()	Returns the currently set color. Please note that the alpha channel is not used and should be ignored.
void SetBpm (int bpm)	Sets the speed of the effect in BPM. Valid values range from 0 to 9999 BPM.
int GetBpm ()	Returns the current effect speed in BPM.
void SetDirection (int dir)	Set the direction for the effect. This effect has two effect specific directions which are described below .
int GetDirection ()	Returns the current direction. The returned values are described below .
void SetFade (int fade)	Set the fade value for the tubes. Valid values range from 1 to 3000.
int GetFade ()	Returns the current fade value.
void SetPixelWidth (int w)	Sets the width of the tubes in pixels. Valid values range from 1 to 9999.
int GetPixelWidth ()	Returns the width of the tubes in pixels.
void SetVectorWidth (float w)	Sets the width of the tubes using relative coordinates between 0 and 1 according to the current matrix size and direction. See below for further details.
float GetVectorWidth ()	Returns the width of the drops as relative value depending on the current matrix size and direction. See below for further details.

void SetUseBass (int enable)	Disables "use bass" if <i>enable</i> is set to <i>false</i> . Otherwise, it will be enabled.
int GetUseBass ()	Returns <i>true</i> if "use bass" is enabled, otherwise <i>false</i> .

This Effect uses the Color Picker. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetSpeed (int speed)	Sets the speed of the effect in FPS. Valid values range from 0 to 166. Please note: This function is deprecated and may be removed in one of the next releases. Use SetBpm instead.
float GetSpeed ()	Returns the current effect speed in FPS. Please note: This function is deprecated and may be removed in one of the next releases. Use GetBpm instead.

Setting The Direction

The effect is able to draw vertical or horizontal lines. The direction can be set by [SetDirection](#) with the following values as parameter:

Value	Description
DIR_HORIZONTAL	Draws the tubes as horizontal lines.
DIR_VERTICAL	Draws the tubes as vertical lines.
DIR_HV	Draws the tubes as vertical and horizontal lines.

Using A Relative Width Of Tubes

The width of the tubes may be given in pixels or as a relative value between 0 and 1. The function **SetVectorWidth** uses such relative coordinates. A value of 0 represents the minimum size and will be increased to 1 pixel. A value of 1 represents the whole size of the matrix.

Furthermore, this value depends on the current direction. If horizontal lines are activated, the value is used in relation to the matrix height. If vertical lines are activated, the value is used in relation to the width of the matrix.

5.5.6 M2L Single Tone Shapes

Functions Provided By M2L Single Tone

The following table provides an overview over all functions the effect provides:

Function	Description
void SetColorMode (color colmode)	Sets the color mode for the effect. See the table below for further details.
color GetColorMode ()	Returns the currently set color mode. See the table below for further details.
void SetShape (int shape)	Sets the shape of the objects. See below for details.
int GetShape ()	Returns the currently used shape of objects. See below for details.
void SetFade (int fade)	Sets the fade value in BPM. Valid values range from 1 to 3000.
int GetFade ()	Returns the current fade value.
void SetSens (int sens)	Sets the "Sensitivity" value of the effect. Valid values range from 0 to 100.
int GetSens ()	Returns the current "Sensitivity" value of the effect.
void SetVectorWidth (float vWidth)	Sets the size of the shapes in percent relative to the matrix size. Valid values range from 0.01 to 99.99. The given value is related to the smaller dimension of the matrix. If there is a matrix with size of 5x20 pixels for example, the given value means n% of 5. Whereas 1.0 means 100%. As you can see, it is possible to set a much greater value then the smaller dimension.
float GetVectorWidth ()	Returns the width of the shapes in percent. See SetVectorWidth for further details.
void SetPixelWidth (int iWidth)	Sets the size of the figures in pixels. Valid values range from 1 to 9999.
int GetPixelWidth ()	Returns the size of the objects in pixels.

void SetVectorPitch (float vPitch)	Sets the pitch of the shapes in percent in relation to the matrix size. Valid values range from 0.01 to 99.99. The given value is related to the smaller dimension of the matrix. E.g. if there is a matrix with a size of 5x20 pixels, the given value means n% of 5. A value of 1.0 means 100%. As you can see, it is possible to set a much greater value than the smaller dimension.
float GetVectorPitch ()	Returns the pitch of the shapes in percent. See <i>SetVectorPitch</i> for further details.
void SetPixelPitch (int iPitch)	Sets the pitch between two shapes in pixels. Valid values range from 1 to 9999.
int GetPixelPitch ()	Returns the currently set pitch.
void SetVectorBorder (float vBorder)	Sets the border width of the shapes in percent. This value relates to the current width of the shapes. E.g. if the current width of the shapes is 10 pixels, a value of 0.5 would mean that the border is drawn with a width of 5 pixels. Valid values range from 0.01 to 99.99. A value of 1.0 means 100% of the shapes width.
float GetVectorBorder ()	Returns the border width of the shapes in percent relative to the shapes width. See <i>SetVectorBorder</i> for further details.
void SetPixelBorder (int iBorder)	Sets the current border thickness of the shapes in pixels.
int GetPixelBorder ()	Returns the current border thickness of the shapes in pixels.
int GetColorCount ()	Returns the number of colors in the color table.
void AddColor (int idx, color c)	Adds another color to the color table at the specified index position. If the index is lower or equal to 0, the new color is added to the first position. If the index is greater than the current number of colors, the new color is added at the end.
void RemoveColor (int idx)	Removes the color located at the specified index. If the given index is out of range, nothing happens. The Color Table of this effect needs to include at least 1 entry.
void SetColor (int idx, color c)	Sets the color with the specified index in the Color Table to the given color. If the index is out of range, nothing happens.
color GetColor (int idx)	Returns the color with the specified index in the Color Table. If the index is out of range, <i>black</i> is returned.

This Effect uses the Color Table. Learn more about [Using Colors](#).

Deprecated Functions

Deprecated functions are outdated functions and should not be used anymore.

Function	Description
void SetPitch (int pitch)	Sets the pitch between two shapes in pixels. Valid values range from 1 to 9999. Please note: This function is deprecated and may be removed in a forthcoming release. Use <i>SetPixelPitch</i> instead

int GetPitch ()	Returns the currently set pitch in pixels. Please note: This function is deprecated and may be removed in a forthcoming release. Use GetPixelPitch instead.
void SetThickness (int thickness)	Sets the thickness of the shapes in pixels. Please note: This function is deprecated and may be removed in a forthcoming release. Use SetPixelBorder instead.
int GetThickness ()	Returns the current thickness. Please note: This function is deprecated and may be removed in a forthcoming release. Use GetPixelBorder instead.
void SetWidth (int width)	Sets the size of the objects in pixels. Valid values range from 1 to 9999. Please note: This function is deprecated and may be removed in a forthcoming release. Use SetPixelWidth instead.
int GetWidth ()	Returns the size of the objects in pixels. Please note: This function is deprecated and may be removed in a forthcoming release. Use GetPixelWidth instead.
void SetDrawMode (int mode)	Sets the current draw mode the effect should use. Please note: This function is deprecated and may be removed in one of the next releases. Use SetShape instead.
int GetDrawMode ()	Returns the current draw mode. Please note: This function is deprecated and may be removed in one of the next releases. Use GetShape instead.

Color Modes

The function **SetColorMode** can be used to determine the used coloring mode:

Value	Description
CM_LOOP	Sets the loop mode. In this mode the effect repetitively selects colors in color table for new shapes in a row.
CM_SHUFFLE	Sets the shuffle mode. In this mode the effect selects the color of a new shape from the color table randomly.
CM_RANDOM	Sets the random mode. In this mode the colors of the shapes are generated randomly. But 10% of the colors will be generated by random access to the color table.

Shapes

This effect uses different geometrical figures. With the function *SetShape* it is possible to change them. The following values can be used as parameter:

Value	Description
DRAW_CROSS	Selects the cross for drawing.
DRAW_CROSS_IMPLODE	Selects the imploding cross for drawing.
DRAW_CROSS_EXPLODE	Selects the exploding cross for drawing.
DRAW_CIRCLE	Selects the unfilled circle for drawing.
DRAW_CIRCLE_IMPLODE	Selects the imploding circle for drawing.
DRAW_CIRCLE_EXPLODE	Selects the exploding circles for drawing.
DRAW_RECT	Selects the unfilled rectangle for drawing.
DRAW_RECT_IMPLODE	Selects the imploding rectangle for drawing.
DRAW_RECT_EXPLODE	Selects the exploding rectangle for drawing.
DRAW_STAR	Selects the star for drawing.
DRAW_STAR_IMPLODE	Selects the imploding star for drawing.
DRAW_STAR_EXPLODE	Selects the exploding star for drawing.
DRAW_DIAMOND	Selects the diamond shape for drawing.
DRAW_DIAMOND_IMPLODE	Selects the imploding diamond for drawing.
DRAW_DIAMOND_EXPLODE	Selects the exploding diamond for drawing.
FILL_CIRCLE	Selects the filled circle for drawing.
FILL_CIRCLE_IMPLODE	Selects the filled imploding circle for drawing.
FILL_CIRCLE_EXPLODE	Selects the filled exploding circle for drawing.
FILL_RECT	Selects the filled rectangle for drawing.
FILL_RECT_IMPLODE	Selects the filled imploding rectangle for drawing.
FILL_RECT_EXPLODE	Selects the filled exploding rectangle for drawing.
FILL_DIAMOND	Selects the filled diamond shape for drawing.
FILL_DIAMOND_IMPLODE	Selects the filled imploding diamond for drawing.
FILL_DIAMOND_EXPLODE	Selects the filled exploding diamond shape for drawing.
DRAW_RANDOM	Selects random shapes for drawing.

Part



VI

6 Storage Place Macro

6.1 Overview

Introduction

Storage Place Macros can be used to manipulate effects located on a Storage Place in MADRIX. In this way, **special functions** can be used to change the outcome and settings. These macros are written in MADRIX Script. Please note that such a macro has effect upon all layers of the effect.

Storage Place Macros are stored together with a MADRIX Setup file. Moreover, it is possible to save macros as separate files. The file extension of a macro is *.mms*. The extension of a compiled macro is *.mcm*.



The button to call up the Storage Place Macro can be found between the **'Description Field'** and the **'Pause'** button.

Functions Called By MADRIX

There are several functions called by MADRIX in order to let the macro react to different events.

- `void InitEffect()`
- `void PreRenderEffect()`
- `void PostRenderEffect()`
- `void MatrixSizeChanged()`

If a function is not needed by a macro, it is not necessary to implement it. Regarding *InitEffect*, *PreRenderEffect*, and *PostRenderEffect* a message is printed out if one of them is missing. This is not an error, but only an information for the developer of the script.

InitEffect

(automatically included in a new script)

InitEffect is called by MADRIX whenever the macro needs to be initialized. This is the case after compiling and starting a new macro or when the user pressed the **"Start"** button of the **Script Editor**. A macro can assume that any global variable is initialized with 0 and that any global field is empty as long as it has not been initialized with a value.

This function is the right place to initialize global variables, reset any fields, set the speed of an effect, or whatever is necessary to (re)start the macro.

PreRenderEffect

(automatically included in a new script)

This function is called directly before *PostRenderEffect*. It may be used if the macro has to initialize any settings before an effect is rendered.

```
void InitEffect()
{
}

void PreRenderEffect()
{
    color c = {random(0, 255), random(0, 255)}; Clear(c);
}
```

This example uses the function *PreRenderEffect* to fill the matrix once after initializing a random color for this task.

PostRenderEffect

(automatically included in a new script)

This function is called after an effect has been rendered completely. Certain functions might want to be called. That could be a filter, for example. The number of calls per second depends on the currently set speed of the effect. It can be received with the help of the function *GetSpeed()* and set with the function *SetSpeed*.

MatrixSizeChanged

(automatically included in a new script)

MatrixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings or because a new map setting was set, e.g. caused by the call of a map function.

Standard Outline

When you open the Storage Place Macro Editor, the empty standard macro will look like this:

```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

void MatrixSizeChanged()
{
    InitEffect();
}
```

6.2 Functions

Specific Resources

- [Functions called by MADRIX](#)
- [Storage Place Macro: Available Functions](#)
- [Parameters](#)

General Resources

- [Keyword Search](#)
- [List Of Functions \(Alphabetical Order\)](#)
- [List Of Functions \(Grouped\)](#)
- [List Of Global Variables and Constants](#)
- [List Of Operations](#)
- [List Of Structures](#)
- [Table Of Frequencies](#)
- [Table Of Notes](#)

Available Functions

Standard Functions

It is useful to consult the [List of Functions \(Alphabetical Order\)](#) for non-specific functions.

Functions Provided By The Storage Place Macro

These functions are neither available in the MAS Script effect, nor in Macros for effects, nor in the Main Output Macro. Each effect pipeline, that is Storage Area A and Storage Area B, has its own Storage Place Macro.

Function	Description
Effect Functions (Incl. All Layers)	
float GetSpeedMaster ()	Retrieves the value of the Speed Master.
void SetSpeedMaster (float value)	Sets the value for the Speed Master. Valid values range from -10.0 to 10.0.

int GetPause ()	Retrieves the status of the pause function.
void SetPause (int state)	Sets the Storage Place to pause mode or not. Valid values are PAUSE and NOPAUSE. » Description
int GetSubMaster ()	Retrieves the current value of the sub master.
void SetSubMaster (int value)	Sets the sub master value. Valid values range from 0 to 255.
string GetDescription ()	Retrieves the currently used description of the Storage Place.
void SetDescription (string text)	Allows to name the Storage Place. The string variable text must be given in quotation marks. E.g. SetDescription("Storage Place 1");
void SetFilter (int filter)	Applies a Filter Effect (FX) to the Storage Place. Valid values for <i>filter</i> are » Filters
int GetFilter ()	Returns which Filter Effect (FX) is applied to the Storage Place.
Layer Functions (For Single Layers)	
int GetLayerCount ()	Retrieves the currently used number of layers.
int GetLayerSolo (int number)	Retrieves if the specified layer is used in solo mode. Layer indexing (<i>number</i>) starts with 0.
void SetLayerSolo (int number, int state)	Sets solo mode for a specified layer. Layer indexing (<i>number</i>) starts with 0.
int GetLayerBlind (int number)	Retrieves if the specified layer is used in blind mode. Layer indexing (<i>number</i>) starts with 0.
void SetLayerBlind (int number, int state)	Sets blind mode for a specified layer. Layer indexing (<i>number</i>) starts with 0.
int GetLayerSubMaster (int number)	Retrieves the value of the sub master of a specified layer. Layer indexing (<i>number</i>) starts with 0.
void SetLayerSubMaster (int number, int value)	Sets the value of the sub master of a specified layer. Layer indexing (<i>number</i>) starts with 0.
void EnableLayerFrameFade (int number, int enable)	Enables frame fade for a specified layer. Layer indexing (<i>number</i>) starts with 0.
int IsLayerFrameFadeEnabled (int number)	Retrieves if frame fade is enabled for the specified layer. Layer indexing (<i>number</i>) starts with 0.
int GetLayerMixMode (int number)	Retrieves the currently used mix mode of the specified layer. Possible values are described here . Layer indexing (<i>number</i>) starts with 0.
void SetLayerMixMode (int number, int mix mode)	Sets the mix mode for the specified layer. Possible values are described here . Layer indexing (<i>number</i>) starts with 0.
int GetLayerLink (int number)	Retrieves if link mode is enabled for the specified layer. Layer indexing (<i>number</i>) starts with 0.
void SetLayerLink (int number, int state)	Enables link mode for the specified layer. Layer indexing (<i>number</i>) starts with 0.
void MapLayerEffectVector (int number, float x, float y, float w, float h)	Maps the effect of the specified layer to a certain area of the matrix using relative values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> and <i>y</i> describe the coordinates. <i>w</i> and <i>h</i> describe width and height, respectively.

void MapLayerEffectPixel (int number, int x, int y, int w, int h)	Maps the effect of the specified layer to a certain area of the matrix using absolute values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> and <i>y</i> describe the coordinates. <i>w</i> and <i>h</i> describe width and height, respectively.
void GetLayerMapPixel (int number, int map[])	Retrieves the mapping settings for a specified layer using absolute values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (Pos X) • map[1] = y coordinate (Pos Y) • map[2] = width (Size X) • map[3] = height (Size Y)
void GetLayerMapVector (int number, float map[])	Retrieves the mapping settings for a specified layer using relative values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (Pos X) • map[1] = y coordinate (Pos Y) • map[2] = width (Size X) • map[3] = height (Size Y)
void MapLayerTileEffectVector (int number, float x, float y, float w, float h)	Sets the tile function for the effect of the specified layer using relative values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> and <i>y</i> describe the coordinates. <i>w</i> and <i>h</i> describe width and height, respectively.
void MapLayerTileEffectPixel (int number, int x, int y, int w, int h)	Sets the tile function for the effect of the specified layer using absolute values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> and <i>y</i> describe the coordinates. <i>w</i> and <i>h</i> describe width and height, respectively.
void SetLayerMapModeMirror (int number, int state)	Sets the mapping mode of the specified layer. Possible values are described here . Layer indexing (<i>number</i>) starts with 0.
void SetLayerMapModeTile (int number, int state)	Sets the tile mode of the specified layer. Possible values are described here . Layer indexing (<i>number</i>) starts with 0.
void GetLayerMapTileEffectVector (int number, float map[])	Retrieves the tile settings for a specified layer using relative values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (Pos X) • map[1] = y coordinate (Pos Y) • map[2] = width (Size X) • map[3] = height (Size Y)
void GetLayerMapTileEffectPixel (int number, int map[])	Retrieves the tile settings for a specified layer using absolute values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (Pos X) • map[1] = y coordinate (Pos Y) • map[2] = width (Size X) • map[3] = height (Size Y)
int IsLayerMapped (int number)	Retrieves if the specified layer is mapped. Layer indexing (<i>number</i>) starts with 0.
int GetLayerMapModeMirror (int number)	Retrieves the currently used mirror mode of the specified layer. Layer indexing (<i>number</i>) starts with 0.
int GetLayerMapModeTile (int number)	Retrieves the currently used tile mode of the specified layer. Layer indexing (<i>number</i>) starts with 0.

void SetLayerVectorTileOffset (int number, float x, float y)	Sets the tile offset settings using relative values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> describes offset x, while <i>y</i> represents offset y.
void SetLayerPixelTileOffset (int number, int x, int y)	Sets the tile offset settings using absolute values. Layer indexing (<i>number</i>) starts with 0. <i>x</i> describes offset x, while <i>y</i> represents offset y.
void GetLayerVectorTileOffset (int number, float map[])	Retrieves the tile offset settings using relative values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (offset x) • map[1] = y coordinate (offset y)
void GetLayerPixelTileOffset (int number, int map[])	Retrieves the tile offset settings using absolute values. Layer indexing (<i>number</i>) starts with 0. The values are saved in a field (<i>map[]</i>). <ul style="list-style-type: none"> • map[0] = x coordinate (offset x) • map[1] = y coordinate (offset y)
void SetLayerVectorMapRotation (int number, float value, int anim)	Sets the rotation value and status of the specified layer using relative values. Layer indexing (<i>number</i>) starts with 0. <i>value</i> describes the degrees. Valid values range from -200.0 to 200.0. Valid values for <i>anim</i> are MAP_ANIM_OFF and MAP_ANIM_ON. » Description
void SetLayerDegreeMapRotation (int number, int value, int anim)	Sets the rotation value and status of the specified layer using relative values. Layer indexing (<i>number</i>) starts with 0. <i>value</i> describes the degrees. Valid values range from -720 to 720. Valid values for <i>anim</i> are MAP_ANIM_OFF and MAP_ANIM_ON. » Description
float GetLayerVectorMapRotation (int number)	Retrieves the rotation value and status of the specified layer using relative values. Layer indexing (<i>number</i>) starts with 0.
int GetLayerDegreeMapRotation (int number)	Sets the rotation value and status of the specified layer using absolute values. Layer indexing (<i>number</i>) starts with 0.
int IsLayerMapRotation (int number)	Retrieves if the layer map rotation animation is active or not. Layer indexing (<i>number</i>) starts with 0.
float GetLayerFrameId (int number)	Returns the ID of the current frame of the specified layer. » Description Layer indexing (<i>number</i>) starts with 0.
void SetLayerFrameId (int number, float value)	Sets the ID of the current frame of the specified layer. » Description Layer indexing (<i>number</i>) starts with 0.
float GetLayerFrameSteps (int number)	Returns the number of frames which are between this and the last call of the specified layer. » Description Layer indexing (<i>number</i>) starts with 0.
float GetLayerFrameCount (int number)	Retrieves the number of frames used by the specified layer. Layer indexing (<i>number</i>) starts with 0.
int GetLayerOpacity (int number)	Retrieves the opacity value of the specified layer. Layer indexing (<i>number</i>) starts with 0.
void SetLayerOpacity (int number, int value)	Sets the opacity value of the specified layer. Layer indexing (<i>number</i>) starts with 0.
void SetLayerFilter (int number, int filter)	Applies a Filter Effect (FX) to the specified layer. Layer indexing (<i>number</i>) starts with 0. Valid values for <i>filter</i> are » Filters

int GetLayerFilter (int number)	Returns to which Filter Effect (FX) is applied to the specified layer. Layer indexing (<i>number</i>) starts with 0.
--	--

Parameters

The following values must be used as parameters for different functions.

Value	Description
Pause Function	
PAUSE	Activates a Storage Place to be paused. To be used with SetPause .
NOPAUSE	Deactivates the pausing mode of a Storage Place. To be used with SetPause .
Rotation Function	
MAP_ANIM_ON	Activates the rotation animation. The layer will rotate constantly. To be used with SetLayerVectorMapRotation and SetLayerDegreeMapRotation .
MAP_ANIM_OFF	Deactivates the rotation animation. The layer will not rotate constantly. To be used with SetLayerVectorMapRotation and SetLayerDegreeMapRotation .

Part



7 Main Output Macro

7.1 Overview

Introduction

Main Output macros can be used to manipulate the final output of MADRIX. In this way, **special functions** can be used to change the outcome and settings. These macros are written in MADRIX Script.

Main Output macros are stored together with a MADRIX Setup file. Moreover, it is possible to save macros as separate files. The file extension of a macro is *.mms*. The extension of a compiled macro is *.mcm*.



The button to call up the Main Output Macro can be found between the **'Fade'** button and the **'Freeze'** button.

Functions Called By MADRIX

There are several functions called by MADRIX in order to let the macro react to different events.

- `void InitEffect()`
- `void PreRenderEffect()`
- `void PostRenderEffect()`
- `void MatrixSizeChanged()`

If a function is not needed by a script, it is not necessary to implement it. Regarding *InitEffect*, *PreRenderEffect*, and *PostRenderEffect* a message is printed out if one of them is missing. This is not an error, but only an information for the developer of the script.

InitEffect

(automatically included in a new script)

InitEffect is called by MADRIX whenever the macro needs to be initialized. This is the case after compiling and starting a new macro or when the user pressed the **"Start"** button of the **Script Editor**. A macro can assume that any global variable is initialized with 0 and that any global field is empty as long as it has not been initialized with a value.

This function is the right place to initialize global variables, reset any fields, set the speed of an effect, or whatever is necessary to (re)start the macro.

PreRenderEffect

(automatically included in a new script)

This function is called directly before *PostRenderEffect*. It may be used if the macro has to initialize any settings before an effect is rendered.

```
void InitEffect()
{
}
void PreRenderEffect()
{
    color c = {random(0, 255), random(0, 255)}; Clear(c);
}
```

This example uses the function *PreRenderEffect* to fill the matrix once after initializing a random color for this task.

PostRenderEffect

(automatically included in a new script)

This function is called after an effect has been rendered completely. Certain functions might want to be called. That could be a filter, for example. The number of calls per second depends on the currently set speed of the effect. It can be received with the help of the function *GetSpeed()* and set with the function *SetSpeed*.

MatrixSizeChanged

(automatically included in a new script)

MatrixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings or because a new map setting was set, e.g. caused by the call of a map function.

Standard Outline

When you open the Main Output Macro Editor, the empty standard macro will look like this::

```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

void MatrixSizeChanged()
{
    InitEffect();
}
```

7.2 Functions

Specific Resources

- [Functions called by MADRIX](#)
- [Main Output Macro: Available Functions](#)
- [Parameters](#)

General Resources

- [Keyword Search](#)
- [List Of Functions \(Alphabetical Order\)](#)
- [List Of Functions \(Grouped\)](#)
- [List Of Global Variables and Constants](#)
- [List Of Operations](#)
- [List Of Structures](#)
- [Table Of Frequencies](#)
- [Table Of Notes](#)

Available Functions

Standard Functions

It is useful to consult the [List of Functions \(Alphabetical Order\)](#) for non-specific functions.

The Main Output macro offers special functions to reflect the options available to the user in the user interface of MADRIX, like the Freeze function or the Cue List.

Functions Provided By The Main Output Macro

These functions are not available in the MAS Script effect, in the Macros for effects, and not in the Storage Place Macro.

Function	Description
int GetFadeType ()	Returns the current fade type. » Example

void SetFadeType (int fadetype)	Set the type of fade. Valid values are CROSSFADE , WHITEFADE , BLACKFADE , COLORFADE . »Example
color GetFadeColor ()	Returns the color that is selected for the color-fade. »Example
void SetFadeColor (color fadecolor)	Set the color of the color-fade. »Example
float GetFadeTime ()	Returns the time in seconds that is used to fade between A and B. »Example
void SetFadeTime (float fadetime)	Set the fade time in seconds. Valid values range from 0 to 3600.0 »Example
int GetFadeValue ()	Returns the value/position of the fader. »Example
void SetFadeValue (int fadervalue)	Set the value for the position of the fader. Valid values range from 0 to 255. »Example
int GetFreeze ()	Returns the status of the freeze function. »Example
void SetFreeze (int freeze/unfreeze)	Set the freeze function to be activated or deactivated. »Example
int GetMasterFader ()	Returns the value/position of the Master Fader, which determines the brightness of the main output. »Example
void SetMasterFader (int masterfadervalue)	Set the value of the Master Fader. Valid values range from 0 to 255. »Example
int GetAudioFader ()	Returns the value/position of the audio fader, which determines the audio input level. »Example
void SetAudioFader (int audiofadervalue)	Set the value of the audio fader. Valid values range from 0 to 255. »Example
void CuelistStop ()	Stop the Cue List. »Example
void CuelistPlay ()	Start the Cue List. Useful when a duration is set. Otherwise, 'CuelistGo' is performed. »Example
void CuelistGo ()	Jump ahead one step in the Cue List. »Example
void CuelistBack ()	Go back one step in the Cue List. »Example
void CuelistGoto (int cuelistentry)	Go to a specific Cue List entry. »Example
int CuelistCurrentCue ()	Returns the currently used Cue in the Cue List. A return value of -1 means that the Cue List is not running. A return value of 0 means that Cue 1 is active, a value of 1 means that Cue 2 is active, and so on. »Example
int CuelistCount ()	Returns the total number of Cues in the Cue List.
int GetStorageSpeedMaster (int storage)	Returns the value of the Speed Master for Storage A or B. Valid values are STORAGE_A , STORAGE_B , and a speed range from -10.0 to 10.0 for the return value. »Example
void SetStorageSpeedMaster (int storage, float speed)	Set the value of the Speed Master for a specific Storage Place. Valid values are STORAGE_A , STORAGE_B . »Example
int GetStoragePause (int storage)	Returns the status of the pause function of Storage A or B. Valid values are STORAGE_A , STORAGE_B . »Example
void SetStoragePause (int storage, int pause/nopause)	Set the pause mode to be activated or deactivated for Storage A or B. Valid values are STORAGE_A , STORAGE_B for <i>storage</i> as well as PAUSE , NOPAUSE . »Example
void GetStoragePlace (int storage)	Returns the currently selected Storage Place of Storage A or B with a value between 0 and 59. Valid values are STORAGE_A , STORAGE_B . »Example

void SetStoragePlace (int storage, int place, int autofade/noautofade)	Set a specific Storage Place to be activated/selected in Storage A or B. Valid values are STORAGE_A , STORAGE_B for <i>storage</i> , and a <i>place</i> between 0 and 59. The third value is optional and valid parameters are: WITH_AUTOFADE , WITHOUT_AUTOFADE . Activated by default is autofade. (When the currently selected Storage Place can be seen in the output window and a new Storage Place without autofade is set with this function, autofade will still be used (without_autofade will be overwritten) to provide a smooth fade on the output.) » Example
int GetStorageSubMaster (int storage)	Returns the value of the submaster of Storage A or B. Valid values are STORAGE_A , STORAGE_B . » Example
void SetStorageSubMaster (int storage, int submastervalue)	Returns the currently selected Storage Place of Storage A or B. Valid values for <i>storage</i> are STORAGE_A , STORAGE_B , and a value range from 0 to 255 for the <i>submastervalue</i> . » Example
int GetStorageFullState (int storage, int place)	Returns 1 if the standard settings of a Storage Place of either Storage A or B have been modified, i.e. if an effect has been set up on a Storage Place. Valid values for <i>storage</i> are STORAGE_A , STORAGE_B . <i>place</i> describes the Storage Place. Returns 0 if the specified Storage Place remains unchanged.
void SetFilterColor (color c)	Sets the color that will be used for the color filter of the main output. » Using Colors
color GetFilterColor ()	Retrieves the color that is currently set for the main output color filter. » Using Colors
void SetBlackout (int mode)	Activates or deactivates a black out of the output. Valid values for mode are BLACKOUT_OFF or BLACKOUT_ON .
int GetBlackout ()	Returns if the black out is active or not.
void SetStorageFilter (int storage, int filter)	Applies a Filter Effect (FX) to the Storage Place. Valid values for <i>storage</i> are STORAGE_A , STORAGE_B . Valid values for <i>filter</i> are » Filters
int GetStorageFilter ()	Returns which Filter Effect (FX) is applied to the Storage Place.
void ImportStoragePlace (int storage, int place, string name)	Automatically loads a single MADRIX Storage Place. Valid values for <i>storage</i> are STORAGE_A , STORAGE_B . Valid values for <i>place</i> are 0 to 59 (Indexing starts with 0). Valid values for <i>name</i> are *.mef file names and the specific location on the harddisk; for example: "C :\\Effect.mef". It is recommended to only use this function in <i>InitEffect()</i> .
void ImportStorage (int storage, string name)	Automatically loads a complete Storage Area. Valid values for <i>storage</i> are STORAGE_A , STORAGE_B . Valid values for <i>name</i> are *.mss file names and the specific location on the harddisk; for example: "C :\\Storage.mss". It is recommended to only use this function in <i>InitEffect()</i> .
int ImportPatch (string file)	Automatically loads a Patch file. Valid values for <i>file</i> are *.mpf file names and the specific location on the harddisk in inverted commas; for example: "C :\\Patch01.mpf". The function returns an integer value. Valid values for <i>int</i> are 1 (loaded successfully) and 0 (failed to load). It is recommended to only use this function in <i>InitEffect()</i> . Please remove <i>InitEffect()</i> from <i>MatrixSizeChanged()</i> in this case. If used in other parts of the Macro, <i>MatrixSizeChanged()</i> should be included in <i>InitEffect()</i> .
void Filter (int filter)	Renders a filter over the matrix. » Valid parameters (Filters) » Description

Parameters

The following values (defines) must be used as parameters for the functions provided above.

Value	Description
Fade Types	
CROSSFADE	Selects the Cross-fade. To be used with SetFadeType .
WHITEFADE	Selects the White-fade. To be used with SetFadeType .
BLACKFADE	Selects the Black-fade. To be used with SetFadeType .
COLORFADE	Selects the Color-fade with the currently selected color. To be used with SetFadeType .
HWIPEFADE	Selects the Horizontal Wipe as fade type. To be used with SetFadeType .
VWIPEFADE	Selects the Vertical Wipe as fade type. To be used with SetFadeType .
HXWIPEFADE	Selects the Horizontal Cross Wipe as fade type. To be used with SetFadeType .
VXWIPEFADE	Selects the Vertical Cross Wipe as fade type. To be used with SetFadeType .
HSLIDEFADE	Selects the Horizontal Slide as fade type. To be used with SetFadeType .
VSLIDEFADE	Selects the Vertical Slide as fade type. To be used with SetFadeType .
HXSLIDEFADE	Selects the Horizontal Cross Slide as fade type. To be used with SetFadeType .
VXSLIDEFADE	Selects the Vertical Cross Slide as fade type. To be used with SetFadeType .
Freeze Function	
FREEZE	Freezes the main output instantly. To be used with SetFreeze .
UNFREEZE	Unfreezes the main output. To be used with SetFreeze .
Storage Area Selection	
STORAGE_A	Selects Storage Area A. To be used with GetStorageSpeedMaster , SetStorageSpeedMaster , GetStoragePause , SetStoragePause , GetStoragePlace , SetStoragePlace , GetStorageSubMaster , SetStorageSubMaster , GetStorageFullState , SetStorageFilter , ImportStoragePlace , and ImportStorage .
STORAGE_B	Selects Storage Area B. To be used with GetStorageSpeedMaster , SetStorageSpeedMaster , GetStoragePause , SetStoragePause , GetStoragePlace , SetStoragePlace , GetStorageSubMaster , SetStorageSubMaster , GetStorageFullState , SetStorageFilter , ImportStoragePlace , and ImportStorage .
Autofade Function	
WITH_AUTOFADE	Activates autofade. To be used with SetStoragePlace .
WITHOUT_AUTOFADE	Deactivates autofade. To be used with SetStoragePlace .
Pause Function	
PAUSE	Activates a Storage Place to be paused. To be used with SetStoragePause .
NOPAUSE	Deactivates the pausing mode of a Storage Place. To be used with SetStoragePause .

7.3 Examples

GetFadeType

Select a different fade type while running the script and monitor the 'Script output' to see the result.

```
@scriptname="get fade type test";
@author="";
@version="";
@description="";

void InitEffect()
{
}

void PreRenderEffect()
{
    switch(GetFadeType())
    {
        case CROSSFADE: WriteText("CrossFade");break;
        case WHITEFADE: WriteText("WhiteFade");break;
        case BLACKFADE: WriteText("BlackFade");break;
        case COLORFADE: WriteText("ColorFade");break;
    }
}

void PostRenderEffect()
{
}
```

(Description)

SetFadeType

Watch the fade area while running the script to see how the fade type is switched.

```
@scriptname="set fade type test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{
}

void PreRenderEffect()
{
    Value%=200;// range 0...200
    switch(Value)
```

```
{
    case 0: SetFadeType(CROSSFADE);break;
    case 50: SetFadeType(WHITEFADE);break;
    case 100: SetFadeType(BLACKFADE);break;
    case 150: SetFadeType(COLORFADE);break;
}
Value++;
}
```

```
void PostRenderEffect()
{}
```

(Description)

GetFadeColor

Select a different color for the color-fade while running the script and monitor the 'Script output' to see the result.

```
@scriptname="get fade color values";
@author="";
@version="";
@description="";

color c;
void InitEffect()
{}
```

```
void PreRenderEffect()
{
    c=GetFadeColor();
    WriteText("Red: "+(string)c.r+", Green: "+(string)c.g+", Blue: "+(string)c.b+",
        White:"+ (string)c.w);
}
```

```
void PostRenderEffect()
{}
```

(Description)

SetFadeColor

Monitor the color window next to the color-fade button to see the result.

```
@scriptname="set fade color red value";
@author="";
@version="";
@description="";

color c;
```

```
void InitEffect()  
{  
  
void PreRenderEffect()  
{  
    c.r++;  
    c.r%=256;// range 0...255  
    SetFadeColor(c);  
}  
  
void PostRenderEffect()  
{  
}
```

(Description)

GetFadeTime

Increase or decrease the fade time and monitor the 'Script output' while running the script.

```
@scriptname="get fade time";  
@author="";  
@version="";  
@description="";  
  
float fTime;  
void InitEffect()  
{  
  
void PreRenderEffect()  
{  
    fTime=GetFadeTime();  
    WriteText("FadeTime: "+(string)fTime);  
}  
  
void PostRenderEffect()  
{  
}
```

(Description)

SetFadeTime

Watch the fade time box while running the script.

```
@scriptname="set fade time";  
@author="";  
@version="";  
@description="";  
  
float fTime=0.0;  
void InitEffect()  
{  
}
```

```
void PreRenderEffect()  
{  
    fTime+=0.1;  
    if(fTime>60.0)  
        fTime=0.0;// range 0...60  
    SetFadeTime(fTime);  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetFadeValue

Slide the fader from A to B or back and monitor the 'Script output' while running the script.

```
@scriptname="get fader value A-B "  
@author="";  
@version="";  
@description="";  
  
int fValue;  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    fValue=GetFadeValue();  
    WriteText((string)fValue);  
}  
  
void PostRenderEffect()  
{}
```

(Description)

SetFadeValue

Watch the fader closely to see the result of the script.

```
@scriptname="set fader value A-B";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}
```

```
void PreRenderEffect()  
{  
    Value++;  
    Value%=256;// range 0...255  
    SetFadeValue(Value);  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetFreeze

To test this script, use the 'Freeze' button and monitor the 'Script output' while running the script.

```
@scriptname="get freeze button test";  
@author="";  
@version="";  
@description="";  
  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    if(GetFreeze())  
        WriteText("Freeze button is pressed");  
    else  
        WriteText("Freeze button is not pressed");  
}  
  
void PostRenderEffect()  
{}
```

(Description)

SetFreeze

Watch the 'Freeze' button to see the effect of the script.

```
@scriptname="freeze, unfreeze test";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}
```

```
void PreRenderEffect()  
{  
    Value++;  
    Value%=100;// range 0...100  
    switch(Value)  
    {  
        case 5: SetFreeze(FREEZE);break;  
        case 55: SetFreeze(UNFREEZE);break;  
    }  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetMasterFader

Adjust the Master Fader while running the script and monitor the 'Script output'.

```
@scriptname="get fader value from master";  
@author="";  
@version="";  
@description="";  
  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    WriteText("MasterFader: "+(string)GetMasterFader());  
}  
  
void PostRenderEffect()  
{}
```

(Description)

SetMasterFader

Watch the Master Fader closely to see the result of the script.

```
@scriptname="set master fader value";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}
```



```
void PreRenderEffect()  
{  
    Value++;  
    Value%=256;// range 0...255  
    SetMasterFader(Value);  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetAudioFader

Adjust the audio fader while running the script and monitor the 'Script output'.

```
@scriptname="get fader value from audio";  
@author="";  
@version="";  
@description="";  
  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    WriteText("AudioFader: "+(string)GetAudioFader());  
}  
  
void PostRenderEffect()  
{}
```

(Description)

SetAudioFader

Monitor the audio fader closely to see the result of the script.

```
@scriptname="set audio fader";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    Value++;  
    Value%=256;// range 0...255
```

```
    SetAudioFader(Value);  
}  
  
void PostRenderEffect()  
{}
```

(Description)

CuelistStop/CuelistPlay

To test this script, a playlist with duration and minimal 2 Cue List entries is required.

```
@scriptname="cue list stop/start test";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
    Value++;  
    Value%=500;// range 0...500  
    switch(Value)  
    {  
        case 10: CuelistStop();break;  
        case 260: CuelistPlay();break;  
    }  
}  
  
void PostRenderEffect()  
{}
```

(Description)

CuelistGo/CuelistBack

To test this script, a playlist with duration and minimal 2 Cue List entries is required.

```
@scriptname="cue list go/back test";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}  
  
void PreRenderEffect()
```

```
{  
  
    Value++;  
    Value%=500;// range 0...500  
    switch(Value)  
    {  
        case 10: CuelistGo();break;  
        case 260: CuelistBack();break;  
    }  
}  
  
void PostRenderEffect()  
{}
```

(Description)

CuelistGoto

To test this script, a playlist without duration and minimal 9 Cue List entries is required.

```
@scriptname="cue list goto test";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
void InitEffect()  
{}  
  
void PreRenderEffect()  
{  
  
    Value++;  
    Value%=250;// range 0...250  
    if(Value%50==0)  
        CuelistGoto(Value/25);    // skip 1,3,5,7,9,1,3,...  
        //CuelistGoto(Value/50);    // skip 1,2,3,4,5,6,1,2,...  
}  
  
void PostRenderEffect()  
{}
```

(Description)

CuelistCurrentCue

To test this script, create a Cue List, activate several Cues, and monitor the 'Script output'.

```
@scriptname="CurrentCue";  
@author="jky / info@madrix.com";  
@version="1.0 Madrix 2.8a";
```

```
@description="View current item from Cue List";

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    WriteText((string)CuelistCurrentCue());
}
```

(Description)

GetStorageSpeedMaster

Adjust the Speed Master of Storage area A or B while running the script and monitor the 'Script output'.

```
@scriptname="get Speed Master test";
@author="";
@version="";
@description="";

void InitEffect()
{

}

void PreRenderEffect()
{
    WriteText("SpeedA: "+(string)GetStorageSpeedMaster(STORAGE_A)
    +" SpeedB: "+(string)GetStorageSpeedMaster(STORAGE_B));
}

void PostRenderEffect()
{

}
```

(Description)

SetStorageSpeedMaster

This script is best tested with the SCE Colorscroll effect. Monitor Preview A and B as well as the Speed Master faders to see the results of the script.

```
@scriptname="Speed Master test";
@author="";
@version="";
@description="";

float Value=0.0;
```

```
void InitEffect()  
{  
  
void PreRenderEffect()  
{  
  
    Value+=0.1;  
    if(Value>10.0)  
        Value=-10.0;// range 0...255  
  
    SetStorageSpeedMaster(STORAGE_A,Value);// storage A  
    SetStorageSpeedMaster(STORAGE_B,Value/2.0);// storage B  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetStoragePause

To test this script, press the 'Pause' button of Storage area A or B and monitor the 'Script output'.

```
@scriptname="get storage pause test";  
@author="";  
@version="";  
@description="";  
  
void InitEffect()  
{  
  
void PreRenderEffect()  
{  
    if(GetStoragePause(STORAGE_A))  
        WriteText("Storage A paused");  
    else  
        WriteText("Storage A running");  
    if(GetStoragePause(STORAGE_B))  
        WriteText("Storage B paused");  
    else  
        WriteText("Storage B running");  
}  
  
void PostRenderEffect()  
{}
```

(Description)

SetStoragePause

This script is best tested with the SCE Colorscroll effect. Monitor the 'Pause' buttons of Storage A and B to see the effects of the script.

```
@scriptname="storage no-/pause test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{
    Value++;
    Value%=200;// range 0...200

    switch(Value)
    {
        case 10: SetStoragePause(STORAGE_A,PAUSE);break;// storage A
        case 60: SetStoragePause(STORAGE_B,PAUSE);break;// storage B
        case 110: SetStoragePause(STORAGE_A,NOPAUSE);break;// storage A
        case 160: SetStoragePause(STORAGE_B,NOPAUSE);break;// storage B
    }
}

void PostRenderEffect()
{}

```

(Description)

GetStoragePlace

While running the script, select different Storage Places across Storage A or B and monitor the 'Script output'. The currently selected Storage Place number minus 1 should be displayed.

```
@scriptname="get storageplace test";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
    WriteText("Storage A:"+(string)GetStoragePlace(STORAGE_A)
    +" Storage B:"+(string)GetStoragePlace(STORAGE_B));
}

```

```
void PostRenderEffect()  
{}
```

(Description)

SetStoragePlace

Monitor the effect sections A and B as well as the two Storage Areas (A and B) to see the results of the script.

```
@scriptname="test setStoragePlace with and without autofade";  
@author="";  
@version="";  
@description="";  
  
int Value=0;  
int place=0;  
void InitEffect()  
{  
    SetFadeTime(1.0); // better for demonstration  
}  
  
void PreRenderEffect()  
{  
    Value++;  
    Value%=1000; // range 0...1000  
    switch(Value)  
    {  
        case 150: SetStoragePlace(STORAGE_A,place,WITH_AUTOFADE);break;  
        // storage A with autofade  
        case 300: SetStoragePlace(STORAGE_B,place,WITH_AUTOFADE);break;  
        // storage B with autofade  
        case 450: place++;  
        case 600: SetStoragePlace(STORAGE_A,place,WITHOUT_AUTOFADE);break;  
        // storage A without autofade, if FaderValue not an place A  
        case 750: SetStoragePlace(STORAGE_B,place,WITHOUT_AUTOFADE);break;  
        // storage B without autofade, if FaderValue not an place B  
        case 900: place++;  
    }  
    if(place>59) // place range 0...59  
        place=0;  
}  
  
void PostRenderEffect()  
{}
```

(Description)

GetStorageSubMaster

Adjust the submasters of Storage A and B ('Sub' button) and monitor the 'Script output' while running the script.

```
@scriptname="storage submaster get test";
@author="";
@version="";
@description="";

int subA;
int subB;
void InitEffect()
{}

void PreRenderEffect()
{
    subA=GetStorageSubMaster(STORAGE_A);// storage A
    subB=GetStorageSubMaster(STORAGE_B);// storage B
    WriteText("Submaster A:" +(string)subA +" , Submaster B:" +(string)subB);
}

void PostRenderEffect()
{
}
```

(Description)

SetStorageSubMaster

Select a SCE effect in both effect areas and monitor the two previews (Preview A and B) while running the script.

```
@scriptname="storage submaster set test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{
    Value++;
    Value%=256;// range 0...255

    SetStorageSubMaster(STORAGE_A,Value);// storage A
    SetStorageSubMaster(STORAGE_B,255-Value);// storage B
    WriteText("SubA: "+(string)Value+" , SubB: "+(string)(255-Value));
}
```



```
void PostRenderEffect()  
{}
```

(Description)

Part



8 Imprint & Copyright

Company And Address



inoage GmbH

Wiener Straße 56

D-01219 Dresden

Germany

Managing Directors: Christian Hertel, Sebastian Pinzer, Sebastian Wissmann

Phone: +49 351 482 056 30

Fax: +49 351 482 056 31

WWW: <http://www.madrix.com>

E-mail: info@madrix.com

Copyright

MADRIX is a trademark of inoage GmbH.

All other company names and/or product names are trademarks and/or entered trademarks of their respective holders. The product might not always be conforming to the presentation, features, and performances. Technical data can differ slightly, depending on the operating system and the chosen hardware.

We withhold the option of changes without notification. inoage GmbH does not give any guaranty for function capability for a certain purpose, the marked ability or other features of the product. No other guaranty claims, on legal or other terms, can be enforced.

Under no circumstances does inoage GmbH take on the responsibility for liabilities for faults for losses in sales volume or profits, that occur through the usage of the product, through the serviceability, through abuse, happenings, circumstances or actions, that we have no influence on. No matter if the damages were caused by the holder of the product or a third person.

Copyright (c) 2001 - 2014 inoage GmbH. All rights reserved.

Index

- - -

- 62, 161
-- 62, 161

- ! -

! 62, 161
!= 56, 62, 161

- % -

% 62, 161
%= 62, 161

- & -

&& 62, 161

- * -

* 62, 161
*.macs 185
*.mas 185
*.mcm 220, 322, 331
*.mms 220, 322, 331
*/ 82
*= 62, 161

- . -

. operator 50

- / -

/ 62, 161
/* 82
// 82
/= 62, 161

- @ -

@author 82, 155
@description 82, 155
@name 82, 155
@version 82, 155

- [-

[] operator 53

- { -

{ 67

- | -

|| 62, 161

- } -

} 67

- + -

+ 62, 161
++ 62, 161
+= 62, 161

- < -

< 56, 62, 161
<= 56, 62, 161

- = -

= 161

- - -

-= 62, 161

- = -

== 56, 62, 161

- > -

> 56, 62, 161

>= 56, 62, 161

- 4 -

43 Hz 209

- 5 -

50 Hz 209

- A -

A 170

A# 170

abs 128, 144, 147

absolute coordinates 83

AddColor 85, 232, 234, 238, 240, 243, 246, 250, 254, 258, 263, 270, 274, 287, 292, 300, 305, 316

AddPixelTransposeEntry 128, 144

AddSparkleColor 246

ALPHA 155

AQUA 92, 155

arccos 128, 144, 147

arccosDeg 128, 144, 147

arcsin 128, 144, 147

arcsinDeg 128, 144, 147

arctan 128, 144, 147

arctanDeg 128, 144, 147

assignment 62

author 82

- B -

backward 209

BLACK 92, 155

BLACK_ALPHA 155

BLACKFADE 334

block 67, 68

BLUE 92, 155

bool 50

bpm 209, 224

break 74, 128

BROWN 155

Button 190

- C -

C 170

C# 170

calling convention 83

case 56, 71, 128

ceil 128, 144

ChangeBrightness 128, 144

CheckScriptEngineVersion 128, 144

CheckSoftwareVersion 128, 144

CHM_LEFT 305, 307

CHM_MONO 305, 307

CHM_RIGHT 305, 307

CHM_STEREO 305, 307

Clear 128, 144

Clear Compiler Messages 34

Clear Messages 34

ClearAlpha 128, 144

ClearColor 128, 144

CM_LOOP 243, 246, 254, 258, 270, 274, 292, 316

CM_RANDOM 243, 246, 254, 258, 270, 274, 292, 316

CM_SHUFFLE 243, 246, 254, 258, 270, 274, 292, 316

col.r 50

color 50, 52, 155, 162

Color Picker 85

Color Ramp 85

Color table 85, 190

COLOR_AND_ALPHA 155

COLOR_BG 250

COLOR_BR 250

COLOR_GB 250

COLOR_GR 250

COLOR_RB 250

COLOR_RG 250

COLORFADE 334

ColorReplace 128, 144

colors 39, 92

Comment 82

compare operation 62

compare strings 56
 Compile 34
 Compiled scripts 34
 concatenate string 62
 const 47, 128
 Constant 47
 continue 74, 128
 control loop 74
 conversion 52
 cos 128, 144, 147
 cosDeg 128, 144, 147
 cosH 128, 144, 147
 cosHDeg 128, 144, 147
 CreatePixelTransposeTable 128, 144
 CROSSFADE 334
 ctrlbutton 190
 ctrlbutton2 190
 ctrlbutton3 190
 ctrlbutton4 190
 ctrlcolor 190
 ctrlcolortable 190
 ctrledit 190
 ctrledit2 190
 ctrledit3 190
 ctrledit4 190
 ctrlslider 190
 CuelistBack 334, 339
 CuelistCount 334
 CuelistCurrentCue 334
 CuelistGo 334, 339
 CuelistGoto 334, 339
 CuelistPlay 334, 339
 CuelistStop 334, 339
 curly bracket 67
 CURVE_ABSSIN 265
 CURVE_PHASE 265
 CURVE_SAWTOOTH_DOWN 265
 CURVE_SAWTOOTH_UP 265
 CURVE_SIN 265
 CURVE_TRIANGLE 265
 CYAN 155

- D -

D 170
 D# 170
 DARK_GRAY 155
 Data type 50, 52

Data types 46
 date 50, 52, 162
 Declaration 47
 default 71, 128
 deg2rad 128, 144, 147
 description 82
 Detonate 246
 Dim 128, 144
 DimPixel 128, 144
 DimPixelArea 128, 144
 DIR_CIRCLE_INWARDS 238
 DIR_CIRCLE_OUTWARDS 238
 DIR_DL 155
 DIR_DOWN 155, 250, 287, 305
 DIR_DR 155
 DIR_HORIZONTAL 296, 314
 DIR_HV 296, 314
 DIR_INWARDS 238, 265, 274, 302, 309
 DIR_INWARDS_CIRCLE 240, 310
 DIR_INWARDS_ELLIPSE 240, 310
 DIR_INWARDS_H 240, 310
 DIR_INWARDS_RECT 240, 310
 DIR_INWARDS_SQUARE 240, 310
 DIR_INWARDS_V 240, 310
 DIR_LEFT 155, 250, 287, 305
 DIR_NONE 155
 DIR_OUTWARDS 238, 265, 274, 302, 309
 DIR_OUTWARDS_CIRCLE 240, 310
 DIR_OUTWARDS_ELLIPSE 240, 310
 DIR_OUTWARDS_H 240, 310
 DIR_OUTWARDS_RECT 240, 310
 DIR_OUTWARDS_SQUARE 240, 310
 DIR_OUTWARDS_V 240, 310
 DIR_RADIAL 238
 DIR_RIGHT 155, 250, 287, 305
 DIR_UL 155
 DIR_UP 155, 250, 287, 305
 DIR_UR 155
 DIR_VERTICAL 296, 314
 DoPreRender 42, 185, 188
 Double-quoted strings 71
 draw 83
 DRAW_CIRCLE 234, 243, 246, 254, 258, 270, 274, 292, 302, 316
 DRAW_CIRCLE_EXPLODE 270, 292, 316
 DRAW_CIRCLE_IMPLODE 270, 292, 316
 DRAW_CROSS 234, 243, 246, 254, 270, 274, 292, 302, 316

DRAW_CROSS_EXPLODE 270, 292, 316
 DRAW_CROSS_IMPLODE 270, 292, 316
 DRAW_DIAMOND 234, 243, 246, 254, 258, 270, 274, 292, 316
 DRAW_DIAMOND_EXPLODE 270, 292, 316
 DRAW_DIAMOND_IMPLODE 270, 292, 316
 DRAW_LINE 234, 243
 DRAW_RANDOM 234, 243, 246, 254, 270, 274, 292, 316
 DRAW_RECT 234, 243, 246, 254, 258, 270, 274, 292, 302, 316
 DRAW_RECT_EXPLODE 270, 292, 316
 DRAW_RECT_IMPLODE 270, 292, 316
 DRAW_STAR 234, 243, 246, 254, 270, 274, 292, 302, 316
 DRAW_STAR_EXPLODE 270, 292, 316
 DRAW_STAR_IMPLODE 270, 292, 316
 DRAW_TEXT 254
 DrawPixelArea 97, 128, 144
 DrawPixelCircle 128, 144
 DrawPixelCross 128, 144
 DrawPixelDiamond 128, 144
 DrawPixelEllipse 128, 144
 DrawPixelLine 128, 144
 DrawPixelRect 128, 144
 DrawPixelStar 128, 144
 DrawPixelText 128, 144, 334
 DrawVectorCircle 128, 144
 DrawVectorCross 128, 144
 DrawVectorDiamond 128, 144
 DrawVectorEllipse 128, 144
 DrawVectorLine 128, 144
 DrawVectorRect 128, 144
 DrawVectorStar 128, 144
 DrawVectorText 128, 144, 334
 dynamic memory 53

- E -

E 170
 Edit 190
 EFFECT_ABS_SINE 285
 EFFECT_CIRCLE 265
 EFFECT_HELIX 265
 EFFECT_RADAR 265
 EFFECT_SAWTOOTH_DOWN 285
 EFFECT_SAWTOOTH_UP 285
 EFFECT_SINE_WAVE 285

EFFECT_TRIANGLE 285
 else 68, 128
 else if 68
 EnableFrameFade 128, 144
 EnableLayerFrameFade 325
 endswith 56, 128, 144
 Examples 171
 ExecutePixelTranspose 128, 144
 Exemplary Scripts 171
 exp 128, 144, 147
 EXPLOSION_SHAPE_DIAMOND 246
 EXPLOSION_SHAPE_RADIAL 246
 EXPLOSION_SHAPE_RANDOM 246
 EXPLOSION_SHAPE_SPHERE 246
 EXPLOSION_SHAPE_SPHERE_GLOW 246
 EXPLOSION_SHAPE_SPIRAL 246
 EXPLOSION_SHAPE_STAR 246
 expression 62
 Extension 185, 220
 Extract 56

- F -

F 170
 F# 170
 FadeAllColors 85, 238, 250, 287, 300
 FadeNoneColors 85, 238, 250, 287, 300
 FaderBox 85
 FALSE 50, 128
 Faster script effects 209
 field 42, 53
 File extension 185, 220
 File operations 78
 fill 83
 FILL_CIRCLE 234, 243, 246, 254, 258, 270, 274, 292, 302, 316
 FILL_CIRCLE_EXPLODE 270, 292, 316
 FILL_CIRCLE_IMPLODE 270, 292, 316
 Fill_DIAMOND 234, 243, 246, 254, 258, 270, 274, 292, 316
 FILL_DIAMOND_EXPLODE 270, 292, 316
 FILL_DIAMOND_IMPLODE 270, 292, 316
 FILL_RECT 234, 243, 246, 254, 258, 270, 274, 292, 302, 316
 FILL_RECT_EXPLODE 270, 292, 316
 FILL_RECT_IMPLODE 270, 292, 316
 FillPixelCircle 128, 144
 FillPixelDiamond 128, 144

FillPixelEllipse 128, 144
FillPixelRect 128, 144
FillVectorCircle 128, 144
FillVectorDiamond 128, 144
FillVectorEllipse 128, 144
FillVectorRect 128, 144
Filter 97, 128, 144
Filter(int filter) 93
FILTER_BLUE 93, 155
FILTER_BLUE_WHITE 93, 155
FILTER_BLUR 93, 155
FILTER_BLUR_BSPLINE 93, 155
FILTER_BLUR_CATMULL_ROM 93, 155
FILTER_BLUR_GAUSS 93, 155
FILTER_BLUR_MITCHELL 93, 155
FILTER_BRIGHTEN 93, 155
FILTER_DARKEN 93, 155
FILTER_EDGES 155
FILTER_EDGES_POPUP 155
FILTER_EMOSS 155
FILTER_EMOSS_POPUP 155
FILTER_GREEN 93, 155
FILTER_GREEN_BLUE 93, 155
FILTER_GREEN_BLUE_WHITE 155
FILTER_GREEN_WHITE 93, 155
FILTER_GREYSCALE 93, 155
FILTER_INVERT_COLOR 93, 155
FILTER_INVERT_H_MATRIX 155
FILTER_INVERT_HV_MATRIX 155
FILTER_INVERT_V_MATRIX 155
FILTER_NONE 93, 155
FILTER_RED 93, 155
FILTER_RED_BLUE 93, 155
FILTER_RED_BLUE_WHITE 93, 155
FILTER_RED_GREEN 93, 155
FILTER_RED_GREEN_BLUE 93, 155
FILTER_RED_GREEN_WHITE 93, 155
FILTER_RED_WHITE 93, 155
FILTER_SHARPE 93, 155
FILTER_WHITE 93, 155
Filters 93
findstring 56, 128, 144
FireRocket 246
First Example 38
Fixed frame rate 216
Fixed render frequency 216
float 50, 52
Floating point frames 216

fmax 128, 144
fmin 128, 144
fmod 128, 144
font 162
for 74, 128
forward 209
fps 209
frame count 188, 213
frame ID 188, 213, 220, 223
framesteps 209, 216
frandom 128, 144, 147
FREEZE 334
Frequency 119, 170
frequency overview 164
FUCHSIA 92, 155
Function 41, 42

- G -

G 170
G# 170
GetAllIntervals 128, 144, 147
GetAllNoteValues 123, 128, 144, 147
GetAmplify 287, 291, 292, 296, 300, 302
GetAmplitude 265
GetAngle 238
GetAnimationSpeed 228
GetApplicationPath 128, 144
GetAudioFader 334, 339
GetAutostart 281
GetBandCount 287, 291, 296, 300
GetBandMode 287
GetBassTone 128, 144, 147
GetBassType 128, 144, 147
GetBassValue 128, 144, 147
GetBlackout() 334
GetBlur 246
GetBorder 258
GetBpm 188, 223, 224, 226, 228, 232, 234, 240, 243, 246, 250, 254, 258, 263, 265, 274, 276, 281, 285, 287, 291, 296, 299, 300, 302, 309, 310, 312, 314
GetCapturing 281
GetCenter 234
GetChangeTime 234
GetChannelView 305, 307
GetCircle 234
GetCollision 232

GetColor	85, 226, 232, 234, 238, 240, 243, 246, 250, 254, 258, 263, 264, 265, 270, 274, 285, 287, 291, 292, 296, 297, 299, 300, 302, 305, 307, 308, 309, 310, 312, 314, 316	GetFilterColor	228, 281, 334
GetColorCount	85, 232, 234, 238, 240, 243, 246, 250, 254, 258, 263, 270, 274, 287, 292, 300, 302, 305, 316	GetFlameSize	250
GetColorDepth	128, 144	GetFontFaceName	254, 276
GetColorFade	85, 238, 250, 263, 287, 300, 305	GetFontHeight	254, 276
GetColorMix	258	GetFontItalic	254, 276
GetColorMixLink	258	GetFontStrikeOut	254, 276
GetColorMode	243, 246, 250, 254, 258, 270, 274, 292, 316	GetFontUnderline	254, 276
GetColorPosition	85, 238, 250, 263, 287, 300, 305	GetFontWeight	254, 276
GetComputerName	128, 144	GetFontWidth	254, 276
GetContinuous	254, 276	GetFrameCount	128, 144, 188, 213
GetCount	243, 258, 265, 270, 274	GetFrameld	128, 144, 188, 213, 220
GetCrossed	238, 240, 287, 300, 310	GetFrameSteps	128, 144, 188, 209
GetCurrentImage	228	GetFreeze	334, 339
GetCurve	265	GetFrequency	254, 264
GetDate	128, 144	GetFrequencyMax	254
GetDegreeMapRotation	128, 144	GetGraphMode	254
GetDepth	274	GetGravity	246
GetDescription	325	GetGrey	228, 281
GetDirection	228, 238, 240, 243, 246, 250, 254, 265, 274, 276, 281, 285, 287, 291, 296, 299, 300, 302, 305, 307, 309, 310, 312, 314	GetHeight	250, 254
GetDmxIn	128, 144	GetHeightMax	254
GetDmxInChannel	128, 144	GetImageCount	228
GetDoubleInwards	287, 300	GetInnerGlow	258
GetDoubleOutwards	287, 300	GetIntensity	250
GetDrawMode	287	GetInvert	287, 291, 296, 300
GetDrawShape	246	GetLayerBlind	325
GetDuration	234	GetLayerCount	325
GetEffectMode	265, 285	GetLayerDegreeMapRotation	325
GetExplosionMode	246	GetLayerFilter	325
GetExplosionShape	246	GetLayerFrameCount	325
GetExplosionSize	246	GetLayerFrameld	325
GetFactor	265	GetLayerFrameSteps	325
GetFade	234, 240, 264, 287, 292, 296, 297, 299, 300, 302, 308, 310, 314, 316	GetLayerLink	325
GetFadeColor	334, 339	GetLayerMapModeMirror	325
GetFadeOut	232, 246	GetLayerMapModeTile	325
GetFadeTime	334, 339	GetLayerMapPixel	325
GetFadeType	334, 339	GetLayerMapTileEffectPixel	325
GetFadeValue	334, 339	GetLayerMapTileEffectVector	325
GetFillMode	234	GetLayerMapVector	325
GetFillTime	234	GetLayerMixMode	325
GetFilter	223, 325	GetLayerOpacity	325
		GetLayerPixelTileOffset	325
		GetLayerSolo	325
		GetLayerSubMaster	325
		GetLayerVectorMapRotation	325
		GetLayerVectorTileOffset	325
		GetLength	243, 265, 274, 291, 312
		GetLink	128, 144
		GetLoop	281

- GetMapModeMirror 128, 144
GetMapModeTile 128, 144
GetMapPixel 111, 128, 144
GetMapTileEffectPixel 115
GetMapTileEffectVector 115
GetMapVector 111, 128, 144
GetMasterFader 334, 339
GetMatrixHeight 128, 144
GetMatrixWidth 128, 144
GetMaxBand 297
GetMaxSize 258
GetMidInControl 128, 144
GetMidInControlValue 128, 144
GetMidInNote 128, 144
GetMidInNoteValue 128, 144
GetMinBand 297
GetMirror 234, 287, 291, 296, 300
GetMixColor 234
GetMixMode 117, 128, 144
GetMode 250, 254, 258, 264, 276
GetModeFrequency 254
GetModeHeight 254
GetModePitch 254
GetModeWidth 254
GetMono 299, 302
GetMonochrome 287, 300
GetNoteValue 123, 128, 144, 147
GetObjects 232
GetOffTime 264
GetOnTime 264
GetOpacity 128, 144
GetOuterGlow 258
GetPause 325
GetPeak 254, 285
GetPitch 234, 243, 254
GetPitchMax 254
GetPixel 128, 144
GetPixelArea 97, 128, 144
GetPixelBorder 270, 292, 316
GetPixelCenterX 265
GetPixelCenterY 265
GetPixelImageHeight 228, 281
GetPixelImagePositionX 228, 281
GetPixelImagePositionY 228, 281
GetPixelImageWidth 228, 281
GetPixelLength 285
GetPixelMaxRandValue 240
GetPixelMinRandValue 240
GetPixelOffset 240
GetPixelPitch 292, 316
GetPixelStepWidth 240
GetPixelTextHeight 276
GetPixelTextOffsetX 276
GetPixelTextOffsetY 276
GetPixelTextPosX 276
GetPixelTextPosY 276
GetPixelTextWidth 276
GetPixelWidth 270, 292, 312, 314, 316
GetPixelXOffset 240
GetPoints 232
GetPushHF 287
GetRandom 240
GetReverseSentence 276
GetReverseWords 276
GetRgbToRgbw 228, 281
GetRocketCount 246
GetRotation 228, 254, 265, 274, 276, 281
GetScale 258
GetScriptEngineVersion 128, 144
GetSens 291, 292, 296, 297, 316
GetShape 232, 234, 243, 254, 258, 270, 274, 292, 302, 316
GetShapeCount 246
GetShapeSize 246
GetSharpness 258
GetShiftInwards 305, 307
GetShiftOutwards 305, 307
GetSize 232
GetSmooth 276
GetSoftwareVersion 128, 144
GetSoundLevel 119, 128, 144, 147
GetSparkleColor 246
GetSparkleColorCount 246
GetSparkleColorMode 246
GetSpeed 188, 224
GetSpeedMaster 325
GetStep 243, 310
GetStorageFilter 334
GetStorageFullState 334, 339
GetStoragePause 334, 339
GetStoragePlace 334, 339
GetStorageSpeedMaster 334, 339
GetStorageSubMaster 334, 339
GetStretch 228, 281
GetSubMaster 128, 144, 325
GetText 254, 276

GetTextColor 276
 GetTile 228, 281
 GetTileEffect 115
 GetTime 128, 144
 GetTimeCode 128, 144
 GetTimeSlot 305, 307
 GetTonality 123, 128, 144, 147
 GetToneScale 123, 128, 144, 147
 GetUseBass 312, 314
 GetUseLevel 291, 292, 296
 GetUserName 128, 144
 GetUserProfileDirectory 128, 144
 GetVectorBorder 270, 292, 316
 GetVectorCenterX 265
 GetVectorCenterY 265
 GetVectorImageHeight 228, 281
 GetVectorImagePositionX 228, 281
 GetVectorImagePositionY 228, 281
 GetVectorImageWidth 228, 281
 GetVectorLength 285
 GetVectorMapRotation 128, 144
 GetVectorMaxRandValue 240
 GetVectorMinRandValue 240
 GetVectorOffset 240
 GetVectorPitch 292, 316
 GetVectorPixel 128, 144
 GetVectorStepWidth 240
 GetVectorTextHeight 276
 GetVectorTextOffsetX 276
 GetVectorTextOffsetY 276
 GetVectorTextPosX 276
 GetVectorTextPosY 276
 GetVectorTextWidth 276
 GetVectorWidth 270, 292, 312, 314, 316
 GetVectorXOffset 240
 GetVideoEndTime 281
 GetVideoLength 281
 GetVideoLoaded 281
 GetVideoPlaySpeed 281
 GetVideoRunning 281
 GetVideoStartTime 281
 GetVideoTime 281
 GetViewLog 287, 300, 302
 GetWidth 234, 243, 254, 274
 GetWidthMax 254
 GetXOffset 310
 global variable 47, 67
 GOLD 155

Graphical user interface 190
 GRAY 92, 155
 GREEN 92, 155
 Greyscale 128, 144
 GUI elements 190

- H -

H 170
 High speed in scripts 209
 HSLIDEFADE 334
 HWIPEFADE 334
 HXSLIDEFADE 334
 HXWIPEFADE 334
 hypot 128, 144, 147
 Hz 164, 170

- I -

Identifier 41
 if 68, 128
 ImportPatch 334
 ImportStorage 334
 ImportStoragePlace 334
 index operator 53
 information about a script 82
 InitEffect 42, 185, 220, 322, 331
 inoage 355
 int 50, 52
 int FILTER_EDGES 93
 int FILTER_EDGES_POPUP 93
 int FILTER_EMBOSS 93
 int FILTER_EMBOSS_POPUP 93
 int FILTER_GREEN_BLUE_WHITE 93
 int FILTER_INVERT_H_MATRIX 93
 int FILTER_INVERT_HV_MATRIX 93
 int FILTER_INVERT_V_MATRIX 93
 interval 123
 InvertColor 128, 144
 InvertColorPositions 85, 238, 250, 263, 287, 300, 305
 InvertColors 85, 238, 250, 263, 287, 300, 305
 InvertMatrix 128, 144
 isalnum 56, 128, 144
 isalpha 56, 128, 144
 IsDmxInEnabled 128, 144
 IsFrameFadeEnabled 128, 144

IsInterval 128, 144, 147
 IsLayerFrameFadeEnabled 325
 IsLayerMapped 325
 IsLayerMapRotation 325
 IsMapped 111, 128, 144
 IsMapRotation 128, 144
 IsMidInEnabled 128, 144
 IsNote 128, 144, 147
 isnum 56, 128, 144
 IsTonality 128, 144, 147

- K -

keywords 128

- L -

length 53
 LIGHT_GRAY 155
 In 128, 144, 147
 Loading 34
 local variable 47, 67
 log10 128, 144, 147
 logical operation 62
 loop 74

- M -

M2L 119, 123, 147
 M2L Color Fade 308
 M2L Color Rings 309
 M2L Color Scroll 310
 M2L Interval Drops 312
 M2L Interval Tubes 314
 M2L Single Tone Shapes 316
 Macro 185, 220, 322, 331
 Macro extension 220
 Macros for Effects 31, 220
 Macros For Effects: Specific Functions 223
 MADRIX_GREEN 92, 155
 MAGENTA 155
 Main Output Macro 31, 331, 334
 Main Output: Specific Functions 334
 Map 148
 Map Dialog 148
 MAP_ANIM_OFF 325
 MAP_ANIM_ON 325

MAP_MIRROR_H 155
 MAP_MIRROR_HV 155
 MAP_MIRROR_NONE 155
 MAP_MIRROR_V 155
 MAP_TILE_MIRROR_H 155
 MAP_TILE_MIRROR_HV 155
 MAP_TILE_MIRROR_V 155
 MAP_TILE_NONE 155
 MAP_TILE_REPEAT 155
 MapEffect 116
 MapEffectPixel 111, 128, 144
 MapEffectVector 111, 128, 144
 MapLayerEffectPixel 325
 MapLayerEffectVector 325
 MapLayerTileEffectPixel 325
 MapLayerTileEffectVector 325
 MapTileEffectPixel 115
 MapTileEffect 116
 MapTileEffectPixel 128, 144
 MapTileEffectVector 115, 128, 144
 MAROON 92, 155
 MAS Script Effect 31, 185
 MAS Script Effect: Specific Functions 188
 Math functions 147
 math operation 62
 MatrixSizeChanged 42, 185, 220, 322, 331
 max 128, 144, 147
 MAX_FREQUENCY_VOLUME 155
 Maximal render frequency 216
 Memory management of fields 53
 min 128, 144, 147
 MirrorMode 115
 MixMode 117
 MIXMODE_AND 155
 MIXMODE_COLORBURN 155
 MIXMODE_COLORDODGE 155
 MIXMODE_DARKEN 155
 MIXMODE_DIFFERENCE 155
 MIXMODE_EXCLUSION 155
 MIXMODE_HARDLIGHT 155
 MIXMODE_HARDMIX 155
 MIXMODE_LIGHTEN 155
 MIXMODE_LINEARBURN 155
 MIXMODE_LINEARDODGE 155
 MIXMODE_LINEARLIGHT 155
 MIXMODE_MASK 155
 MIXMODE_MULTIPLY 155
 MIXMODE_NAND 155

MIXMODE_NOR 155
 MIXMODE_NORMAL 155
 MIXMODE_OR 155
 MIXMODE_OVERLAY 155
 MIXMODE_PINLIGHT 155
 MIXMODE_SCREEN 155
 MIXMODE_SOFTLIGHT 155
 MIXMODE_VIVIDLIGHT 155
 MIXMODE_XOR 155
 MixModes 117
 MODE_BAR 287
 MODE_BLURRY 258
 MODE_CHAR 254, 276
 MODE_CIRCLE 258
 MODE_CLEAR 258
 MODE_COLLAPSE 234
 MODE_COSINE 254
 MODE_CUBIC 254, 258
 MODE_DIAMOND 258
 MODE_DROPS 234
 MODE_EXPLOSIONS 246
 MODE_FIRE 250
 MODE_FIREWORKS 246
 MODE_FLAMES 250
 MODE_FLAT 234
 MODE_LINEAR 254, 258
 MODE_MEDIUM 258
 MODE_NONE 254
 MODE_PULSE 264
 MODE_QUADRATIC 254, 258
 MODE_RADIAL 287
 MODE_RADIAL_DOT 287
 MODE_RADIAL_LINE 287
 MODE_RADIAL_OUTLINE 287
 MODE_RANDOM 234, 254, 258
 MODE_RECTANGLE 258
 MODE_SENTENCE 254, 276
 MODE_SINE 254
 MODE_SLIGHTLY_BLURRY 258
 MODE_SLIGHTLY_CLEAR 258
 MODE_SNAKE 234
 MODE_SQRT 254, 258
 MODE_SQUARE 254
 MODE_STROBO 264
 MODE_TETRIS 234
 MODE_TRIANGLE 254
 MODE_UNIFORM 254, 258
 MODE_VERY_BLURRY 258

MODE_VERY_CLEAR 258
 MODE_WORD 254, 276
 multidimensional field 53
 Music 2 Light 119, 123, 147

- N -

name 41, 82
 NAVY 92, 155
 No FX 93, 223
 Non-primitive data types 50
 NOPAUSE 325, 334
 note 123
 note table 170
 notes 170

- O -

OLIVE 92, 155
 operator 62
 ORANGE 155

- P -

Parameter 42
 PAUSE 325, 334
 persistent 47, 128
 PI 155
 PINK 155
 pixel 83
 PixelFloodFill 128, 144
 PostRenderEffect 42, 220, 322, 331
 pow 128, 144, 147
 PreRenderEffect 42, 185, 220, 322, 331
 primitive data types 50
 PURPLE 92, 155

- R -

rad2deg 128, 144, 147
 random 128, 144, 147
 ReadAsync 128, 144
 RED 92, 155
 relative coordinates 83
 RemoveColor 85, 232, 234, 238, 240, 243, 246,
 250, 254, 258, 263, 270, 274, 287, 292, 300, 305, 316
 RemoveSparkleColor 246

Render frequency 209
 RenderEffect 42, 185
 replace 56, 128, 144
 return 42, 128
 Return value 42
 rfindstring 56, 128, 144
 ROTATION_CCW 265
 ROTATION_CW 265
 ROTATION_TEXT_180 334
 ROTATION_TEXT_270 334
 ROTATION_TEXT_90 334
 ROTATION_TEXT_NONE 334
 round 128, 144, 147

- S -

S2L 119, 147, 155
 S2L EQ Drops 291
 S2L EQ Shapes 292
 S2L EQ Tubes 296
 S2L Equalizer 287
 S2L Frequency Flash 297
 S2L Level Color 299
 S2L Level Meter 300
 S2L Level Ring 302
 S2L Waveform 305
 S2L Wavegraph 307
 save data 47
 Save macro 220
 save script 185
 Saving 34
 scale 123
 SCE Bitmap 228
 SCE Bounce 232
 SCE Color 226
 SCE Color Change 234
 SCE Color Fill 234
 SCE Color Ramp 238
 SCE Color Scroll 240
 SCE Drops 243
 SCE Explosions 246
 SCE Fire 250
 SCE Graph 254
 SCE Metaballs 258
 SCE Plasma 263
 SCE Pulse / Stroboscope 264
 SCE Radial 265
 SCE Shapes 270
 SCE Starfield 274
 SCE Ticker 276
 SCE Video 281
 SCE Wave 285
 Script 185
 Script Editor 34
 Script Editor Window 34
 script effect 185
 Script Examples 171
 script information 82
 Script variables 47
 SeedRandom 258
 SeekVideo 281
 Service variables 47
 SetAmplify 287, 291, 292, 296, 300, 302
 SetAmplitude 265
 SetAngle 238
 SetAnimationSpeed 228
 SetAudioFader 334, 339
 SetAutostart 281
 SetBandCount 287, 291, 296, 300
 SetBandMode 287
 SetBlackout 334
 SetBlur 246
 SetBorder 258
 SetBpm 188, 223, 224, 226, 228, 232, 234, 240, 243, 246, 250, 254, 258, 263, 265, 274, 276, 281, 285, 287, 291, 296, 299, 300, 302, 309, 310, 312, 314
 SetCenter 234
 SetChangeTime 234
 SetChannelView 305, 307
 SetCircle 234
 SetCollision 232
 SetColor 85, 226, 232, 234, 238, 240, 243, 246, 250, 254, 258, 263, 264, 265, 270, 274, 285, 287, 291, 292, 296, 297, 299, 300, 302, 305, 307, 308, 309, 310, 312, 314, 316
 SetColorFade 85, 238, 250, 263, 287, 300, 305
 SetColorMix 258
 SetColorMixLink 258
 SetColorMode 243, 246, 250, 254, 258, 270, 274, 292, 316
 SetColorPosition 85, 238, 250, 263, 287, 300, 305
 SetContinuous 254, 276
 SetCount 243, 258, 265, 270, 274
 SetCrossed 238, 240, 287, 300, 310
 SetCurrentImage 228
 SetCurve 265
 SetDegreeMapRotation 128, 144

SetDepth 274
SetDescription 325
SetDirection 228, 238, 240, 243, 246, 250, 254, 265, 274, 276, 281, 285, 287, 291, 296, 299, 300, 302, 305, 307, 309, 310, 312, 314
SetDoubleInwards 287, 300
SetDoubleOutwards 287, 300
SetDrawMode 287
SetDrawShape 246
SetDuration 234
SetEffectMode 265, 285
SetExplosionMode 246
SetExplosionShape 246
SetExplosionSize 246
SetFactor 265
SetFade 234, 240, 264, 287, 292, 296, 297, 299, 300, 302, 308, 310, 314, 316
SetFadeColor 334, 339
SetFadeOut 232, 246
SetFadeTime 334, 339
SetFadeType 334, 339
SetFadeValue 334, 339
SetFillMode 234
SetFillTime 234
SetFilter 223, 325
SetFilterColor 228, 281, 334
SetFixedFrameRate 188, 216
SetFlameSize 250
SetFontFaceName 254, 276
SetFontHeight 254, 276
SetFontItalic 254, 276
SetFontStrikeOut 254, 276
SetFontUnderline 254, 276
SetFontWeight 254, 276
SetFontWidth 254, 276
SetFrameCount 188, 213
SetFrameId 128, 144, 188, 213, 220
SetFreeze 334, 339
SetFrequency 254, 264
SetFrequencyMax 254
SetGraphMode 254
SetGravity 246
SetGrey 228, 281
SetHeight 250, 254
SetHeightMax 254
SetInnerGlow 258
SetIntensity 250
SetInvalid 128, 144
SetInvert 287, 291, 296, 300
SetLayerBlind 325
SetLayerDegreeMapRotation 325
SetLayerFilter 325
SetLayerFrameId 325
SetLayerLink 325
SetLayerMapModeMirror 325
SetLayerMapModeTile 325
SetLayerMixMode 325
SetLayerOpacity 325
SetLayerPixelTileOffset 325
SetLayerSolo 325
SetLayerSubMaster 325
SetLayerVectorMapRotation 325
SetLayerVectorTileOffset 325
SetLength 243, 265, 274, 291, 312
SetLink 128, 144
SetLoop 281
SetMapMode 116
SetMapModeMirror 115, 128, 144
SetMapModeTile 115, 128, 144
SetMasterFader 334, 339
SetMatrix 97
SetMaxBand 297
SetMaxSize 258
SetMinBand 297
SetMirror 234, 287, 291, 296, 300
SetMirrorMode 115
SetMixColor 234
SetMixMode 117, 128, 144
SetMode 250, 254, 258, 264, 276
SetModeFrequency 254
SetModeHeight 254
SetModePitch 254
SetModeWidth 254
SetMono 299, 302
SetMonochrome 287, 300
SetObjects 232
SetOffTime 264
SetOnTime 264
SetOpacity 128, 144
SetOuterGlow 258
SetPause 325
SetPeak 254, 285
SetPitch 234, 243, 254
SetPitchMax 254
SetPixel 107, 128, 144
SetPixelBorder 270, 292, 316

- SetPixelCenter 265
- SetPixelGreyscale 107, 128, 144
- SetPixelImagePosition 228, 281
- SetPixelLength 285
- SetPixelOffset 240
- SetPixelPitch 292, 316
- SetPixelRandValues 240
- SetPixelStepWidth 240
- SetPixelTextOffset 276
- SetPixelTransposeEntry 128, 144
- SetPixelWidth 270, 292, 312, 314, 316
- SetPixelXOffset 240
- SetPoints 232
- SetPushHF 287
- SetRandom 240
- SetReadAsyncInterval 128, 144
- SetReverseSentence 276
- SetReverseWords 276
- SetRgbToRgba 228, 281
- SetRocketCount 246
- SetRotation 228, 254, 265, 274, 276, 281
- SetScale 258
- SetSens 291, 292, 296, 297, 316
- SetShape 232, 234, 243, 254, 258, 270, 274, 292, 302, 316
- SetShapeCount 246
- SetShapeSize 246
- SetSharpness 258
- SetShiftInwards 305, 307
- SetShiftOutwards 305, 307
- SetSize 232
- SetSmooth 276
- SetSparkleColor 246
- SetSparkleColorMode 246
- SetSpeed 188, 224
- SetSpeedMaster 325
- SetStep 243, 310
- SetStorageFilter 334
- SetStoragePause 334, 339
- SetStoragePlace 334, 339
- SetStorageSpeedMaster 334, 339
- SetStorageSubMaster 334, 339
- SetStretch 228, 281
- SetSubMaster 128, 144, 325
- SetText 254, 276
- SetTextColor 276
- SetTile 228, 281
- SetTileMode 115
- SetTimeSlot 305, 307
- SetUniformDistances 85, 238, 250, 263, 287, 300, 305
- SetUseBass 312, 314
- SetUseFloatFrames 188, 209
- SetUseLevel 291, 292, 296
- SetValid 128, 144
- SetVectorBorder 270, 292, 316
- SetVectorCenter 265
- SetVectorImagePosition 228, 281
- SetVectorLength 285
- SetVectorMapRotation 128, 144
- SetVectorOffset 240
- SetVectorPitch 292, 316
- SetVectorPixel 128, 144
- SetVectorRandValues 240
- SetVectorStepWidth 240
- SetVectorTextOffset 276
- SetVectorWidth 270, 292, 312, 314, 316
- SetVectorXOffset 240
- SetVideoAspectRatio 281
- SetVideoEndTime 281
- SetVideoPlaySpeed 281
- SetVideoStartTime 281
- SetVideoTime 281
- SetViewLog 287, 300, 302
- SetWidth 234, 243, 254, 274
- SetWidthMax 254
- SetXOffset 310
- SHIFT_C_IN_OUT 155
- SHIFT_C_OUT_IN 155
- SHIFT_DL 155
- SHIFT_DOWN 155
- SHIFT_DR 155
- SHIFT_H_IN_OUT 155
- SHIFT_H_OUT_IN 155
- SHIFT_LEFT 155
- SHIFT_RIGHT 155
- SHIFT_UL 155
- SHIFT_UP 155
- SHIFT_UR 155
- SHIFT_V_IN_OUT 155
- SHIFT_V_OUT_IN 155
- ShiftMatrix 96
- ShiftPixelMatrix 96, 128, 144
- ShiftVectorMatrix 96, 128, 144
- SILVER 92, 155
- sin 128, 144, 147

sinDeg 128, 144, 147
 sinH 128, 144, 147
 sinHDeg 128, 144, 147
 SKY 155
 Slider 190
 Sound 2 Light 119, 147, 155
 Sound Data 155
 SOUND_DATA_LEFT 119, 155
 SOUND_DATA_LEFT[] 155
 SOUND_DATA_RIGHT 119, 155
 SOUND_DATA_RIGHT[] 155
 Spectrum 164
 speed 224
 speedmaster 209
 sqrt 128, 144, 147
 startswith 56, 128, 144
 StartVideo 281
 StartVideoBackward 281
 statement 67
 StopVideo 281
 Storage Place Macro 31, 322, 325
 STORAGE_A 334
 STORAGE_B 334
 Store macro 220
 store script 185
 strcmp 56, 128, 144
 String 50, 52, 62, 150
 string operations 56
 strip 56, 128, 144
 structure 52, 162
 structures 50, 162
 substring 56, 128, 144
 switch 56, 71, 128
 Syntax Highlighting 39

- T -

tan 128, 144, 147
 tanDeg 128, 144, 147
 tanH 128, 144, 147
 tanHDeg 128, 144, 147
 TEAL 92, 155
 Text field 190
 TileEffect 115, 116
 TileMode 115
 time 50, 52, 162
 tokenize 56, 128, 144
 tolower 56, 128, 144

tonality 123
 tone scale 123
 toupper 56, 128, 144
 TRACE 128, 144
 TRUE 50, 128
 trunc 128, 144, 147
 TURQUOISE 155

- U -

UNFREEZE 334

- V -

Variable 41
 Variable declaration 47
 Variables 46, 47
 vector 83
 VectorFloodFill 128, 144
 version 82
 void Filter(int filter) 93
 VSLIDEFADE 334
 VWIPEFADE 334
 VXSLIDEFADE 334
 VXWIPEFADE 334

- W -

while 74, 128
 WHITE 92, 155
 WHITE_ALPHA 155
 WHITE_SPACES 56, 155
 WHITEFADE 334
 WITH_AUTOFAD 334
 WITHOUT_AUTOFAD 334
 WriteText 128, 144

- Y -

YELLOW 92, 155