MADRIX 5 Script Help and Manual

[Software User Guide]

MADRIX Version: 5.3b

MADRIX Script Version: 3.12

Date: July 2020

Table Of Contents

MADRIX LIGHTING CONTROL

//PART A

What Is New

1 What Is New

Overview

The current MADRIX Script version is 3.12.

There are a lot of new features within MADRIX and therefore also in MADRIX Script.

News For Script Engine Version 3.12 (MADRIX 5.3)

Added Support For:

- **BlackTrax**

New General Functions:

- **FilterColor**
- **CaptureScreen**
- **DrawPixelBitmap**
- **DrawVectorBitmap**
- **DrawPixelBitmap3D**
- **DrawVectorBitmap3D**

New Functions For the Global Macro:

- **ImportFixtureGroupController**

News For Script Engine Version 3.10 (MADRIX 5.1)

New Effects:

- **SCE Credits**
- **SCE Split Shapes**
- **SCE Water**
- **TRI Flash**
- **TRI Fluid**
- **TRI Split Shapes**
- **TRI Water**

New Functions For SCE Capture:

- **SetAlphaBlending**
- **GetAlphaBlending**

New Functions For SCE Gradient And SCE Wave / Radial:

- **SetDisplacement**
- **GetDisplacement**
- **SetDisplacementSpeed**
- **GetDisplacementSpeed**
- **SetDisplacementDistribution**
- **GetDisplacementDistribution**

Changed Functions For SCE Wave / Radial

- **SetPeak**
- **GetPeak**

New And Updated Functions For MADRIX Effects That Use The Shape Table:

- **ShapeTableSetShapeMinSize**
- **ShapeTableGetShapeMinSize**

- **ShapeTableAddShape**

New And Updated Functions for MADRIX Effects That Use The String Table:

- **StringTableGetPagesCount**
- **StringTableAddPage**
- **StringTableRemovePage**
- **StringTableMovePageToNext**
- **StringTableMovePageToPrev**
- **StringTableInvertPages**
- **StringTableSwapPages**

- **StringTableGetCount**
- **StringTableMoveStringUp**
- **StringTableMoveStringDown**
- **StringTableInvert**
- **StringTableSwapStrings**
- **StringTableGetString**
- **StringTableSetString**
- **StringTableAddString**
- **StringTableRemoveString**

New Functions For SCE Drops, S2L Drops, M2L Drops, And TRI Drops:

- **SetPeak** (now uses float instead of int)
- **GetPeak** (now uses float instead of int)

New Constants For Effects That Use Rendering Mode Extended:

- **SHAPE_TYPE_PENTAGON_OUTLINED**
- **SHAPE_TYPE_PENTAGON_OUTLINED_IMPLODE**
- **SHAPE_TYPE_PENTAGON_OUTLINED_EXPLODE**
- **SHAPE_TYPE_PENTAGON_OUTLINED_PULSE**
- **SHAPE_TYPE_PENTAGON_FILLED**
- **SHAPE_TYPE_PENTAGON_FILLED_IMPLODE**
- **SHAPE_TYPE_PENTAGON_FILLED_EXPLODE**
- **SHAPE_TYPE_PENTAGON_FILLED_PULSE**
- **SHAPE_TYPE_HEXAGON_OUTLINED**
- **SHAPE_TYPE_HEXAGON_OUTLINED_IMPLODE**
- **SHAPE_TYPE_HEXAGON_OUTLINED_EXPLODE**
- **SHAPE_TYPE_HEXAGON_OUTLINED_PULSE**
- **SHAPE_TYPE_HEXAGON_FILLED**
- **SHAPE_TYPE_HEXAGON_FILLED_IMPLODE**
- **SHAPE_TYPE_HEXAGON_FILLED_EXPLODE**
- **SHAPE_TYPE_HEXAGON_FILLED_PULSE**

News For Script Engine Version 3.08 (MADRIX 5.0e)

Updated Functions And Constants For The Global Macro:

- **CueSetTimeCode**
- **CueSetDuration**
- **LEFT, RIGHT, and MAIN** (streamlined from STORAGE_LEFT and STORAGE_RIGHT)

News For Script Engine Version 3.07 (MADRIX 5.0c)

New Effects:

- **SCE Rotating Shapes**
- **TRI Shapes**

New Functions And Constants For MADRIX Effects That Use The Shape Table:

- **ShapeTableGetShapeCount**
- **ShapeTableMoveShapeUp**
- **ShapeTableMoveShapeDown**
- **ShapeTableSwapShapes**
- **ShapeTableAddShape**
- **ShapeTableRemoveShape**
- **ShapeTableSetShapeType**
- **ShapeTableGetShapeType**
- **ShapeTableSetShapeAlignment**
- **ShapeTableGetShapeAlignment**
- **ShapeTableSetShapeOrigin**
- **ShapeTableGetShapeOrigin**
- **ShapeTableSetShapeInnerGlow**

- ShapeTableGetShapeInnerGlow
- ShapeTableSetShapeBorder
- ShapeTableGetShapeBorder
- ShapeTableSetShapeOuterGlow
- ShapeTableGetShapeOuterGlow
- ShapeTableSetMode
- ShapeTableGetMode
- SHAPETABLE_MODE_LOOP
- SHAPETABLE_MODE_SHUFFLE

New Functions For SCE Drops, S2L Drops, And M2L Drops:

- Shape Table
- Shape Rotation
- SetLengthMin
- GetLengthMin
- SetPixelLengthMin
- GetPixelLengthMin
- SetLengthMax
- GetLengthMax
- SetPixelLengthMax
- GetPixelLengthMax
- SetLengthDistribution
- GetLengthDistribution
- SeedRandomLength
- SetRotationOffset
- GetRotationOffset

- **SetDisplacement**
- **GetDisplacement**
- **SetDisplacementSpeed**
- **GetDisplacementSpeed**
- **SetDisplacementDistribution**
- **GetDisplacementDistribution**

New Constants For MADRIX Effects That Use Shapes And For **RenderShape**:

- **ORIGIN_GEOMETRIC_CENTER**

News For Script Engine Version 3.04 (MADRIX 5.0)

- MADRIX 5.0 includes a large number of (technical) improvements. That is why parts of Script Engine Version 3.04 were updated as well.
- Previous macros and scripts written for MADRIX 3.X should work for MADRIX 5.X without any changes in most cases.

New Effects:

- TRI Color Change
- TRI Drops
- TRI Explosions
- TRI Tubes

New Filters (FX):

- FILTER_BRIGHTNESS_GRAPH_XYZ
- FILTER_BRIGHTNESS_GRAPH_XZY
- FILTER_BRIGHTNESS_GRAPH_YXZ
- FILTER_RGB_TO_BGR
- FILTER_RGB_TO_BRG
- FILTER_RGB_TO_GRB
- FILTER_RGB_TO_GBR
- FILTER_RGB_TO_RBG
- FILTER_RGBW_TO_WBGR
- FILTER_RGBW_TO_WBRG
- FILTER_RGBW_TO_WGRB
- FILTER_RGBW_TO_WGBR
- FILTER_RGBW_TO_WRBG
- FILTER_RGBW_TO_WRGB
- FILTER_RGBW_TO_BWGR
- FILTER_RGBW_TO_BWRG
- FILTER_RGBW_TO_GWRB
- FILTER_RGBW_TO_GWBR

- FILTER_RGBW_TO_RWBG
- FILTER_RGBW_TO_RWGB
- FILTER_RGBW_TO_BGWR
- FILTER_RGBW_TO_BRWG
- FILTER_RGBW_TO_GRWB
- FILTER_RGBW_TO_GBWR
- FILTER_RGBW_TO_RBWG
- FILTER_RGBW_TO_RGWB
- FILTER_SWAP_H_1X
- FILTER_SWAP_H_2X
- FILTER_SWAP_H_3X
- FILTER_SWAP_H_4X
- FILTER_SWAP_H_5X
- FILTER_SWAP_V_1X
- FILTER_SWAP_V_2X
- FILTER_SWAP_V_3X
- FILTER_SWAP_V_4X
- FILTER_SWAP_V_5X
- FILTER_SWAP_HV_1X
- FILTER_SWAP_HV_2X
- FILTER_SWAP_HV_3X
- FILTER_SWAP_HV_4X
- FILTER_SWAP_HV_5X
- FILTER_SWAP_D_1X
- FILTER_SWAP_D_2X
- FILTER_SWAP_D_3X

- FILTER_SWAP_D_4X
- FILTER_SWAP_D_5X
- FILTER_SWAP_HD_1X
- FILTER_SWAP_HD_2X
- FILTER_SWAP_HD_3X
- FILTER_SWAP_HD_4X
- FILTER_SWAP_HD_5X
- FILTER_SWAP_VD_1X
- FILTER_SWAP_VD_2X
- FILTER_SWAP_VD_3X
- FILTER_SWAP_VD_4X
- FILTER_SWAP_VD_5X
- FILTER_SWAP_HVD_1X
- FILTER_SWAP_HVD_2X
- FILTER_SWAP_HVD_3X
- FILTER_SWAP_HVD_4X
- FILTER_SWAP_HVD_5X

New Functions For MADRIX Effects That Use The Center Control:

- SetCenterX
- GetCenterX
- SetCenterY
- GetCenterY
- SetCenterZ
- GetCenterZ
- SetCenter

- **SetPixelCenterX**
- **GetPixelCenterX**
- **SetPixelCenterY**
- **GetPixelCenterY**
- **SetPixelCenterZ**
- **GetPixelCenterZ**
- **SetPixelCenter**

New Functions For SCE Wave / Radial:

- **Center Control** (replaces the **Position Control**)

New Functions For SCE Graph:

- **SetShapeRotation**
- **GetShapeRotation**
- **SetBorder**
- **GetBorder**
- **SetPixelBorder**
- **GetPixelBorder**
- **SetOuterGlow**
- **GetOuterGlow**
- **SetPixelOuterGlow**
- **GetPixelOuterGlow**
- **SetInnerGlow**
- **GetInnerGlow**
- **SetPixelInnerGlow**

- GetPixelInnerGlow

New Functions For SCE Starfield:

- SetShapeRotation
- GetShapeRotation
- SetRenderingMode
- GetRenderingMode
- SetPathAligned
- GetPathAligned
- SetBorder
- GetBorder
- SetPixelBorder
- GetPixelBorder
- SetOuterGlow
- GetOuterGlow
- SetPixelOuterGlow
- GetPixelOuterGlow
- SetInnerGlow
- GetInnerGlow
- SetPixelInnerGlow
- GetPixelInnerGlow

New Functions For The Effect Parameter Chaser:

- ChaserSetAutostart
- ChaserGetAutostart

- **ChaserToggleAutostart**

New Functions And Constants For The Cue List:

- **CuelistSetTimecodeSource**
- **CuelistGetTimecodeSource**
- **CuelistSetTimecodeFormat**
- **CuelistGetTimecodeFormat**
- **TIMECODE_SOURCE_NONE**
- **TIMECODE_SOURCE_ARTNET**
- **TIMECODE_SOURCE_MIDI**
- **TIMECODE_SOURCE_SMPTE**
- **TIMECODE_SOURCE_SYSTEM_TIME**
- **TIMECODE_FORMAT_24_FPS**
- **TIMECODE_FORMAT_25_FPS**
- **TIMECODE_FORMAT_30_DROP**
- **TIMECODE_FORMAT_30_FPS**

New Functions For The Global Macro:

- **GetAudioInputFader** (replaces **GetAudioFader**)
- **SetAudioInputFader** (replaces **SetAudioFader**)
- **GetAudioInputMute**
- **SetAudioInputMute**
- **GetAudioOutputFader**
- **SetAudioOutputFader**
- **GetAudioOutputMute**

- **SetAudioOutputMute**

New Functions:

- **Numbers** starting with *0x* are interpreted as hexadecimal values
- **hex**
- **Hex**

New Graphical User Interface Elements:

- **ctrlcheckbutton**
- **ctrlcheckbutton2**
- **ctrlcheckbutton3**
- **ctrlcheckbutton4**
- **ctrlcheckbutton5**
- **ctrlcheckbutton6**

Changed Functions:

- **PixelFloodFill** now has an optional parameter to specify the z-plane
- **VectorFloodFill** now has an optional parameter to specify the z-plane
- **ReadAsync** now has an optional parameter to specify the file encoding

New Operators For **Expressions**:

- **++** (as prefix)
- **--** (as prefix)
- **~**

- ^ and ^=
- | and |=
- & and &=
- << and <<=
- >> and >>=
- >>> and >>>=

MADRIX LIGHTING CONTROL

//PART B

MADRIX Script (Introduction)

2 MADRIX Script (Introduction)

2.1 Basics

2.1.1 Getting Started

Introduction

- MADRIX Script is the scripting language of MADRIX. It is built into MADRIX.
- This document is for all those who want to develop and modify light effects with the help of MADRIX Script. This does not require any programming knowledge although such knowledge can be helpful.

Examples

With MADRIX Script you could do the following, for example:

- Displaying the current time using SCE Ticker / Scrolling Text.
- Increasing and decreasing the size of shapes according to the audio input level.
- Automatically activating or deactivating a Blackout at certain times.
- Setting different Layer filters for different Layers according to automatic parameters.

Overview

4 Locations

There are 4 possibilities to use MADRIX Script.

- ***MAS Script Effect***
  The first option is to create a new effect from scratch.
  Learn more »**MAS Script Effect**
- ***Macros for Effects***
  The second option involves modifying the settings of a MADRIX Effect. This includes all SCE, S2L, M2L, TRI, and MAS effects.
  Learn more »**Macros for Effects**
- ***Storage Place Macro***
  Third, you can use Storage Place macros to influence every single Storage Place individually.
  Learn more »**Storage Place Macro**
- ***Global Macro***
  The fourth possibility controls global features and mainly the Main Output directly.
  Learn more »**Global Macro**

MAS Script Effect — Create Your Own Effects

MADRIX offers endless possibilities to create a light show. However, there are a lot more things that you are maybe not able to do with the current stock effects. The script effect, called ***MAS Script Effect***, provides the possibility to program your own, original effects.

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 outcome. For example, render parts of an effect transparent or change the color with a gray filter.

Storage Place Macro — Control Individual Storage Places Including All Layers

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

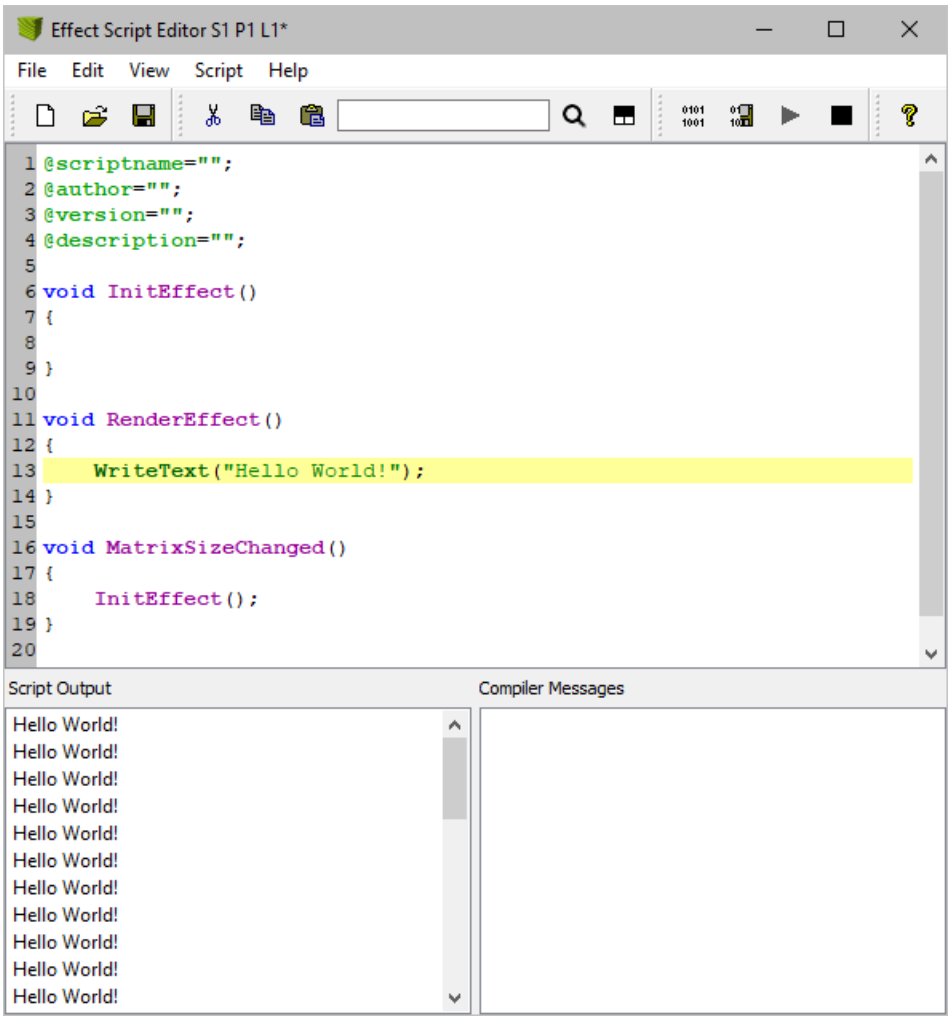
Global Macro — Control Your Final Output

Whereas the MAS Script Effect is an individual effect in itself and while macros can be used to manipulate single effects or Layers, the Global Macro mainly affects the final output of MADRIX towards your LEDs.

2.1.2 The Script/Macro Editor

Introduction

The Script/Macro Editor is the tool to control and manage macros and scripts.



- ***Menu*** - Includes several submenus to perform various actions of the Script/Macro Editor.
- ***Toolbar -*** Includes various shortcuts to quickly execute the most common actions, such as Save, Load, or Compile.
- ***Text Field -*** The text field to enter source code and functions takes up the largest part of the Script/Macro Editor.

- ***Script Output -*** Provides a display to put out messages from the script itself.
  - Go to ***Edit > Clear Script Output*** or ***Edit > Clear Script Output / Compiler Messages*** to delete messages that are not needed anymore.
- ***Compiler Messages -*** Provides information of the Script/Macro Editor and displays errors or other messages.
  - Go to ***Edit > Clear Compiler Messages*** or ***Edit > Clear Script Output / Compiler Messages*** to delete messages that are not needed anymore.

Overview

Creating, Loading, and Saving Scripts

- Go to ***File > New*** to create a new template, which is the basis of a new script or macro. Or use the toolbar shortcut.
- Go to ***File > Open...*** to load an existing script or macro from an external file. Or use the toolbar shortcut.
- Go to ***File > Save*** or ***File > Save As...*** to store a script or macro on your harddisk or any other storage medium. Or use the toolbar shortcut.
- Go to ***File > Close*** to close the Script/Macro Editor. Or use the window shortcut.

Compiling and Executing Scripts

Before a script or macro can run, it needs to be compiled. While compiling the script, it is analyzed and translated into the format MADRIX understands internally.

- Go to ***Script > Compile And Run*** to compile the current source code. Or use the toolbar shortcut. 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.
- Go to ***Script > Compile And Save...*** to create a crypted, secured macro or script. Or use the toolbar shortcut.
- Go to ***Script > Run Script*** to start a script or macro once it has been compiled and stopped again. Or use the toolbar shortcut.
- Go to ***Script > Stop Script*** to halt the execution of a script. Or use the toolbar shortcut.

Special Keyboard Shortcuts

- ***Pos1/Home*** will at first press go to the beginning of the code, and at second press go to the beginning of the line.

File Types

.mas	***MADRIX Script File*** A script for the MAS Script Effect.
.macs	***Encrypted MADRIX Script File*** A compiled and encrypted script for the MAS Script Effect. 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. 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.
.mms	***MADRIX Effect Macro*** A Macro for an effect, a Storage Place Macro, or a Global Macro.
.mcm	***MADRIX Crypted Effect Macro*** A compiled and encrypted Macro for an effect, a Storage Place Macro, or a Global Macro. This is a macro which does not contain any source code, but only the runtime code. This means that with the help of compiled macros it is possible to share macros, but to let the source code remain a secret at the same time. Because a compiled macro does not contain any source code, only the macro'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.

2.1.3 Writing A Script

Introduction

- In general, a script or macro consists of many instructions, which you enter in the corresponding Script/Macro editor.
- The result is called source code, macro, or script.

The First Example

- A first example of a MADRIX Script can be seen below for the MAS Script Effect.
- You can simply copy and paste the source code and execute it.
- The example repeatedly writes a certain text line in the Script Output of the editor.

1) 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();
}
```

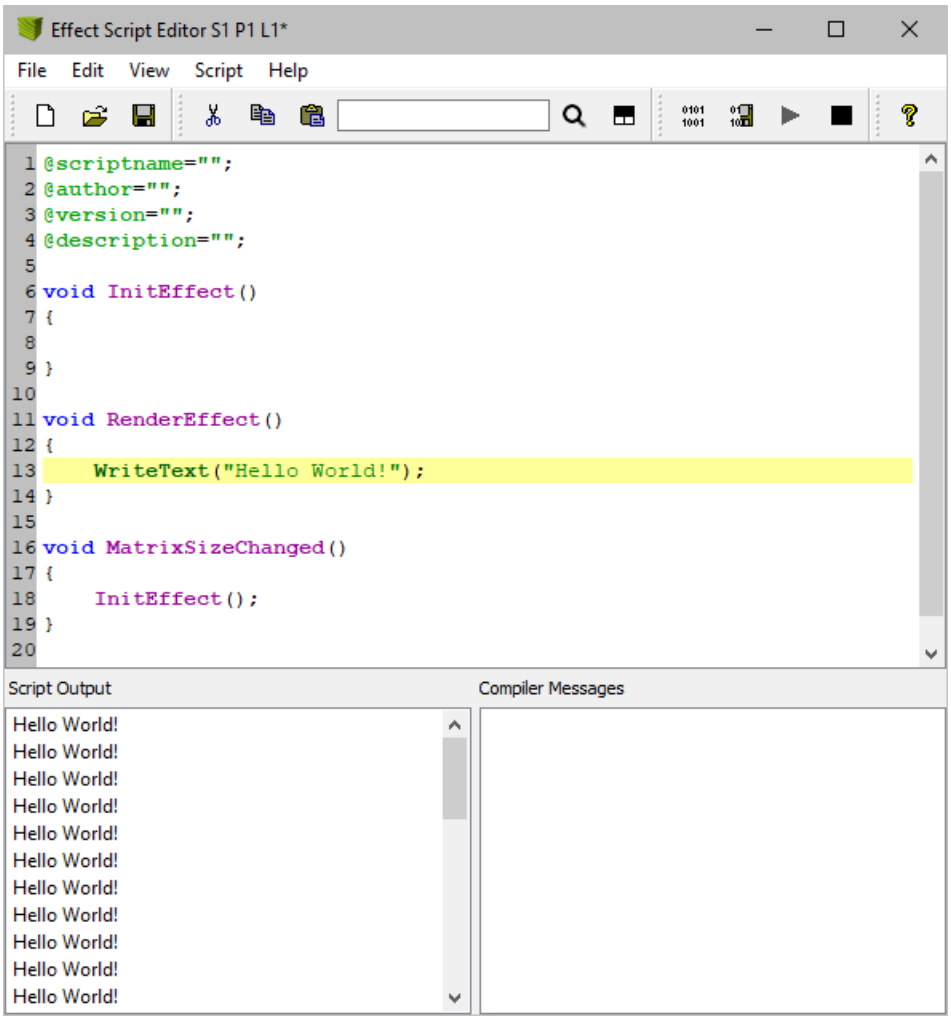
2) Compile the script (go to ***Script > Compile And Run***).

3) The function *WriteText(string text)* writes a given character string into the Script Output of the Script Editor. In this case it is "Hello World!".

4) You should see this message in the Script Output.

2.1.4 Syntax Highlighting

Overview



- The Script/Macro Editor supports syntax highlighting: as you can see from the screenshot above, MADRIX Script highlights code according to a color scheme.
- Highlighting will help you to read the source code, to distinguish different types of phrases (e.g., comments, data types, etc.), to quickly recognize functions, and to find errors.
- In addition, code structuring is enhanced since every line is automatically indented like the last line.

`@scriptname, SetFilter`	Represents functions that can only be used in certain areas of MADRIX Script (for example, Storage Place Macro).
`void`	Represents data types.
`InitEffect`	Represents predefined methods.
`FILTER_BLUR`	Represents global variables and constants.
`Filter, WriteText`	Represents functions that can be used in all areas of MADRIX Script (MAS Script Effect, Macros for Effects, Storage Place Macro, Global Macro)
`"Hello World!"`	Represents written text for the Script Output.

2.1.5 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

2.1.6 Functions

Working With Functions

- A script/macro in MADRIX Script consists of a set of functions.
- Some of these functions are required and called by MADRIX.
- Others may be used to split the script into smaller parts.
- Functions form 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

Parameters are always passed via copy by value (The exception are »**arrays**). 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.

Note: A reference is created for arrays. Hence, changing an array results also in changing the array 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 testArray[];
    int len = 2;
    testFunc(len, testArray);
}
```

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

Note: Passed parameters are always copied to a function, while this is not the case with arrays.

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

- Each macro or script includes a number of predefined functions called by MADRIX.
- If a function is not needed by a script, it is not necessary to implement it. 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, Effect Macros, Storage Place Macro, Global Macro) may include a different combination of these five functions as this is just an overview:

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

void MatrixSizeChanged()
```

- More information is available in the corresponding chapters.

  Learn more »**MAS Script Effect**

  Learn more »**Macros for Effects**

  Learn more »**Storage Place Macro**

  Learn more »**Global Macro**

Further Information

- There are a lot of functions which can be used in MADRIX Script for different functionality (e.g., to draw objects the matrix, get the data of the sound analysis, or mathematical functions).
- Learn more »**List Of Functions (Alphabetical Order)**

2.1.7 Data Types And 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.

```
@scriptname="SetRandomPixel";
@author="inoage";
@version="";
```

```
@description="";

void InitEffect()
{

}

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
}

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

2.1.7.1 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 = {0xFF};
date d = {24, 11, 1980};
```

If not all elements of a structure are initialized, the rest will be set to 0 or false or an empty string.

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 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 on the other hand are available within the whole script, beginning with when they are defined. 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 in a separate file or in 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 (for example, 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**.

2.1.7.2 Using Data Types

Primitive Data Types

Overview

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	Sample Values	Description
int	0, 3, -345, 234, 0x7f, 0xC001CAFE	32 bit data type. It stores integral numbers between -2 million and +2 million. Numbers starting with *0x* are interpreted as hexadecimal values.
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 internally and cannot be used to declare a variable. This data type only has to possible values, *true* or *false.* It is used for logical operations and 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

As stated above, 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 and values 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)* or 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	▪ int r ▪ int g ▪ int b ▪ int w ▪ int a	*color* stores a color value. There are 5 channels (red, green, blue, white, alpha) with values between 0 and 255. Example: color c = {255, 255, 0, 0}; Member examples: c.r, c.g, c.b, c.w, c.a Learn more »**List Of Global Variables And Constants** »**Script Example**
date	▪ int day ▪ int weekday ▪ int month ▪ int year	*date* stores a date. Values for *day* include 1 to 31 for a single day of the month. Values for *weekday* include: 0 = Sunday, 1 = Monday, ..., 6 = Saturday. Values for *month* include 1 to 12 for every single month of the year. Values for *year* include year dates. Example: date d = {24, 11, 1980}; Member examples: d.day, d.weekday, d.month, d.year »**Script Example**

time	▪ int hour ▪ int min ▪ int sec	*time* stores a certain time. Valid values are: hours: 0 .. 23, minutes: 0 .. 59, seconds: 0 .. 59. Example: time t = {12, 05, 00}; Member examples: t.hour, t.min, t.sec »**Script Example**
font	▪ 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	*font* stores a specific font face. *height* specifies the size of the font and requires an *integer* value. *width* specifies the wideness of the font and requires an *integer* value. *escapement* specifies the desired rotation angle in tenths of a degree and requires an *integer* value. *orientation* should be set to the same value as *escapement* and requires an *integer* value. *weight* specifies the weight of the font. Valid values are: **FONT_WEIGHT_DONTCARE** **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** *italic* specifies the sloping of the font and requires an *integer* value: *0* (off) or *1* (on). *underline* draws a line under the font and requires an *integer* value: *0* (off) or *1* (on). *strikeOut* draws a line through the middle of the font and requires an *integer* value: *0* (off) or *1* (on). *charset* 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** *outprecision* 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** *clipprecision* 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** **PRECIS_CLIP_EMBEDDED** *quality* specifies the quality of the font. Valid values are: **QUALITY_DEFAULT** **QUALITY_DRAFT** **QUALITY_PROOF** **QUALITY_NONANTIALIASED** **QUALITY_ANTIALIASED** **QUALITY_CLEARTYPE** **QUALITY_CLEARTYPE_NATURAL** *pitch* specifies the pitch of the font. Valid values for are: **PITCH_DEFAULT** **PITCH_FIXED** **PITCH_VARIABLE** **PITCH_MONO_FONT** *family* 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** *fontname* requires a *string*. For example "MS Sans Serif". »**Script Example**

shape	▪ int renderingMode ▪ int shapeAlignment ▪ int shapeRotation ▪ int blendingMode ▪ int originType ▪ float border ▪ float innerGlow ▪ float outerGlow ▪ int innerGlowInterpolation ▪ int outerGlowInterpolation ▪ float proportion ▪ float diagonalLength	*shape* stores specific information for shapes. Valid values for *renderingMode* are: **RENDERING_MODE_EXTENDED** **RENDERING_MODE_SIMPLE** Valid values for *shapeAlignment* are: **LOOKAT_FRONT** **LOOKAT_BACK** **LOOKAT_LEFT** **LOOKAT_RIGHT** **LOOKAT_TOP** **LOOKAT_BOTTOM** **LOOKAT_RANDOM** Valid values for *shapeRotation* are: **ROTATION_CCW_0** **ROTATION_CCW_90** **ROTATION_CCW_180** **ROTATION_CCW_270** **ROTATION_CW_0** **ROTATION_CW_90** **ROTATION_CW_180** **ROTATION_CW_270** *blendingMode* is only available for RENDERING_MODE_EXTENDED. Valid values are: **BLENDING_MODE_NONE** **BLENDING_MODE_ALPHA** Valid values for *originType* are: **ORIGIN_CENTER** **ORIGIN_GEOMETRIC_CENTER** **ORIGIN_FRONT** **ORIGIN_BACK** **ORIGIN_LEFT** **ORIGIN_RIGHT** **ORIGIN_TOP** **ORIGIN_BOTTOM** **ORIGIN_TOP_LEFT** **ORIGIN_TOP_RIGHT** **ORIGIN_BOTTOM_LEFT** **ORIGIN_BOTTOM_RIGHT** **ORIGIN_FRONT_LEFT** **ORIGIN_FRONT_RIGHT** **ORIGIN_BACK_LEFT** **ORIGIN_BACK_RIGHT** **ORIGIN_FRONT_TOP** **ORIGIN_FRONT_BOTTOM** **ORIGIN_BACK_TOP** **ORIGIN_BACK_BOTTOM** **ORIGIN_FRONT_TOP_LEFT** **ORIGIN_FRONT_TOP_RIGHT** **ORIGIN_FRONT_BOTTOM_LEFT** **ORIGIN_FRONT_BOTTOM_RIGH** **ORIGIN_BACK_TOP_LEFT**

		ORIGIN_BACK_TOP_RIGHT **ORIGIN_BACK_BOTTOM_LEFT** **ORIGIN_BACK_BOTTOM_RIGHT** *border* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *innerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *outerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. Valid values for *innerGlowInterpolation* are: **INTERPOLATION_TYPE_LINEAR** **INTERPOLATION_TYPE_EASE_BOUNCE_IN** **INTERPOLATION_TYPE_EASE_BOUNCE_OUT** **INTERPOLATION_TYPE_EASE_BOUNCE_INOUT** **INTERPOLATION_TYPE_EASE_CIRC_IN** **INTERPOLATION_TYPE_EASE_CIRC_OUT** **INTERPOLATION_TYPE_EASE_CIRC_INOUT** **INTERPOLATION_TYPE_EASE_CUBIC_IN** **INTERPOLATION_TYPE_EASE_CUBIC_OUT** **INTERPOLATION_TYPE_EASE_CUBIC_INOUT** **INTERPOLATION_TYPE_EASE_SINE_IN** **INTERPOLATION_TYPE_EASE_SINE_OUT** **INTERPOLATION_TYPE_EASE_SINE_INOUT** **INTERPOLATION_TYPE_EASE_EXPO_IN** **INTERPOLATION_TYPE_EASE_EXPO_OUT** **INTERPOLATION_TYPE_EASE_EXPO_INOUT** Valid values for *outerGlowInterpolation* are: See *innerGlowInterpolation* *proportion* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_CROSS_STRAIGHT, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, SHAPE_TYPE_3D_CROSS_STRAIGHT, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*. *diagonalLength* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*. »**Script Example**

2.1.7.3 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 cases a warning will be displayed in the ***Compiler Messages*** section 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)GetSpeedPitch(); //if the Pitch was 6.2, i is now 6
string s = (string)i;    //s now consists of the characters "6"
```

GetSpeedPitch() 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 of the 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 the value is not 0.
string	If the string starts with a number, it is converted into an integer/float value, otherwise the result is 0.		-	N/A	Is true if the string is not empty.
structure	N/A			Conversion between different structures: N/A.	N/A

Implicit Conversions Within Expressions

- If one of the two operands of an 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 an 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
int i = 3 / 4 * 2	0	*3 / 4* is an *integer* operation and results in *0*. Hence, the whole expression results in *0*.
float f = 3 / 4.0 * 2	1.5	Because *4.0* is a *float* value, *3* is converted into *float*, too. This way, *3 / 4.0* results in *0.75*. Then, *2* is also converted into *float* since the other operand is of type *float*, and the result is therefore *1.5*.
string test = "It is " + 9 + " o'clock."	"It is 9 o'clock."	Since the first operand is a *string*, *9* is also converted into a *string* and concatenated with the first *string*.
string test = 2 + 3 + "40"	"540"	*2* and *3* are both *integer* values and will be added up due to an *integer* operation. The second operation has a *string* as operand and therefore the other operand is converted into a *string* and both are concatenated together.

2.1.7.4 Arrays

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 arrays. They are declared like variables, followed by *[]*.

```
int    aiIntArray[];      //a 1-dimensional array of integer values
date   adDateArray[];     //a 1-dimensional array of dates
float  aafFloatArray[][]; //a 2-dimensional array of float values
```

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

```
//initialize an array with 5 integer values
int aiIntArray[] = {2, 3, 4, 5, 6};
```

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

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

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

The Length Or Size Of Arrays

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

```
int l = aiIntArray.length; //store the length of the array in l
```

Note: The length of an array is defined by the highest index that was used to access an element.

Multi-Dimensional Arrays

Up to this point, one-dimensional arrays were introduced. But multi-dimensional arrays are also possible. To declare a multi-dimensional array, 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/macro is running. It is also possible to initialize arrays using a list of elements for each dimension. Here are some examples:

```
int aaArray[][];     //a 2-dimensional array
int aaaArray3[][][]; //a 3-dimensional array

//initializes the array with two dimensions and three values each
int aaArry2[][] = {{2, 3, 4}, {6, 7, 8}};
```

```
//a 2-dimensional array of color elements
color aaCArray[][] = {
                      { {0, 0, 0, 0, 255}, {255, 255, 255}, {255, 255} },
                      { {255            }, {255, 255, 255} },
                      { {0, 255         }, {255, 255, 255} } };
```

The operator *[expression]* accesses a single element of an array, which for a multi-dimensional array may be another array. In order to access a single element, the applicable index must be used. For example, the fifth 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 array. Here are two examples:

```
int aaArray[][];
int aArray1[] = {1, 2, 3, 4, 5, 6}; //initialize the array

aaArray[0] = aArray1; //assign aArray1 to the first element of aaArray
aaArray[0][aArray1.length] = aArray1.length + 1;
aaArray[1][0] = 1;
aaArray[1][2] = 2;
```

Explanation: At the end of this example *aaArray* consists of two arrays 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 arrays in aaArray: " + aaArray.length);
WriteText("Number of elements in aaArray[0]: " + aaArray[0].length);
WriteText("Number of elements in aaArray[1]: " + aaArray[1].length);
```

Memory Management Of Arrays

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

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

After the assignment, the array 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 = 400.000.000 bytes, add up to around 382 megabytes (MB) of memory. So please pay attention when using very big arrays.

Full Example

This example just plays with the arrays and its content. It is just to show how to work with arrays and to get a feeling for them. A better example is given in the chapter »**Loops**

```
void InitEffect()
{
    int aaArray[][];
    int aArray1[] = {1, 2, 3, 4, 5, 6}; //initialize the array

    aaArray[0] = aArray1; //assign aArray1 to first element of aaArray
    aaArray[0][aArray1.length] = aArray1.length + 1;
    aaArray[1][0] = 1;
    aaArray[1][2] = 2;

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

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

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

void RenderEffect()
{

}
```

2.1.7.5 Strings And String Operations

Introduction

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

Compatibility

Strings in MADRIX 5 support the UTF-8 encoding standard. That mainly means that a variety of characters of different languages as well as special characters can be used.

Several special characters can be masekd to make sure that they are included in the text. Otherwise, the script language will interpret them as parts of the language with much different functionality and not part of the text.

- A \ can be masked with "\\".
- A " can be masked with "\"".
- A new line/an end-of-line can be masked with "\n".
- A tab/tabulator can be masked with "\t".
- A carriage return can be masked with "\r".

Operations On Strings

Assigning Data Types

It is possible to assign *integer*, *float,* and *string* values to another *string* as 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 a single character 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* contain more or less than one character:

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

Comparing Two Strings

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

Operator	Description
str1 == str2	Checks if the strings are equal.
str1 != str2	Checks if the strings are not equal.
str1 < str2	Checks if the first string is less than the second string.
str1 <= str2	Checks if the first string is less or equal to the second string.
str1 > str2	Checks if the first string is greater than the second string.
str1 >= str2	Checks if the first string is greater or equal to the second string.

Please note: The sorting order of the strings is case-sensitive and depends on the contained ASCII characters. Therefore, all upper case characters are "less" than the set of lower case characters, e.g. the string array *{ "and", "Alice", "Bob", "and me" }* results in the ascendingly sorted string array *{ "Alice", "Bob", "and", "and me" }*.

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

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

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

It is also possible to compare a single character of a *string* with an *integer* number:

```
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 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;
    //it 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 for the labels of cases. 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)	Searches for the *substring* within *text* starting at *startIndex*. The function returns an index that describes the position at which the *substring* begins. »**Description** Example: findstring(0, "`Hello World!`", "World") returns 6. If the *substring* is not found within *text*, *-1* is returned.

string **substring**(string text, int start, int count)	The function extracts *count* characters from the given *text* starting with *start*. If *count* is *-1*, all characters of the string starting at *start* are returned. »**Description** E.g.: substring("Hello", 0, 2) returns "He".
int **rfindstring**(int startIndex, string text, string substring)	This functions looks for the *substring* in the given text from its end to the beginning. The function starts its search at a specified position of the entire *text* using *startIndex* and returns an index that describes the position at which the *substring* begins. If the substring was not found, *-1* is returned.
int **startswith**(string text, string substring)	This function checks if the string *text* starts with the given *substring*. If *text* starts with *substring*, 1 (**TRUE**) is returned, otherwise 0 (**FALSE**).
int **endswith**(string text, string substring)	This function checks if the string *text* ends with the given *substring*. If *text* ends with *substring*, 1 (**TRUE**) is returned, otherwise 0 (**FALSE**).
int **isalnum**(string text)	Returns 1 (**TRUE**) if the given string contains only characters and figures and if its length is greater then 0. Otherwise, 0 (**FALSE**) is returned.
int **isalpha**(string text)	Returns 1 (**TRUE**) if the given string contains only characters and if its length is greater than 0, otherwise 0 (**FALSE**) is returned.
int **isnum**(string text)	Returns 1 (**TRUE**) if the given *text* represents a number. This may be an integer number or a floating point number (e.g. 1.3). Otherwise, it returns 0 (**FALSE**).
void **tolower**(string text)	Converts each character of the given *string* into a lower-case character.
void **toupper**(string text)	Converts each character of the given *string* into an upper-case character.
string **hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are lower-cased (a-f).
string **Hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are upper-cased (A-F).
void **strip**(string text)	Removes leading and ending white spaces, like space, tabulator, line feeds, etc. from the given *string*.
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 *delemiter*. The result is returned in *reslist*. »**Description**

Tokenizing Strings

The function **tokenize** enables you to split a string into smaller parts separated by specified delimiters. 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 comma 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 displayed 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

2.1.8 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 operations 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 Operation

To assign any value to a variable the assignment 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;
```

Arithmetical Operations

'++'/'--' Operator

The "++" and "--" operators are unary operators and only defined for usage with variables of the type *int*. These operators are known from C/C++ and can be used in two different ways, as prefix and as suffix operators (i++ and ++i).

The prefix operator increments/decrements the current value of the given variable and results in the new value afterwards.

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

In the end, *i* is 10. In the first line, *4* is assigned to the integer value *i*. In the second line, the expression *++i* results in 5. The multiplication therefore is "5 * 2" which results in 10. This value is assigned to *i* and therefore, *i* = 10.

The suffix operator results in the current value of the given variable and increments/decrements it afterwards.

```
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 * 4.2);
```

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 Operators

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.

'%' Operator

The "%" operator calculates the *integer* remainder of an *integer* division. It is called the modulo operator.

```
10 % 2;
```

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

Concatenating Strings

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

```
string s = "Hello " + "World!";
```

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 = i + "th run";
```

This example results in "4th run". **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 expected "34th run".

Logical Operations

'!' Operator

The "!" operator is a unary operator, which logically negates the value of the given expression. *false* becomes *true* and vice versa.

```
!3
!(3 > 4)
!"Hello World!"
```

'&&'/'||' Operator

The "&&" and "||" operators are logical operators. They need operands of the data type *bool* and always result in a *bool* value. "&&" is called the logical AND operator, "||" is called the logical OR operator.

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 are some examples for using those operators:

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

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

Comparison Operators

With comparison operators you can test two expressions for a certain relation. Possible comparisons (and operators) are "less than" (<), "less equal" (<=), "greater than" (>), "greater 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 (comparison operator). Comparison operators always return a *bool* value, the comparison is either *true* or *false*.

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

Bit Operations

Bit operations manipulate *integer* values bitwise. In MADRIX Script *integer* values always consist of 32 bits. The functionality of the following operators becomes clearer if you imagine a decimal number as a series of 32 bits, for example: $(5)_{10}$ = $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101)_2$ = $(101)_2$. The leftmost (most significant) bit specifies the sign of an *integer* value, for example: $(-5)_{10}$ = $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011)_2$.

'~' Operator

The "~" operator is a unary operator, which negates each bit of the given value. 0 becomes 1 and vice versa.

```
~5      // (5)  = (0000 0000 0000 0000 0000 0000 0000 0101) -> (1111 1111 1111 1111 1111 1111 1111 1
~(-5)   // (-5) = (1111 1111 1111 1111 1111 1111 1111 1011) -> (0000 0000 0000 0000 0000 0000 0000 0
```

'^'/'&'/'|' Operator

The "^", "&" and "|" operators are bit operators. "^" is called the bitwise XOR operator, "&" is called the bitwise AND operator, "|" is called the bitwise OR operator. Here are some examples:

```
int i = 6 & 5;        // i = (110) & (101) = (100) = (4)
int j = 3 | i;        // j = (011) | (100) = (111) = (7)
int k = i ^ 5;        // k = (100) | (101) = (001) = (1)
```

Make sure that you can distinguish the meaning of a single ampersand (bitwise AND operator) and a double ampersand (logical AND operator). Bit operators combine the corresponding bits of *integer* values while logical operators combine *bool* values.

```
int i = 6 & 5;        // i is 4 (see example above)
int j = 6 && 5;       // j is true (1)
```

The difference is the same for "|" and "||".

Shifting Bits

The shifting operators "<<", ">>" and ">>>" move the bits of an *integer* value by a specified offset to the left or right.

```
int i = 6 << 1;       // i = (110) << (1) = (1100) = (12)
int j = 6 >> 2;       // j = (110) >> (2) = (1)
```

The right shifting operators ">>" and ">>>" cause the same results for positive values, but for negative numbers only ">>>" preserves the sign.

```
int i = -6 >> 2;      // i = (1111 1111 1111 1111 1111 1111 1111 1010) >>  (2) = (0011 1111 1111 111
int j = -6 >>> 2;     // j = (1111 1111 1111 1111 1111 1111 1111 1010) >>> (2) = (1111 1111 1111 111
```

Again, pay attention to the difference between the shifting operators ("<<", ">>", ">>>") and the comparison operators ("<", ">").

Additional Assignment Operations

To have less code and increase readability, there are additional assignment operators: +=, -=, *=, /=, %=, ^=, |=, &=, <<=, >>=, >>>=.

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

is the same as

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

and so 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 for the MAS Script Effect 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;
    SetAsync(true);
    SetBpm(600.0);
}

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

2.1.9 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.

2.1.9.1 'If' And 'Else If' Statements

'If' Construct

Very often it is necessary to make decisions within a script/macro. 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, as 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* can be directly followed by an *if*. The *else if* structure is very useful to make code with 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 and paste it in the function *RenderEffect*.

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

Full Example

The following example for the MAS Script Effect 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;
    SetAsync(true);
    SetBpm(300);
}

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);
        }//for[each line]
}
```

2.1.9.2 'Switch' Statements

Introduction

If it is required to compare an integer variable with a lot of different values, using the *if*-statement may be very impractical. 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. 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 from the last example for the MAS Script Effect in the following way:

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

void InitEffect()
{
    g_iCol = 0;
    SetAsync(true);
    SetBpm(300);
}

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 state]

        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 mentioned earlier, 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 else; break;
            ...
        }
    }
}
```

2.1.9.3 'For' And '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);
```

```
        x++;
    }
```

'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 after running the *statement* and before checking the *condition*. 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(x++ >= GetMatrixWidth())
        break; //leave loop now!

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

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 for the MAS Script Effect draws parts of a cosinus curve onto the matrix while changing the background color.

```
void InitEffect()
{
    SetAsync(true);
    SetBpm(300);
}

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 for the MAS Script Effect 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 array
        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;
            }

        SetAsync(true);
        SetBpm(300);
    }
```

2.1.10 Reading From External Files

Asynchronous File Reading

The function

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

reads the content from a *file* with a certain *encoding* as text 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: "https://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", txt);
```

Here is an example which reads some numbers from a file (to be found at C:/temp/src.txt) and renders a curve on the matrix.

```
float g_pos[] = {0.8, 0.5};
```

```
string file = "C:/temp/src.txt";
string txt;

void InitEffect()
{
    SetAsync(true);
    SetBpm(300);
}

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 USB 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
int **FILE_OK**	The function could read the file without problems.
int **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 an HTTP request, this error is returned when the file does not exist on the host.
int **FILE_ERROR**	This is returned if any error occurred while reading the content of a local file.
int **INVALID_HOST**	This value is returned if the file was an HTTP request and the specified host does not exist.
int **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.

In addition, *ReadAsync* can take a third parameter to specify the file encoding. If the parameter is omitted, an automatic detection is applied. The following values are valid:

Value	Description
int **FILE_ENCODE_AUTO_DETECT**	Applies an automatic detection. This is the default. If the file starts with the special characters BOM (byte order mark), the corresponding file encoding is expected. Otherwise, an ANSI encoded file is expected.
int **FILE_ENCODE_ANSI**	Expects an ANSI encoded file.
int **FILE_ENCODE_UTF8**	Expects a UTF-8 encoded file not starting with the special characters BOM (byte order mark).
int **FILE_ENCODE_UTF8_BOM**	Expects a UTF-8 encoded file starting with the special characters BOM (byte order mark).
int **FILE_ENCODE_UTF16_LE**	Expects a UTF-16 Little Endian encoded file starting with the special characters BOM (byte order mark).

UTF (Unicode Transformation Format) encoded files are recommended in order to handle special characters (e.g. from Asian languages) correctly. The actual encoding of a text file can be figured out or even changed by using an appropriate text editor, for example.

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 for SCE Ticker / Scrolling Text:

```
@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 1000 ms 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 500 ms 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 begin 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

Overview

Up to now, MADRIX Script is not designed to operate on strings. There aren't many 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 Script. It may be better to have external tools (e.g. python scripts, PHP, or Visual basic scripts) which prepare files for MADRIX Script in order to have faster scripts.

Another interesting possibility is to have interactive scripts. Imagine a small application which retrieves input from a 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";
?>
```

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

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 started with "/*" and they end with "*/".

```
/* the comment starts here

the comment ends here */
```

2.1.12 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:

```
@scriptname="Name of the script"
@author="John Smith";
@version="1.24";
@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.

2.2 Advanced Techniques

2.2.1 Draw And Render Functions

2.2.1.1 Pixels Vs. Vectors

Introduction

- In a lot of cases, MADRIX Script offers both a pixel-based or a vector-based version of a function.
- When working with a matrix and pixels, the dimensions of matrix and effects, such as a vertical line, are absolute and adjusted to that particular matrix.
  - For example, you might want to draw a 20x2 horizontal line on a 50x50 matrix.
- However, this can be a disadvantage since you might want to write a macro/script that can be applied to a multitude of matrixes with different sizes or someone else might want to use your script. In this case using relative values can help a lot, since effects are scalable then.
  - For example, the horizontal line should always be of 50% width and 5% height.
  - Valid values for relative values range from 0.0 to 1.0.

Rules Of Calling Functions

Overview

- Functions of MADRIX Script often tell you what their purpose is and which kind of values they require (absolute or relative).
- The name of a draw or render function often starts with a description of what it does, like *Draw* or *Shift*. It is followed by *Pixel* or *Vector* to describe whether the function requires pixel coordinates or relative coordinates. The final part describes the application of the function, like *Line* or *Shape*. Here are two examples:

- **`DrawPixelLine - Draws a line using absolute values`**
- **`DrawVectorShape - Draws a shape using relative values`**
- **`ShiftPixelMatrix - Moves an area of the matrix using absolute values`**

Using Absolute Coordinates

- Functions with *Pixel* in their name operate on absolute pixel values. Imagine 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 every matrix.
  - On the one hand, the rectangle appears to be very small on large matrices.
  - On the other hand, on small matrices it seems to be a big rectangle because it fills a larger area of the matrix.

Using Relative Coordinates

- Functions with *Vector* in their name operate on relative values. 0.0, 0.0 stands for the top left corner of the effects matrix. 1.0, 1.0 describes its bottom right corner, or its whole width 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.
  - In contrast, the usage of absolute coordinates dictates the utilization of the same amount of pixels.

2.2.1.2 Using Filters

Introduction

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

The functions look as follows:

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

- These functions basically work wherever a matrix is used, which is the case with the MAS Script Effect, Macros for Effects, the Storage Place Macro, and the Global Macro. Insert one of these functions 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.

Filter Constants

Filters are divided into several groups, such as color correction filters or style filters, for example.

Identifier / Filter (FX)	Description
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_SHARPEN**	This filter sharpens the output.
Brightness Graph Filters	

Identifier / Filter (FX)	Description
int **FILTER_BRIGHTNESS_GRAPH_XYZ**	This filter displays the distribution of the color intensity on the y-axis.
int **FILTER_BRIGHTNESS_GRAPH_XZY**	This filter displays the distribution of the color intensity on the z-axis.
int **FILTER_BRIGHTNESS_GRAPH_YXZ**	This filter displays the distribution of the color intensity on the x-axis.
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_GRAYSCALE**	The grayscale filter to render the matrix grayscale, i.e. in gray colors.
int **FILTER_INVERT_COLOR**	The invert color filter to invert every color channel.
Color Swap Filters	
int **FILTER_RGB_TO_BGR**	The filter reorders the color channels to blue, green, red. The white color channel remains unchanged.
int **FILTER_RGB_TO_BRG**	The filter reorders the color channels to blue, red, green. The white color channel remains unchanged.
int **FILTER_RGB_TO_GRB**	The filter reorders the color channels to green, red, blue. The white color channel remains unchanged.
int **FILTER_RGB_TO_GBR**	The filter reorders the color channels to green, blue, red. The white color channel remains unchanged.
int **FILTER_RGB_TO_RBG**	The filter reorders the color channels to red, blue, green. The white color channel remains unchanged.
int **FILTER_RGBW_TO_WBGR**	The filter reorders the color channels to white, blue, green, red.
int **FILTER_RGBW_TO_WBRG**	The filter reorders the color channels to white, blue, red, green.
int **FILTER_RGBW_TO_WGRB**	The filter reorders the color channels to white, green, red, blue.
int **FILTER_RGBW_TO_WGBR**	The filter reorders the color channels to white, green, blue, red.
int **FILTER_RGBW_TO_WRBG**	The filter reorders the color channels to white, red, blue, green.

Identifier / Filter (FX)	Description
int **FILTER_RGBW_TO_WRGB**	The filter reorders the color channels to white, red, green, blue.
int **FILTER_RGBW_TO_BWGR**	The filter reorders the color channels to blue, white, green, red.
int **FILTER_RGBW_TO_BWRG**	The filter reorders the color channels to blue, white, red, green.
int **FILTER_RGBW_TO_GWRB**	The filter reorders the color channels to green, white, red, blue.
int **FILTER_RGBW_TO_GWBR**	The filter reorders the color channels to green, white, blue, red.
int **FILTER_RGBW_TO_RWBG**	The filter reorders the color channels to red, white, blue, green.
int **FILTER_RGBW_TO_RWGB**	The filter reorders the color channels to red, white, green, blue.
int **FILTER_RGBW_TO_BGWR**	The filter reorders the color channels to blue, green, white, red.
int **FILTER_RGBW_TO_BRWG**	The filter reorders the color channels to blue, red, white, green.
int **FILTER_RGBW_TO_GRWB**	The filter reorders the color channels to green, red, white, blue.
int **FILTER_RGBW_TO_GBWR**	The filter reorders the color channels to green, blue, white, red.
int **FILTER_RGBW_TO_RBWG**	The filter reorders the color channels to red, blue, white, green.
int **FILTER_RGBW_TO_RGWB**	The filter reorders the color channels to red, green, white, blue.
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.
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.

Identifier / Filter (FX)	Description
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.
Kaleidoscope Filters	
int **FILTER_KALEIDOSCOPE_6X**	The ***6x*** mix mode.
int **FILTER_KALEIDOSCOPE_8X**	The ***8x*** mix mode.
int **FILTER_KALEIDOSCOPE_12X**	The ***12x*** mix mode.
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_EMBOSS**	The emboss filter to create an image with just highlights and shadows.
int **FILTER_EMBOSS_POPUP**	The emboss popup filter to create an image with just highlights and shadows depending on the motif.
Mirror 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°.
int **FILTER_INVERT_D_MATRIX**	The filter flips the matrix regarding the depth.

Identifier / Filter (FX)	Description
int **FILTER_INVERT_HD_MATRIX**	The filter flips the matrix horizontally and regarding the depth.
int **FILTER_INVERT_VD_MATRIX**	The filter flips the matrix vertically and regarding the depth.
int **FILTER_INVERT_HVD_MATRIX**	The filter flips the matrix horizontally, vertically, and regarding the depth.
Swap Filters	
int **FILTER_SWAP_H_1X**	This filter divides the matrix into 2 columns and transposes them.
int **FILTER_SWAP_H_2X**	This filter divides the matrix into 4 columns and transposes them.
int **FILTER_SWAP_H_3X**	This filter divides the matrix into 8 columns and transposes them.
int **FILTER_SWAP_H_4X**	This filter divides the matrix into 16 columns and transposes them.
int **FILTER_SWAP_H_5X**	This filter divides the matrix into 32 columns and transposes them.
int **FILTER_SWAP_V_1X**	This filter divides the matrix into 2 rows and transposes them.
int **FILTER_SWAP_V_2X**	This filter divides the matrix into 4 rows and transposes them.
int **FILTER_SWAP_V_3X**	This filter divides the matrix into 8 rows and transposes them.
int **FILTER_SWAP_V_4X**	This filter divides the matrix into 16 rows and transposes them.
int **FILTER_SWAP_V_5X**	This filter divides the matrix into 32 rows and transposes them.
int **FILTER_SWAP_HV_1X**	This filter divides the matrix into 2 columns and rows, and transposes them.
int **FILTER_SWAP_HV_2X**	This filter divides the matrix into 4 columns and rows, and transposes them.
int **FILTER_SWAP_HV_3X**	This filter divides the matrix into 8 columns and rows, and transposes them.
int **FILTER_SWAP_HV_4X**	This filter divides the matrix into 16 columns and rows, and transposes them.
int **FILTER_SWAP_HV_5X**	This filter divides the matrix into 32 columns and rows, and transposes them.
int **FILTER_SWAP_D_1X**	This filter divides the matrix into 2 depth segments and transposes them.

Identifier / Filter (FX)	Description
int **FILTER_SWAP_D_2X**	This filter divides the matrix into 4 depth segments and transposes them.
int **FILTER_SWAP_D_3X**	This filter divides the matrix into 8 depth segments and transposes them.
int **FILTER_SWAP_D_4X**	This filter divides the matrix into 16 depth segments and transposes them.
int **FILTER_SWAP_D_5X**	This filter divides the matrix into 32 depth segments and transposes them.
int **FILTER_SWAP_HD_1X**	This filter divides the matrix into 2 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_2X**	This filter divides the matrix into 4 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_3X**	This filter divides the matrix into 8 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_4X**	This filter divides the matrix into 16 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_5X**	This filter divides the matrix into 32 columns and depth segments, and transposes them.
int **FILTER_SWAP_VD_1X**	This filter divides the matrix into 2 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_2X**	This filter divides the matrix into 4 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_3X**	This filter divides the matrix into 8 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_4X**	This filter divides the matrix into 16 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_5X**	This filter divides the matrix into 32 rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_1X**	This filter divides the matrix into 2 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_2X**	This filter divides the matrix into 4 columns, rows and depth segments, and transposes them.

Identifier / Filter (FX)	Description
int **FILTER_SWAP_HVD_3X**	This filter divides the matrix into 8 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_4X**	This filter divides the matrix into 16 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_5X**	This filter divides the matrix into 32 columns, rows and depth segments, and transposes them.

Example

Paste the following example into the Global 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);
}
```

2.2.1.3 Using Mix Modes

Introduction

The function *SetMixMode* offers the possibility to set the mix mode of a Layer. 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.

Mix Mode Constants

Identifier / Mix Mode	Description
int MIXMODE_NORMAL	The ***Normal*** mix mode.
int MIXMODE_DARKEN	The ***Darken*** mix mode.
int MIXMODE_MULTIPLY	The ***Multiply*** 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_LINEARDODGE	The ***Linear Dodge*** mix mode.
int MIXMODE_OVERLAY	The ***Overlay*** mix mode.
int MIXMODE_SOFTLIGHT	The ***Soft Light*** 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 ***Hard Mix*** mix mode.
int MIXMODE_DIFFERENCE	The ***Difference*** mix mode.
int MIXMODE_EXCLUSION	The ***Exclusion*** mix mode.
int MIXMODE_MASK	The ***Mask*** mix mode.

MADRIX 3.X To MADRIX 5.X Migration Hints

The following constants are not supported anymore. Please follow the hints to migrate your macros.

int **MIXMODE_AND**	Use **MIXMODE_MULTIPLY** instead.
int **MIXMODE_OR**	Use **MIXMODE_LIGHTEN** instead.
int **MIXMODE_XOR**	Use **MIXMODE_DIFFERENCE** instead.
int **MIXMODE_NAND**	This constant is not supported anymore.
int **MIXMODE_NOR**	This constant is not supported anymore.

2.2.1.4 Mapping / Tiling / Rotation

Overview

- MADRIX provides advanced mapping functionality, including mapping, tiling, and rotation.
- The following functions are available for the MAS Script Effect as well as for Macros for Effects.
- The Storage Place Macro provides its own functions. Learn more: »**Storage Place Macro: Functions**

Functions

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Global Macro
int **MapDlgIsMapped**()	Retrieves if the Layer is mapped.	+	+		
void **MapDlgGetMapVector**(float map[])	Retrieves the map settings for the Layer using relative values. The values are saved in a array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapVector**(float x, float y, float w, float h)	Maps the Layer to a certain area of the matrix using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgGetMapVector3D** (float map[])	Retrieves the map settings for the Layer in 3D using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapVector3D** (float x, float y, float z, float w, float h, float d)	Maps the Layer to a certain area of the matrix in 3D using relative values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetMapPixel**(int map[])	Retrieves the map settings for the Layer using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapPixel**(int x, int y, int w, int h)	Maps the Layer to a certain area of the matrix using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetMapPixel3D**(int map[])	Retrieves the map settings for the Layer in 3D using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapPixel3D**(int x, int y, int z, int w, int h, int d)	Maps the Layer to a certain area of the matrix in 3D using pixel values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
int **MapDlgGetMapMode**()	Retrieves the currently used ***Map Mirror Mode*** of the Layer. See **below** for a list of constants.	+	+		
void **MapDlgSetMapMode**(int mode)	Sets the ***Map Mirror Mode*** for the Layer. See **below** for a list of constants.	+	+		

void **MapDlgGetTileVector**(float tile[])	Retrieves the ***Tiling*** settings for the Layer using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTileVector**(float x, float y, float w, float h)	Tiles the Layer to a certain area of the mapping using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetTileVector3D** (float tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTileVector3D** (float x, float y, float z, float w, float h, float d)	Tiles the Layer in a certain area of the mapping in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetTilePixel**(int tile [])	Retrieves the ***Tiling*** settings for the Layer using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTilePixel**(int x, int y, int w, int h)	Tiles the Layer in a certain area of the mapping using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgGetTilePixel3D**(int tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTilePixel3D**(int x, int y, int z, int w, int h, int d)	Tiles the Layer in a certain area of the mapping using pixel values. *x, y and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetTileOffsetVector** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetVector** (float x, float y)	Sets the ***Tiling Offset*** of the Layer using relative values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgGetTileOffsetVector3D** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetVector3D** (float x, float y, float z)	Sets the ***Tiling Offset*** of the Layer in 3D using relative values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
void **MapDlgGetTileOffsetPixel** (int offset[])	Retrieves the ***Tiling Offset*** of the Layer using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetPixel** (int x, int y)	Sets the ***Tiling Offset*** of the Layer using pixel values. *x* describes Offset X, while *y* represents Offset Y.	+	+		

void **MapDlgGetTileOffsetPixel3D**(int offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetPixel3D**(int x, int y, int z)	Sets the ***Tiling Offset*** of the Layer in 3D using pixel values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
int **MapDlgGetTileMode**()	Retrieves the currently used ***Tile Mode*** of the Layer. See **below** for a list of Tile Mode constants.	+	+		
void **MapDlgSetTileMode**(int mode)	Sets the ***Tile Mode*** of the Layer. See **below** for a list of Tile Mode constants.	+	+		
void **MapDlgGetRotationVector** (float rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values. The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationVector** (float x, float y, float z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values.	+	+		
void **MapDlgGetRotationDegree** (int rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationDegree** (int x, int y, int z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.	+	+		
float **MapDlgGetRotationXVector**()	Retrieves the ***Rotation*** for the X-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationXVector** (float value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using degree values.	+	+		
int **MapDlgGetRotationXDegree** ()	Retrieves the ***Rotation*** for the X-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationXDegree**(int value)	Sets the ***Rotation*** *value* for the X-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationXMode**()	Retrieves the ***Rotation*** mode for the X-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		

void **MapDlgSetRotationXMode** (int mode)	Sets the ***Rotation*** mode for the X-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationYVector**()	Retrieves the ***Rotation*** for the Y-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationYVector** (float value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationYDegree** ()	Retrieves the ***Rotation*** for the Y-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationYDegree**(int value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationYMode**()	Retrieves the ***Rotation*** mode for the Y-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationYMode** (int mode)	Sets the ***Rotation*** mode for the Y-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationZVector**()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationZVector** (float value)	Sets the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationZDegree** ()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationZDegree**(int value)	Sets the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationZMode**()	Retrieves the ***Rotation*** mode for the Z-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationZMode** (int mode)	Sets the ***Rotation*** mode for the Z-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgGetRotationMode** (int mode[])	Retrieves the ***Rotation*** mode for axes *x, y, and z* of the Layer. See **below** for a list of Rotation Mode constants. The values are saved in an array (*mode[]*). ▪ mode[0] = X-coordinate (X-Axis) ▪ mode[1] = Y-coordinate (Y-Axis) ▪ mode[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationMode** (int x, int y, int z)	Sets the ***Rotation*** ***Mode*** for axes *x, y, and z* of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
int **MapDlgGetAntiAliasing**()	Retrieves the ***Anti-Aliasing*** mode of the Layer. See **below** for a list of Anti-Aliasing Mode constants.	+	+		
void **MapDlgSetAntiAliasing**(int aaLevel)	Sets the ***Anti-Aliasing*** mode of the Layer. See **below** for a list of Anti-Aliasing Mode constants.	+	+		

Map Mirror Mode Constants

Constant	Description
int MAP_MIRROR_NONE	Sets no Mirror 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 horizontally and vertically.
int MAP_MIRROR_D	Mirrors the content of the matrix in the depth.
int MAP_MIRROR_HD	Mirrors the content of the matrix horizontally and in the depth.
int MAP_MIRROR_VD	Mirrors the content of the matrix vertically and in the depth.
int MAP_MIRROR_HVD	Mirrors the content of the matrix horizontally, vertically, and in the depth.

Map Tile Mode Constants

Constant	Description
int MAP_TILE_NONE	Sets no Tile Mode.
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 horizontally and vertically.
int MAP_TILE_MIRROR_D	Mirrors tiles in the depth.
int MAP_TILE_MIRROR_HD	Mirrors tiles horizontally and in the depth.
int MAP_TILE_MIRROR_VD	Mirrors tiles vertically and in the depth.
int MAP_TILE_MIRROR_HVD	Mirrors tiles horizontally, vertically, and in the depth.

Map Rotation Mode Constants

Constant	Description
int MAP_ROTATION_FIXED	The rotation value is fixed and not animated.

Constant	Description
int **MAP_ROTATION_LOOP**	The rotation value is used for a looped animation.

Map Anti-Aliasing Mode Constants

Constant	Description
int **MAP_AA_NONE**	Sets no anti-aliasing.
int **MAP_AA_2X**	Sets simple anti-aliasing (requires some performance).
int **MAP_AA_4X**	Sets complex anti-aliasing (requires a lot performance).

2.2.1.5 'ShiftMatrix'

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

```
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 *0.0* and *1.0*.

- *x*, *y*, *w*, and *h* define the area that should be shifted.
- *step* defines how wide the content should be shifted into a given direction, which is 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 for the MAS Script Effect fills the matrix with yellow and draws a red cross onto it during initialization. During the rendering of the content, 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);
    DrawVectorShape(col,DRAWSHAPE_CROSS,0.0,0.0,0.0,1.0,1.0,1.0);
}
```

```
void RenderEffect()
{
    ShiftPixelMatrix(0,0,GetMatrixWidth(),GetMatrixHeight(),SHIFT_DOWN,1);
}
```

The script also demonstrates an important behavior of the function. As you can see, the matrix remains yellow, but the cross moves to the bottom. Furthermore, red lines will be drawn on the left and the 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 content is moved into the given direction, the original matrix is left unchanged. This leaves the first pixel line unchanged in our example.

2.2.1.6 'DrawPixelArea'

Introduction

Three 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 similar function which retrieves the content of a specific screen area. Third, there is a function which is able to draw pixels onto the matrix using the obtained data array(s) as source.

Retrieving Content Of The Matrix

The function *GetPixelArea* retrieves data from the matrix and stores it into a 2-dimensional array 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)
```

The function *GetPixelArea3D* retrieves data from the matrix and stores it into a 3-dimensional array of colors. Data is stored in the background at a certain position of the virtual matrix.

```
void GetPixelArea3D(matrix[][][], int xSrc, int ySrc, int zSrc, int w, int h, int d, int x
```

Explanation:

- *matrix[][]* is a 2-dimensional array of colors in which the content of the virtual matrix is saved.
  matrix[][][] is a 3-dimensional array of colors in which the content of the virtual matrix is saved.
- *xSrc, ySrc, zSrc* describe the position of the source area (upper left front corner). The default values are *0*.
- *w, h, d* describe the width, height and depth of the source area. The default values are *-1*.
  A value of *-1* means that the whole width, height or depth of the virtual matrix will be retrieved (the complete matrix).
- *xDst, yDst, zDst* describe the position of the destination area (upper left front corner). The default values are *0*.
  The retrieved array 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[3D]* several times and to draw them only once by calling *DrawPixelArea[3D]* one time (see below).
  If your array 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 array, it is possible to just call:

```
color matrix[][];      //3D: color matrix[][][];
GetPixelArea(matrix); //3D: GetPixelArea3D(matrix);
```

- **Summary:** You can store the complete virtual matrix or only parts of it in an array. The array can be stored at the default position or an individual position. The array is stored on a matrix in the background. The background matrix acts as source for *DrawPixelArea[3D]*. It can contain several arrays.

Retrieving Content Of The Screen

The function *CaptureScreen* retrieves data from the screen and stores it into a 2-dimensional array of colors. Data is stored in the background.

```
void CaptureScreen(matrix[][], int xSrc, int ySrc, int w, int h)
```

Explanation:

- *matrix[][]* is a 2-dimensional array of colors in which the content of the screen is saved.
- *xSrc, ySrc* describe the position of the source area (upper left corner).

- *w*, *h* describe the width and height of the source area.
  The retrieved array will be stored in the background.
  If your array is larger than your target destination allows, it will be reduced to fit the size of the virtual matrix.
- The function *CaptureScreen* can have a notable performance impact, depending on the size of the screen area and the hardware setup.
- **Summary:** You can store the complete screen area or only parts of it in an array. The array is stored on a matrix in the background. The background matrix acts as source for *DrawPixelArea*.

Drawing Content Onto The Matrix

The function *DrawPixelArea* copies data from a 2-dimensional array 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
```

The function *DrawPixelArea3D* copies data from a 3-dimensional array of colors from the background and renders it onto the actual matrix.

```
void DrawPixelArea3D(matrix[][][], int xDst, int yDst, int zDst, int w, int h, int d, int
```

Explanation:

- *matrix[][]* is a 2-dimensional array of colors that holds the source for *DrawPixelArea*. Use *GetPixelArea* or *CaptureScreen* as described above to retrieve the data.
  matrix[][][] is a 3-dimensional array of colors that holds the source for *DrawPixelArea3D*. Use *GetPixelArea3D* as described above to retrieve the data.
- *xDst*, *yDst*, *zDst* describe the destination area. The default values are *0*.
  This allows you to draw the array onto the default position of your virtual matrix or an individual position.
  If the source array is larger than your target destination allows, it will be reduced to fit the size of the virtual matrix.
- *w*, *h*, *d* describe the width, height and depth of the render area. The default values are *-1*.
  A value of *-1* means that the whole width, height or depth of the given array will be copied to the virtual matrix (the complete array).

- *xSrc*, *ySrc*, *zSrc* describe the source area. The default values are *0*.
  DrawPixelArea[3D] can use and render the complete background matrix that was retrieved with *GetPixelArea [3D]* or *CaptureScreen*. Or it can only access and render a certain part of it.
- *filter* is a multiplicative color filter of the data type *color*. The default values are *{255, 255, 255}*.
  By defining a color, these color values will be passed through and all other colors will be filtered out.
- In order to draw the whole matrix it is possible to call:

```
color matrix[][];       //3D: color matrix[][][];
DrawPixelArea(matrix); //3D: DrawPixelArea3D(matrix);
```

- **Summary:** *DrawPixelArea[3D]* can access the background matrix that was created with *GetPixelArea[3D]* or *CaptureScreen*. With *DrawPixelArea[3D]* you can render the complete array or only parts of it onto your virtual matrix. The array can be drawn at the default or an individual position.
- (For the drawing operation, it is assumed that the array describes a rectangular area (or a box in 3D) in which every single line has the same number of columns.)

Example

DrawPixelArea

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 array 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, WHITE);
    }
```

2.2.1.7 'PixelTranspose'

Introduction

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

1. Creating the pixel transpose table which holds the information for each pixel that should 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.
4. Releasing the pixel transpose table.

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 necessarily required to use the second parameter because the *growsize* is set to *128* by default.

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 in the predefined table. This index count starts with 0 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 coordinates and the fourth and fifth parameter, *destX* and *destY*, set the destination coordinates.

```
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 made available again.

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 Gradient effect and insert the following source code into the Global Macro. The differences will be easily visible.

Clockwise Rotation

As a Global Macro, this macro rotates the output by 90° clockwise.

```
@scriptname="Output Rotation";
@author="inoage";
@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

As a Global Macro, this macro rotates the output by 90° counter-clockwise.

```
@scriptname="Output Rotation";
@author="inoage";
@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

As a Global Macro, this macro mirrors the output at the diagonal axis.

```
@scriptname="Output Rotation";
@author="inoage";
@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();
}
```

2.2.1.8 'SetPixel'

Functionality

SetPixel functions offer the possibility to change the color of pixels. You can either specify a certain color or use grayscale. The following examples use only some of the functions that are available.

Examples

SetPixel

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

This sample paints red pixels onto the complete matrix with different brightness values.

```
@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();
}
```

SetPixel3D

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

This sample paints golden pixels onto the complete matrix. Please use a 3D matrix.

```
@scriptname="SetPixel3D test, use with MAS script effect";
@author="";
@version="";
@description="";

int maxX,maxY,maxZ,x,y,z;

void InitEffect()
{
  maxX=GetMatrixWidth();
  maxY=GetMatrixHeight();
  maxZ=GetMatrixDepth();
}

void RenderEffect()
{
  x=random(0,maxX-1);
  y=random(0,maxY-1);
  z=random(0,maxZ-1);
  SetPixel3D(GOLD,x,y,z);
}

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="inoage";
@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();
}
```

SetPixelGrayscale (MAS Script)

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

```
@scriptname="sample of grayscale 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++)
```

```
    {
      SetPixelGrayscale(i,i); // line from top left to bottom right
      SetPixelGrayscale(X-i-1,i); // line from top right to bottom left
    }

    // to render the complete matrix in grayscale, the grayscale() command
    // offers higher performance:
    // Grayscale();
  }

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

SetPixelGrayscale (Macro)

To test this script, you can use the Global Macro.

But first, please select for example the SCE Color Scroll effect in Deck A or Deck B and display the effect on the output.

```
@scriptname="sample of grayscale for a 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++)
    {
      SetPixelGrayscale(i,i); // line from top left to bottom right
      SetPixelGrayscale(i,Y-i-1); // line from top right to bottom left
    }
  }
  else // height larger than width
  {
    for(int i=0;i<X;i++)
    {
      SetPixelGrayscale(i,i); // line from top left to bottom right
      SetPixelGrayscale(X-i-1,i); // line from top right to bottom left
    }
  }
```

```
    // to render the complete matrix in grayscale, the Grayscale() command
    //  offers higher performance
    // Grayscale();
  }
```

2.2.1.9 Draw Shapes

Overview

- MADRIX Script provides powerful functions to draw shapes onto the matrix.
- You can either use absolute pixel values or you can use relative vector values.
- These functions can be used for 2D as well as 3D.

Functions

- **`void DrawPixelShape(color col, int shape, int x, int y, int z, int w, int h, int d, int lineWidth, int drawMode, int lookAtType)`**

color col - Is of the data type structure and defines a color for the shape. »**Using Data Types**

int shape - Is of the data type integer and defines the type of shape. See **below** for a list of Shape Type constants.

int x - Is of the data type integer and defines the X-coordinate (left).

int y - Is of the data type integer and defines the Y-coordinate (top).

int z - Is of the data type integer and defines the Z-coordinate (front).

int w - Is of the data type integer and defines the width of the shape.

int h - Is of the data type integer and defines the height of the shape.

int d - Is of the data type integer and defines the depth of the shape.

int lineWidth - Is of the data type integer and defines the width of the shapes' border. This is not supported by all shapes. See **below** for a list of supported shapes and modes.

int drawMode - Is of the data type integer and defines the Draw Mode. See **below** for a list of Draw Mode constants.

int lookAtType - Is of the data type integer and defines the Look-At Type. See **below** for a list of Look-At Type Constants.

- **`void DrawVectorShape(color col, int shape, float x, float y, float z, float w, float h, float d, int lineWidth, int drawMode, int lookAtType)`**

color col - Is of the data type structure and defines a color for the shape. »**Using Data Types**

int shape - Is of the data type integer and defines the type of shape. See **below** for a list of Shape Type constants.

float x - Is of the data type float and defines the X-coordinate relative to the matrix size (left).

float y - Is of the data type float and defines the Y-coordinate relative to the matrix size (top).

float z - Is of the data type float and defines the Z-coordinate relative to the matrix size (front).

float w - Is of the data type float and defines the width of the shape relative to the matrix size.

float h - Is of the data type float and defines the height of the shape relative to the matrix size.

float d - Is of the data type float and defines the depth of the shape relative to the matrix size.

int lineWidth - Is of the data type integer and defines the width of the shapes' border. This is not supported by all shapes. See **below** for a list of supported shapes and modes.

int drawMode - Is of the data type integer and defines the Draw Mode. See **below** for a list of Draw Mode constants.

int lookAtType - Is of the data type integer and defines the Look-At Type. See **below** for a list of Look-At Type Constants.

- Valid values for relative float values range from 0.0 to 1.0.

Shape Type Constants

Constant	Description	Supported Draw Modes	lineWidth
int DRAWSHAPE_RECTANGLE	Renders a rectangle.	int DRAWMODE_LINE int DRAWMODE_FACE	Only supported with int DRAWMODE_LINE
int DRAWSHAPE_SQUARE	Renders a square.	int DRAWMODE_LINE int DRAWMODE_FACE	Only supported with int DRAWMODE_LINE
int DRAWSHAPE_CIRCLE	Renders a circle.	int DRAWMODE_LINE int DRAWMODE_FACE	Only supported with int DRAWMODE_LINE
int DRAWSHAPE_ELLIPSE	Renders a ellipse.	int DRAWMODE_LINE int DRAWMODE_FACE	Only supported with int DRAWMODE_LINE
int DRAWSHAPE_CROSS	Renders a cross.	int DRAWMODE_LINE	Not Supported
int DRAWSHAPE_STAR	Renders a star.	int DRAWMODE_LINE	Not Supported
int DRAWSHAPE_DIAMOND	Renders a diamond.	int DRAWMODE_LINE int DRAWMODE_FACE	Only supported with int DRAWMODE_LINE
int DRAWSHAPE_SPHERE	Renders a sphere.	int DRAWMODE_LINE int DRAWMODE_FACE int DRAWMODE_VOLUME	Only supported with int DRAWMODE_FACE
int DRAWSHAPE_OCTAHEDRON	Renders a octahedron.	int DRAWMODE_LINE int DRAWMODE_FACE int DRAWMODE_VOLUME	Only supported with int DRAWMODE_FACE
int DRAWSHAPE_BOX	Renders a box.	int DRAWMODE_LINE int DRAWMODE_FACE int DRAWMODE_VOLUME	Only supported with int DRAWMODE_FACE
int DRAWSHAPE_CUBE	Renders a cube.	int DRAWMODE_LINE int DRAWMODE_FACE int DRAWMODE_VOLUME	Only supported with int DRAWMODE_FACE

Draw Mode Constants

Constant	Description
int **DRAWMODE_LINE**	Renders the outline of the shape (2D and 3D: ***Outlined***).
int **DRAWMODE_FACE**	Renders the outer surfaces (2D: ***Filled*** , 3D : ***Unfilled***).
int **DRAWMODE_VOLUME**	Renders the complete inner and outer shape (3D: ***Filled***).

Look-At Type Constants

Constant	Description
int **LOOKAT_FRONT**	Applies no rotation and allows you to look directly at the shape (***Front***).
int **LOOKAT_BACK**	Represents a rotation of 180° around the Y-axis (***Back***).
int **LOOKAT_LEFT**	Represents a rotation of -90° around the Y-axis (***Left***).
int **LOOKAT_RIGHT**	Represents a rotation of 90° around the Y-axis (***Right***).
int **LOOKAT_TOP**	Represents a rotation of -90° around the X-axis (***Top***).
int **LOOKAT_BOTTOM**	Represents a rotation of 90° around the X-axis (***Bottom***).

2.2.1.10 Render Shapes

Overview

- MADRIX Script provides powerful functions to render shapes onto the matrix.
- These functions can be used for 2D as well as 3D.

Functions

- **`void RenderShape(struct color, int shapeType, float positionX, float positionY, float po`**
  `float sizeX, float sizeY, float sizeZ, struct shape, int unitType)`

struct color - Is of the data type structure and defines a color for the shape. »**Using Data Types**

int shapeType - Is of the data type integer and defines the type of shape. See **below** for a list of Shape Type constants.

float positionX - Is of the data type float and defines the X-coordinate (left). The valid range depends on the parameter *unitType*.

float positionY - Is of the data type float and defines the Y-coordinate (top). The valid range depends on the parameter *unitType*.

float positionZ - Is of the data type float and defines the Z-coordinate (front). The valid range depends on the parameter *unitType*.

float sizeX - Is of the data type float and defines the width of the shape. The valid range depends on the parameter *unitType*.

float sizeY - Is of the data type float and defines the width of the shape. The valid range depends on the parameter *unitType*.

float sizeZ - Is of the data type float and defines the width of the shape. The valid range depends on the parameter *unitType*.

struct shape - Is of the data type structure and defines various settings for the shape. This structure needs be initialized first. Initialize it with its default settings by using the function *GetDefaultShape()*. Define settings as required. See **below** for more information.

int unitType - Is of the data type integer and defines the usage of units. See **below** for a list of Unit Type constants.

Shape Type Constants

Constant	Description
int **SHAPE_TYPE_LINE**	Sets the Shape Type to ***Line***.
int **SHAPE_TYPE_CURVE**	Sets the Shape Type to ***Curve***.
int **SHAPE_TYPE_FILLED**	Sets the Shape Type to ***Filled***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED**	Sets the Shape Type to ***Rectangle Outlined***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Rectangle Outlined Implode***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Rectangle Outlined Explode***.

Constant	Description
int **SHAPE_TYPE_RECTANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Rectangle Outlined Pulse***.
int **SHAPE_TYPE_RECTANGLE_FILLED**	Sets the Shape Type to ***Rectangle Filled***.
int **SHAPE_TYPE_RECTANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Rectangle Filled Implode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Rectangle Filled Explode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_PULSE**	Sets the Shape Type to ***Rectangle Filled Pulse***.
int **SHAPE_TYPE_SQUARE_OUTLINED**	Sets the Shape Type to ***Square Outlined*** .
int **SHAPE_TYPE_SQUARE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Square Outlined Implode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Square Outlined Explode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_PULSE**	Sets the Shape Type to ***Square Outlined Pulse***.
int **SHAPE_TYPE_SQUARE_FILLED**	Sets the Shape Type to ***Square Filled***.
int **SHAPE_TYPE_SQUARE_FILLED_IMPLODE**	Sets the Shape Type to ***Square Filled Implode***.
int **SHAPE_TYPE_SQUARE_FILLED_EXPLODE**	Sets the Shape Type to ***Square Filled Explode***.
int **SHAPE_TYPE_SQUARE_FILLED_PULSE**	Sets the Shape Type to ***Square Filled Pulse***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED**	Sets the Shape Type to ***Ellipse Outlined***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Ellipse Outlined Implode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Ellipse Outlined Explode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_PULSE**	Sets the Shape Type to ***Ellipse Outlined Pulse***.
int **SHAPE_TYPE_ELLIPSE_FILLED**	Sets the Shape Type to ***Ellipse Filled***.
int **SHAPE_TYPE_ELLIPSE_FILLED_IMPLODE**	Sets the Shape Type to ***Ellipse Filled Implode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_EXPLODE**	Sets the Shape Type to ***Ellipse Filled Explode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_PULSE**	Sets the Shape Type to ***Ellipse Filled Pulse***.
int **SHAPE_TYPE_CIRCLE_OUTLINED**	Sets the Shape Type to ***Circle Outlined***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Circle Outlined Implode***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Circle Outlined Explode***.

Constant	Description
int **SHAPE_TYPE_CIRCLE_OUTLINED_PULSE**	Sets the Shape Type to ***Circle Outlined Pulse***.
int **SHAPE_TYPE_CIRCLE_FILLED**	Sets the Shape Type to ***Circle Filled***.
int **SHAPE_TYPE_CIRCLE_FILLED_IMPLODE**	Sets the Shape Type to ***Circle Filled Implode***.
int **SHAPE_TYPE_CIRCLE_FILLED_EXPLODE**	Sets the Shape Type to ***Circle Filled Explode***.
int **SHAPE_TYPE_CIRCLE_FILLED_PULSE**	Sets the Shape Type to ***Circle Filled Pulse*** .
int **SHAPE_TYPE_DIAMOND_OUTLINED**	Sets the Shape Type to ***Diamond Outlined***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_IMPLODE**	Sets the Shape Type to ***Diamond Outlined Implode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_EXPLODE**	Sets the Shape Type to ***Diamond Outlined Explode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_PULSE**	Sets the Shape Type to ***Diamond Outlined Pulse***.
int **SHAPE_TYPE_DIAMOND_FILLED**	Sets the Shape Type to ***Diamond Filled***.
int **SHAPE_TYPE_DIAMOND_FILLED_IMPLODE**	Sets the Shape Type to ***Diamond Filled Implode***.
int **SHAPE_TYPE_DIAMOND_FILLED_EXPLODE**	Sets the Shape Type to ***Diamond Filled Explode***.
int **SHAPE_TYPE_DIAMOND_FILLED_PULSE**	Sets the Shape Type to ***Diamond Filled Pulse***.
int **SHAPE_TYPE_PENTAGON_OUTLINED**	Sets the Shape Type to ***Pentagon Outlined***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_IMPLODE**	Sets the Shape Type to ***Pentagon Outlined Implode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_EXPLODE**	Sets the Shape Type to ***Pentagon Outlined Explode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_PULSE**	Sets the Shape Type to ***Pentagon Outlined Pulse***.
int **SHAPE_TYPE_PENTAGON_FILLED**	Sets the Shape Type to ***Pentagon Filled***.
int **SHAPE_TYPE_PENTAGON_FILLED_IMPLODE**	Sets the Shape Type to ***Pentagon Filled Implode***.
int **SHAPE_TYPE_PENTAGON_FILLED_EXPLODE**	Sets the Shape Type to ***Pentagon Filled Explode***.
int **SHAPE_TYPE_PENTAGON_FILLED_PULSE**	Sets the Shape Type to ***Pentagon Filled Pulse***.
int **SHAPE_TYPE_HEXAGON_OUTLINED**	Sets the Shape Type to ***Hexagon Outlined***.
int **SHAPE_TYPE_HEXAGON_OUTLINED_IMPLODE**	Sets the Shape Type to ***Hexagon Outlined Implode***.

Constant	Description
int **SHAPE_TYPE_HEXAGON_OUTLINED_EXPLODE**	Sets the Shape Type to ***Hexagon Outlined Explode***.
int **SHAPE_TYPE_HEXAGON_OUTLINED_PULSE**	Sets the Shape Type to ***Hexagon Outlined Pulse***.
int **SHAPE_TYPE_HEXAGON_FILLED**	Sets the Shape Type to ***Hexagon Filled***.
int **SHAPE_TYPE_HEXAGON_FILLED_IMPLODE**	Sets the Shape Type to ***Hexagon Filled Implode***.
int **SHAPE_TYPE_HEXAGON_FILLED_EXPLODE**	Sets the Shape Type to ***Hexagon Filled Explode***.
int **SHAPE_TYPE_HEXAGON_FILLED_PULSE**	Sets the Shape Type to ***Hexagon Filled Pulse***.
int **SHAPE_TYPE_HEART_OUTLINED**	Sets the Shape Type to ***Heart Outlined***.
int **SHAPE_TYPE_HEART_OUTLINED_IMPLODE**	Sets the Shape Type to ***Heart Outlined Implode***.
int **SHAPE_TYPE_HEART_OUTLINED_EXPLODE**	Sets the Shape Type to ***Heart Outlined Explode***.
int **SHAPE_TYPE_HEART_OUTLINED_PULSE**	Sets the Shape Type to ***Heart Outlined Pulse***.
int **SHAPE_TYPE_HEART_FILLED**	Sets the Shape Type to ***Heart Filled***.
int **SHAPE_TYPE_HEART_FILLED_IMPLODE**	Sets the Shape Type to ***Heart Filled Implode***.
int **SHAPE_TYPE_HEART_FILLED_EXPLODE**	Sets the Shape Type to ***Heart Filled Explode***.
int **SHAPE_TYPE_HEART_FILLED_PULSE**	Sets the Shape Type to ***Heart Filled Pulse***.
int **SHAPE_TYPE_CROSS_OUTLINED**	Sets the Shape Type to ***Cross Outlined***.
int **SHAPE_TYPE_CROSS_OUTLINED_IMPLODE**	Sets the Shape Type to ***Cross Outlined Implode***.
int **SHAPE_TYPE_CROSS_OUTLINED_EXPLODE**	Sets the Shape Type to ***Cross Outlined Explode***.
int **SHAPE_TYPE_CROSS_OUTLINED_PULSE**	Sets the Shape Type to ***Cross Outlined Pulse***.
int **SHAPE_TYPE_CROSS_FILLED**	Sets the Shape Type to ***Cross Filled***.
int **SHAPE_TYPE_CROSS_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Filled Implode***.
int **SHAPE_TYPE_CROSS_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Filled Explode***.
int **SHAPE_TYPE_CROSS_FILLED_PULSE**	Sets the Shape Type to ***Cross Filled Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED**	Sets the Shape Type to ***Cross Straight Outlined***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_IMPLODE**	Sets the Shape Type to ***Cross Straight Outlined Implode***.

Constant	Description
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_EXPLODE**	Sets the Shape Type to ***Cross Straight Outlined Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_PULSE**	Sets the Shape Type to ***Cross Straight Outlined Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED**	Sets the Shape Type to ***Cross Straight Filled***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Straight Filled Implode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Straight Filled Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_PULSE**	Sets the Shape Type to ***Cross Straight Filled Pulse***.
int **SHAPE_TYPE_STAR_OUTLINED**	Sets the Shape Type to ***Star Outlined***.
int **SHAPE_TYPE_STAR_OUTLINED_IMPLODE**	Sets the Shape Type to ***Star Outlined Implode***.
int **SHAPE_TYPE_STAR_OUTLINED_EXPLODE**	Sets the Shape Type to ***Star Outlined Explode***.
int **SHAPE_TYPE_STAR_OUTLINED_PULSE**	Sets the Shape Type to ***Star Outlined Pulse***.
int **SHAPE_TYPE_STAR_FILLED**	Sets the Shape Type to ***Star Filled***.
int **SHAPE_TYPE_STAR_FILLED_IMPLODE**	Sets the Shape Type to ***Star Filled Implode***.
int **SHAPE_TYPE_STAR_FILLED_EXPLODE**	Sets the Shape Type to ***Star Filled Explode***.
int **SHAPE_TYPE_STAR_FILLED_PULSE**	Sets the Shape Type to ***Star Filled Pulse***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED**	Sets the Shape Type to ***Triangle Outlined***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Triangle Outlined Implode***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Triangle Outlined Explode***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Triangle Outlined Pulse***.
int **SHAPE_TYPE_TRIANGLE_FILLED**	Sets the Shape Type to ***Triangle Filled***.
int **SHAPE_TYPE_TRIANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Triangle Filled Implode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Triangle Filled Explode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_PULSE**	Sets the Shape Type to ***Triangle Filled Pulse***.
int **SHAPE_TYPE_BOX_OUTLINED**	Sets the Shape Type to ***3D Box Outlined*** .

Constant	Description
int **SHAPE_TYPE_BOX_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Box Outlined Implode***.
int **SHAPE_TYPE_BOX_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Box Outlined Explode***.
int **SHAPE_TYPE_BOX_OUTLINED_PULSE**	Sets the Shape Type to ***3D Box Outlined Pulse***.
int **SHAPE_TYPE_BOX_UNFILLED**	Sets the Shape Type to ***3D Box Unfilled***.
int **SHAPE_TYPE_BOX_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Box Unfilled Implode***.
int **SHAPE_TYPE_BOX_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Box Unfilled Explode***.
int **SHAPE_TYPE_BOX_UNFILLED_PULSE**	Sets the Shape Type to ***3D Box Unfilled Pulse***.
int **SHAPE_TYPE_BOX_FILLED**	Sets the Shape Type to ***3D Box Filled***.
int **SHAPE_TYPE_BOX_FILLED_IMPLODE**	Sets the Shape Type to ***3D Box Filled Implode***.
int **SHAPE_TYPE_BOX_FILLED_EXPLODE**	Sets the Shape Type to ***3D Box Filled Explode***.
int **SHAPE_TYPE_BOX_FILLED_PULSE**	Sets the Shape Type to ***3D Box Filled Pulse***.
int **SHAPE_TYPE_CUBE_OUTLINED**	Sets the Shape Type to ***3D Cube Outlined***.
int **SHAPE_TYPE_CUBE_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Cube Outlined Implode***.
int **SHAPE_TYPE_CUBE_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Cube Outlined Explode***.
int **SHAPE_TYPE_CUBE_OUTLINED_PULSE**	Sets the Shape Type to ***3D Cube Outlined Pulse***.
int **SHAPE_TYPE_CUBE_UNFILLED**	Sets the Shape Type to ***3D Cube Unfilled*** .
int **SHAPE_TYPE_CUBE_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Unfilled Implode***.
int **SHAPE_TYPE_CUBE_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Unfilled Explode***.
int **SHAPE_TYPE_CUBE_UNFILLED_PULSE**	Sets the Shape Type to ***3D Cube Unfilled Pulse***.
int **SHAPE_TYPE_CUBE_FILLED**	Sets the Shape Type to ***3D Cube Filled***.
int **SHAPE_TYPE_CUBE_FILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Filled Implode***.
int **SHAPE_TYPE_CUBE_FILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Filled Explode***.
int **SHAPE_TYPE_CUBE_FILLED_PULSE**	Sets the Shape Type to ***3D Cube Filled Pulse***.

Constant	Description
int SHAPE_TYPE_SPHERE_UNFILLED	Sets the Shape Type to ***3D Sphere Unfilled***.
int SHAPE_TYPE_SPHERE_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Sphere Unfilled Implode***.
int SHAPE_TYPE_SPHERE_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Sphere Unfilled Explode***.
int SHAPE_TYPE_SPHERE_UNFILLED_PULSE	Sets the Shape Type to ***3D Sphere Unfilled Pulse***.
int SHAPE_TYPE_SPHERE_FILLED	Sets the Shape Type to ***3D Sphere Filled***.
int SHAPE_TYPE_SPHERE_FILLED_IMPLODE	Sets the Shape Type to ***3D Sphere Filled Implode***.
int SHAPE_TYPE_SPHERE_FILLED_EXPLODE	Sets the Shape Type to ***3D Sphere Filled Explode***.
int SHAPE_TYPE_SPHERE_FILLED_PULSE	Sets the Shape Type to ***3D Sphere Filled Pulse***.
int SHAPE_TYPE_ELLIPSOID_UNFILLED	Sets the Shape Type to ***3D Ellipsoid Unfilled***.
int SHAPE_TYPE_ELLIPSOID_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Ellipsoid Unfilled Implode***.
int SHAPE_TYPE_ELLIPSOID_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Ellipsoid Unfilled Explode***.
int SHAPE_TYPE_ELLIPSOID_UNFILLED_PULSE	Sets the Shape Type to ***3D Ellipsoid Unfilled Pulse***.
int SHAPE_TYPE_ELLIPSOID_FILLED	Sets the Shape Type to ***3D Ellipsoid Filled***.
int SHAPE_TYPE_ELLIPSOID_FILLED_IMPLODE	Sets the Shape Type to ***3D Ellipsoid Filled Implode***.
int SHAPE_TYPE_ELLIPSOID_FILLED_EXPLODE	Sets the Shape Type to ***3D Ellipsoid Filled Explode***.
int SHAPE_TYPE_ELLIPSOID_FILLED_PULSE	Sets the Shape Type to ***3D Ellipsoid Filled Pulse***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED	Sets the Shape Type to ***3D Octahedron Unfilled***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Implode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Explode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_PULSE	Sets the Shape Type to ***3D Octahedron Unfilled Pulse***.
int SHAPE_TYPE_OCTAHEDRON_FILLED	Sets the Shape Type to ***3D Octahedron Filled***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Filled Implode***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Filled Explode***.

Constant	Description
int **SHAPE_TYPE_OCTAHEDRON_FILLED_PULSE**	Sets the Shape Type to ***3D Octahedron Filled Pulse***.
int **SHAPE_TYPE_3D_HEART_UNFILLED**	Sets the Shape Type to ***3D Heart Unfilled***.
int **SHAPE_TYPE_3D_HEART_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Heart Unfilled Implode***.
int **SHAPE_TYPE_3D_HEART_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Heart Unfilled Explode***.
int **SHAPE_TYPE_3D_HEART_UNFILLED_PULSE**	Sets the Shape Type to ***3D Heart Unfilled Pulse***.
int **SHAPE_TYPE_3D_HEART_FILLED**	Sets the Shape Type to ***3D Heart Filled***.
int **SHAPE_TYPE_3D_HEART_FILLED_IMPLODE**	Sets the Shape Type to ***3D Heart Filled Implode***.
int **SHAPE_TYPE_3D_HEART_FILLED_EXPLODE**	Sets the Shape Type to ***3D Heart Filled Explode***.
int **SHAPE_TYPE_3D_HEART_FILLED_PULSE**	Sets the Shape Type to ***3D Heart Filled Pulse***.
int **SHAPE_TYPE_3D_STAR_UNFILLED**	Sets the Shape Type to ***3D Star Unfilled***.
int **SHAPE_TYPE_3D_STAR_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Star Unfilled Implode***.
int **SHAPE_TYPE_3D_STAR_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Star Unfilled Explode***.
int **SHAPE_TYPE_3D_STAR_UNFILLED_PULSE**	Sets the Shape Type to ***3D Star Unfilled Pulse***.
int **SHAPE_TYPE_3D_STAR_FILLED**	Sets the Shape Type to ***3D Star Filled***.
int **SHAPE_TYPE_3D_STAR_FILLED_IMPLODE**	Sets the Shape Type to ***3D Star Filled Implode***.
int **SHAPE_TYPE_3D_STAR_FILLED_EXPLODE**	Sets the Shape Type to ***3D Star Filled Explode***.
int **SHAPE_TYPE_3D_STAR_FILLED_PULSE**	Sets the Shape Type to ***3D Star Filled Pulse***.
int **SHAPE_TYPE_3D_CROSS_UNFILLED**	Sets the Shape Type to ***3D Cross Unfilled***.
int **SHAPE_TYPE_3D_CROSS_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Cross Unfilled Implode***.
int **SHAPE_TYPE_3D_CROSS_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Cross Unfilled Explode***.
int **SHAPE_TYPE_3D_CROSS_UNFILLED_PULSE**	Sets the Shape Type to ***3D Cross Unfilled Pulse***.
int **SHAPE_TYPE_3D_CROSS_FILLED**	Sets the Shape Type to ***3D Cross Filled***.
int **SHAPE_TYPE_3D_CROSS_FILLED_IMPLODE**	Sets the Shape Type to ***3D Cross Filled Implode***.

Constant	Description
int SHAPE_TYPE_3D_CROSS_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Filled Explode***.
int SHAPE_TYPE_3D_CROSS_FILLED_PULSE	Sets the Shape Type to ***3D Cross Filled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED	Sets the Shape Type to ***3D Cross Straight Unfilled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Unfilled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED	Sets the Shape Type to ***3D Cross Straight Filled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Filled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Filled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Filled Pulse***.
int SHAPE_TYPE_PYRAMID_UNFILLED	Sets the Shape Type to ***3D Pyramid Unfilled***.
int SHAPE_TYPE_PYRAMID_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Implode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Explode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_PULSE	Sets the Shape Type to ***3D Pyramid Unfilled Pulse***.
int SHAPE_TYPE_PYRAMID_FILLED	Sets the Shape Type to ***3D Pyramid Filled***.
int SHAPE_TYPE_PYRAMID_FILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Filled Implode***.
int SHAPE_TYPE_PYRAMID_FILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Filled Explode***.
int SHAPE_TYPE_PYRAMID_FILLED_PULSE	Sets the Shape Type to ***3D Pyramid Filled Pulse***.
int SHAPE_TYPE_TEXT	Sets the Shape Type to ***Text***.
int SHAPE_TYPE_WAVE_LINEAR	Sets the Shape Type to ***Wave Linear***.
int SHAPE_TYPE_WAVE_RADAR	Sets the Shape Type to ***Wave Radar***.
int SHAPE_TYPE_WAVE_HELIX	Sets the Shape Type to ***Wave Helix***.
int SHAPE_TYPE_WAVE_CIRCLE	Sets the Shape Type to ***Wave Circle***.
int SHAPE_TYPE_WAVE_SQUARE	Sets the Shape Type to ***Wave Square***.

Constant	Description
int **SHAPE_TYPE_WAVE_DIAMOND**	Sets the Shape Type to ***Wave Diamond***.
int **SHAPE_TYPE_WAVE_SPHERE**	Sets the Shape Type to ***Wave Sphere***.
int **SHAPE_TYPE_WAVE_CUBE**	Sets the Shape Type to ***Wave Cube***.
int **SHAPE_TYPE_WAVE_OCTAHEDRON**	Sets the Shape Type to ***Wave Octahedron***.
int **SHAPE_TYPE_RANDOM**	Selects an available Shape Type for each shape randomly. Sets the Shape Type to ***Random***.
int **SHAPE_TYPE_RANDOM_STATIC**	Sets the Shape Type to ***Random Static***.
int **SHAPE_TYPE_RANDOM_IMPLODE**	Sets the Shape Type to ***Random Implode***.
int **SHAPE_TYPE_RANDOM_EXPLODE**	Sets the Shape Type to ***Random Explode***.
int **SHAPE_TYPE_RANDOM_PULSE**	Sets the Shape Type to ***Random Pulse***.
int **SHAPE_TYPE_RANDOM_OUTLINED**	Sets the Shape Type to ***Random Outlined***.
int **SHAPE_TYPE_RANDOM_OUTLINED_STATIC**	Sets the Shape Type to ***Random Outlined Static***.
int **SHAPE_TYPE_RANDOM_OUTLINED_IMPLODE**	Sets the Shape Type to ***Random Outlined Implode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_EXPLODE**	Sets the Shape Type to ***Random Outlined Explode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_PULSE**	Sets the Shape Type to ***Random Outlined Pulse***.
int **SHAPE_TYPE_RANDOM_UNFILLED**	Sets the Shape Type to ***Random Unfilled*** .
int **SHAPE_TYPE_RANDOM_UNFILLED_STATIC**	Sets the Shape Type to ***Random Unfilled Static***.
int **SHAPE_TYPE_RANDOM_UNFILLED_IMPLODE**	Sets the Shape Type to ***Random Unfilled Implode***.
int **SHAPE_TYPE_RANDOM_UNFILLED_EXPLODE**	Sets the Shape Type to ***Random Unfilled Explode***.
int **SHAPE_TYPE_RANDOM_UNFILLED_PULSE**	Sets the Shape Type to ***Random Unfilled Pulse***.
int **SHAPE_TYPE_RANDOM_FILLED**	Sets the Shape Type to ***Random Filled***.
int **SHAPE_TYPE_RANDOM_FILLED_STATIC**	Sets the Shape Type to ***Random Filled Static***.
int **SHAPE_TYPE_RANDOM_FILLED_IMPLODE**	Sets the Shape Type to ***Random Filled Implode***.
int **SHAPE_TYPE_RANDOM_FILLED_EXPLODE**	Sets the Shape Type to ***Random Filled Explode***.
int **SHAPE_TYPE_RANDOM_FILLED_PULSE**	Sets the Shape Type to ***Random Filled Pulse***.

Constant	Description
int **SHAPE_TYPE_2D_RANDOM**	Sets the Shape Type to ***2D Random***.
int **SHAPE_TYPE_2D_RANDOM_STATIC**	Sets the Shape Type to ***2D Random Static***.
int **SHAPE_TYPE_2D_RANDOM_IMPLODE**	Sets the Shape Type to ***2D Random Implode***.
int **SHAPE_TYPE_2D_RANDOM_EXPLODE**	Sets the Shape Type to ***2D Random Explode***.
int **SHAPE_TYPE_2D_RANDOM_PULSE**	Sets the Shape Type to ***2D Random Pulse***.
int **SHAPE_TYPE_2D_RANDOM_OUTLINED**	Sets the Shape Type to ***2D Random Outlined***.
int **SHAPE_TYPE_2D_RANDOM_OUTLINED_STATIC**	Sets the Shape Type to ***2D Random Outlined Static***.
int **SHAPE_TYPE_2D_RANDOM_OUTLINED_IMPLODE**	Sets the Shape Type to ***2D Random Outlined Implode***.
int **SHAPE_TYPE_2D_RANDOM_OUTLINED_EXPLODE**	Sets the Shape Type to ***2D Random Outlined Explode***.
int **SHAPE_TYPE_2D_RANDOM_OUTLINED_PULSE**	Sets the Shape Type to ***2D Random Outlined Pulse***.
int **SHAPE_TYPE_2D_RANDOM_FILLED**	Sets the Shape Type to ***2D Random Filled***.
int **SHAPE_TYPE_2D_RANDOM_FILLED_STATIC**	Sets the Shape Type to ***2D Random Filled Static***.
int **SHAPE_TYPE_2D_RANDOM_FILLED_IMPLODE**	Sets the Shape Type to ***2D Random Filled Implode***.
int **SHAPE_TYPE_2D_RANDOM_FILLED_EXPLODE**	Sets the Shape Type to ***2D Random Filled Explode***.
int **SHAPE_TYPE_2D_RANDOM_FILLED_PULSE**	Sets the Shape Type to ***2D Random Filled Pulse***.
int **SHAPE_TYPE_3D_RANDOM**	Sets the Shape Type to ***3D Random***.
int **SHAPE_TYPE_3D_RANDOM_STATIC**	Sets the Shape Type to ***3D Random Static***.
int **SHAPE_TYPE_3D_RANDOM_IMPLODE**	Sets the Shape Type to ***3D Random Implode***.
int **SHAPE_TYPE_3D_RANDOM_EXPLODE**	Sets the Shape Type to ***3D Random Explode***.
int **SHAPE_TYPE_3D_RANDOM_PULSE**	Sets the Shape Type to ***3D Random Pulse***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED**	Sets the Shape Type to ***3D Random Unfilled***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_STATIC**	Sets the Shape Type to ***3D Random Unfilled Static***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Random Unfilled Implode***.

Constant	Description
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Random Unfilled Explode***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_PULSE**	Sets the Shape Type to ***3D Random Unfilled Pulse***.
int **SHAPE_TYPE_3D_RANDOM_FILLED**	Sets the Shape Type to ***3D Random Filled***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_STATIC**	Sets the Shape Type to ***3D Random Filled Static***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_IMPLODE**	Sets the Shape Type to ***3D Random Filled Implode***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_EXPLODE**	Sets the Shape Type to ***3D Random Filled Explode***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_PULSE**	Sets the Shape Type to ***3D Random Filled Pulse***.

Shape Structure

shape	▪ int renderingMode ▪ int shapeAlignment ▪ int shapeRotation ▪ int blendingMode ▪ int originType ▪ float border ▪ float innerGlow ▪ float outerGlow ▪ int innerGlowInterpolation ▪ int outerGlowInterpolation ▪ float proportion ▪ float diagonalLength	*shape* stores specific information for shapes. Valid values for *renderingMode* are: **RENDERING_MODE_EXTENDED** **RENDERING_MODE_SIMPLE** Valid values for *shapeAlignment* are: **LOOKAT_FRONT** **LOOKAT_BACK** **LOOKAT_LEFT** **LOOKAT_RIGHT** **LOOKAT_TOP** **LOOKAT_BOTTOM** **LOOKAT_RANDOM** Valid values for *shapeRotation* are: **ROTATION_CCW_0** **ROTATION_CCW_90** **ROTATION_CCW_180** **ROTATION_CCW_270** **ROTATION_CW_0** **ROTATION_CW_90** **ROTATION_CW_180** **ROTATION_CW_270** *blendingMode* is only available for RENDERING_MODE_EXTENDED. Valid values are: **BLENDING_MODE_NONE** **BLENDING_MODE_ALPHA** Valid values for *originType* are: **ORIGIN_CENTER** **ORIGIN_GEOMETRIC_CENTER** **ORIGIN_FRONT** **ORIGIN_BACK** **ORIGIN_LEFT** **ORIGIN_RIGHT** **ORIGIN_TOP** **ORIGIN_BOTTOM** **ORIGIN_TOP_LEFT** **ORIGIN_TOP_RIGHT** **ORIGIN_BOTTOM_LEFT** **ORIGIN_BOTTOM_RIGHT** **ORIGIN_FRONT_LEFT** **ORIGIN_FRONT_RIGHT** **ORIGIN_BACK_LEFT** **ORIGIN_BACK_RIGHT** **ORIGIN_FRONT_TOP** **ORIGIN_FRONT_BOTTOM** **ORIGIN_BACK_TOP** **ORIGIN_BACK_BOTTOM** **ORIGIN_FRONT_TOP_LEFT** **ORIGIN_FRONT_TOP_RIGHT** **ORIGIN_FRONT_BOTTOM_LEFT** **ORIGIN_FRONT_BOTTOM_RIGH** **ORIGIN_BACK_TOP_LEFT**

		ORIGIN_BACK_TOP_RIGHT **ORIGIN_BACK_BOTTOM_LEFT** **ORIGIN_BACK_BOTTOM_RIGHT** *border* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *innerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *outerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. Valid values for *innerGlowInterpolation* are: **INTERPOLATION_TYPE_LINEAR** **INTERPOLATION_TYPE_EASE_BOUNCE_IN** **INTERPOLATION_TYPE_EASE_BOUNCE_OUT** **INTERPOLATION_TYPE_EASE_BOUNCE_INOUT** **INTERPOLATION_TYPE_EASE_CIRC_IN** **INTERPOLATION_TYPE_EASE_CIRC_OUT** **INTERPOLATION_TYPE_EASE_CIRC_INOUT** **INTERPOLATION_TYPE_EASE_CUBIC_IN** **INTERPOLATION_TYPE_EASE_CUBIC_OUT** **INTERPOLATION_TYPE_EASE_CUBIC_INOUT** **INTERPOLATION_TYPE_EASE_SINE_IN** **INTERPOLATION_TYPE_EASE_SINE_OUT** **INTERPOLATION_TYPE_EASE_SINE_INOUT** **INTERPOLATION_TYPE_EASE_EXPO_IN** **INTERPOLATION_TYPE_EASE_EXPO_OUT** **INTERPOLATION_TYPE_EASE_EXPO_INOUT** Valid values for *outerGlowInterpolation* are: See *innerGlowInterpolation* *proportion* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_CROSS_STRAIGHT, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, SHAPE_TYPE_3D_CROSS_STRAIGHT, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*. *diagonalLength* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*.

Unit Type Constants

Constant	Description
int **UNIT_TYPE_NORM**	Uses normalized values ranging from 0.0 to 1.0. It is the default parameter.
int **UNIT_TYPE_VOXEL**	Uses voxel units based on the Matrix Size.
int **UNIT_TYPE_PERCENT**	Uses percentage values from 0 to 100.

Example

```
@scriptname="RenderShape";
@author="";
@version="MADRIX 3.4";
@description="Example";

color col = WHITE;
shape shapeProps = GetDefaultShape();
float pos_x = 0.5;
float pos_y = 0.5;
float pos_z = 0.5;
float size_x = 0.7;
float size_y = 0.7;
float size_z = 1.0;
int shapeType = SHAPE_TYPE_CIRCLE_OUTLINED;
int unitType = UNIT_TYPE_NORM;

void InitEffect()
{
    shapeProps.renderingMode = RENDERING_MODE_EXTENDED;
    shapeProps.shapeAlignment = LOOKAT_FRONT ;
    shapeProps.shapeRotation = ROTATION_CW_90;
    shapeProps.blendingMode = BLENDING_MODE_ALPHA;
    shapeProps.originType = ORIGIN_GEOMETRIC_CENTER;
    shapeProps.border = 0.01;
    shapeProps.innerGlow = 0.5;
    shapeProps.outerGlow = 0.2;
    shapeProps.innerGlowInterpolation = INTERPOLATION_TYPE_LINEAR;
    shapeProps.outerGlowInterpolation = INTERPOLATION_TYPE_LINEAR;
  //shapeProps.proportion;
  //shapeProps.diagonalLength;

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{
    RenderShape(col, shapeType, pos_x, pos_y, pos_z, size_x, size_y, size_z, shapeProps, unitType);
    RenderShape(col, shapeType, pos_x - 0.25, pos_y - 0.25, pos_z - 0.25, size_x, size_y, size_z, sh
}
```

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

2.2.2 Sound2Light And Music2Light

MADRIX can analyze audio input for Sound2Light (S2L) or Music2Light (M2L) effects. This data is also available in MADRIX Script and may be used to create even more advanced effects, controlled by music.

It is necessary to differentiate between sound and music data.

- Sound data refers to data based on the frequency of the given input signal. A common effect based on sound data is the equalizer (audio spectrum). The volume is also this kind of sound data.
- Music data refers to information known from actual music theory. Therefore, MADRIX identifies musical parameters, such as tonality, bass, intervals or the current tone (or chord) itself. Using the tone and the tonality you could see that C Major or d minor is currently played, for example. There are numerous examples.

In order to receive sound and music data in MADRIX Script, the audio input has to be activated, which is true by default. To check or change your configuration, please navigate to *Preferences > Device Manager... > Audio* via the MADRIX main menu.

2.2.2.1 Sound2Light (S2L)

Functionality

- As described above, sound data refers mainly to frequency-based data. First of all, the function *GetSoundLevel* retrieves the volume of the audio channels (left and right). The *float* value returned ranges from *0.0* to *1.0*, the lowest and highest level possible, respectively.
- Moreover, other available data are the frequency values. They are stored in the arrays **SOUND_DATA_LEFT** and **SOUND_DATA_RIGHT,** which are both of the data type *float*. There are up to *511* values and each describes the volume of a »**well-defined frequency**.
- The *length*-operator of the »**arrays** tells how much valid data they contain. A check may be necessary and then just take the values that are actually provided to have a proper effect.
- In contrast to other dynamic arrays, in MADRIX Script those arrays will not grow because their size is fixed. Trying to get an invalid element, always results in *0*.

Example

The following example for the MAS Script Effect analyses the left and the right audio channel. Lines will be drawn onto the matrix to indicate the average frequency.

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

void InitEffect()
{
}

float avgFrequ(float array[])
{
    float result;

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

        result /= (float)array.length;
    }

    return(result);
}

void RenderEffect()
{
    int valL = (int)(GetSoundLevel(0)*255.0);
```

```
    int valR = (int)(GetSoundLevel(1)*255.0);
    color c = {valL, valR, (valL * valR) % 255, (valR + valL) / 2, 0};

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

    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);
}
```

2.2.2.2 Music2Light (M2L)

Introduction

Music2Light effects can go a step further and 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 And Scale

Let us start with tonality and scale of a chord. The two can be retrieved via the functions *GetTonality* and *GetToneScale*, respectively. The exemplary sample source code for the MAS Script Effect 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}    //B
                                 };
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 array index and possibly 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 a C with 8.25 Hz. The highest note is a G with 12.6 kHz. There are two functions available to get information about the identified notes. *GetNoteValue* retrieves the volume of the given note and *IsNote* returns 1 (**TRUE**) if the given note has been detected, otherwise 0 (**FALSE**). There is the function *GetAllNoteValues*, which fills an array with the volume of each note. Using this method for a lot of notes is much faster then calling *GetNoteValue* each time. An overview of which index corresponds to which note is given by the »**Table Of Notes**

The next example for the MAS Script Effect 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 1056KHz
    val[1] = (float)GetNoteValue(84) / 127.0;
```

```
    //C with 2112KHz
    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 from *0.0* to *1.0*. Hereby, only three C notes are used.
- Then, a color is initialized and the matrix is filled.

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 1 (**TRUE**) if the specified interval could be analyzed, otherwise 0 (**FALSE**). Again, the function *GetAllIntervals* fills an array rapidly, each element with either 1 (**TRUE**) (interval was analyzed) or 0 (**FALSE**) (interval was not analyzed). The following short example for the MAS Script Effect 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 occurrences. 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 the middle to the bottom (if not analyzed).

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 in the » **List Of Functions**

MADRIX LIGHTING CONTROL

//PART C

MADRIX Script (Programming Language Overview)

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 Ouput Macro, and the Storage Place Macro (all listed below), the following table lists additional functions. The "+" symbol indicates, in which areas of MADRIX Script the functions can be used.

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Global 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. *srcX* and *srcY* are the coordinates in X and Y of the source. *destX* and *destY* are the destination coordinates. »Description	+	+	+	+

void **AddPixelTransposeEntry3D** (int srcX, int srcY, int srcZ, int destX, int destY, int destZ)	Adds one entry to the pixel transpose table and resizes the table if necessary. *srcX*, *srcY*, and srcZ are the coordinates in X, Y, and Z of the source. *destX*, *destY*, and *destZ* are the coordinates of the destination. »**Description**	+	+	+	+
float **arccos**(float a) float **arccosDeg**(float a)	Returns the arc cosine of the angle *a* in radian measure or degrees, respectively.	+	+	+	+
float **arccot**(float a) float **arccotDeg**(float a)	Returns the arc cotangent 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 **arctan**(float a) float **arctanDeg**(float a)	Returns the arc tangent of the angle *a* in radian measure or degrees, respectively	+	+	+	+
float **arctan2**(float x, float y)	Returns the angle of polar coordinates from two Cartesian coordinates *x* and *y*.	+	+	+	+
void **CaptureScreen**(color matrix [][], int xSrc, int ySrc, int w, int h)	Retrieves data from the screen and stores it into a 2-dimensional array of colors. »**Description**	+	+	+	+
float **ceil**(float f)	Rounds up the given value to the next integer value. E.g. ceil(2.00001) = 3.0	+	+	+	+
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 *1* if the version is equal or higher to the version specified with *major* and *minor*. Or else *0* is returned. The current Script Engine Version is 3.12. 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 *major*, *minor*, *subminor*, and *subsubminor*. Or else *0* is returned. The current MADRIX version is 5.3.2.30. 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 *alpha*.	+	+	+	+
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 *oldCol* with a new one (*newCol*).	+	+	+	+
float **cotan**(float a) float **cotanDeg**(float a)	Returns the cotangent 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 **cosH**(float a) float **cosHDeg**(float a)	Returns the hyperbolic cosine of the angle *a* in radian measure or degrees, respectively.	+	+	+	+
void **CreatePixelTransposeTable** (int size, int growsize)	Creates the pixel transpose table with the given *size* and *growsize*. »**Description**	+	+	+	+

float **deg2rad**(float a)	Converts the angle *a* from degrees to radian measure.	+	+	+	+
void **Dim**(float value)	Reduces the brightness of the complete virtual matrix. Valid values for *value* range from 0.0 to 1.0.	+	+	+	+
void **DimPixel**(float value, int x, int y)	Reduces the brightness of an individual pixel. *x* and *y* are the coordinates of the pixel. Valid values for *value* range from 0.0 to 1.0.	+	+	+	+
void **DimPixel3D**(float value, int x, int y, int z)	Reduces the brightness of an individual voxel. *x*, *y*, and z are the coordinates of the voxel. Valid values for *value* 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. *x* and *y* are the coordinates of the area (upper left corner). *width* and *height* specify the width and height of the area. Valid values for *value* range from 0.0 to 1.0.	+	+	+	+
void **DimPixelArea3D**(float value, int x, int y, int z, int width, int height, int depth)	Reduces the brightness of a certain area of the virtual 3D matrix. *x*, *y*, and z are the coordinates of the area (front upper left corner). *width*, *height*, and *depth* specify the width, height, and depth of the area. Valid values for *value* 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 array of colors and renders it on the matrix. »**Description**	+	+	+	+
void **DrawPixelArea3D**(color matrix[][][], int xDst, int yDst, int zDst, int w, int h, int d, int xSrc, int ySrc, int zSrc, color filter)	Copies data from a 3-dimensional array of colors and renders it on the matrix. »**Description**	+	+	+	+
void **DrawPixelBitmap**(string file, int x, int y)	Loads the specified image file and renders it on the matrix starting at position *(x, y)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawPixelBitmap3D**(string file, int x, int y, int z)	Loads the specified image file and renders it on the matrix starting at position *(x, y, z)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawPixelLine**(color col, int x1, int y1, int x2, int y2)	Draws a line from pixel (*x1, y1*) to pixel (*x2, y2*) with the specified color *col*.	+	+	+	+
void **DrawPixelLine3D**(color col, int x1, int y1, int z1, int x2, int y2, int z2)	Draws a line from voxel (*x1, y1, z1*) to voxel (*x2, y2, z2*) with the specified color *col*.	+	+	+	+
void **DrawPixelShape**(color col, int shape, int x, int y, int z, int w, int h, int d, int lineWidth, int drawMode, int lookAtType)	Draws a given *shape* onto the matrix with the given size (width *w*, height *h*, depth *d*) at the given position (*x*, *y*, *z*) with the specified color *col*. Learn more »**Draw Shapes**	+	+	+	+

void **DrawPixelText**(color c, font f, string t, int x, int y, int rotation)	Draws a text across the main output. »*color c* and »*font f* are structures. Valid values for rotation are **ROTATION_CCW_0**, **ROTATION_CCW_90**, **ROTATION_CCW_180**, **ROTATION_CCW_270** for counter-clockwise rotation and **ROTATION_CW_0**, **ROTATION_CW_90**, **ROTATION_CW_180**, **ROTATION_CW_270** for clockwise rotation. »Example 1 » Example 2 »Example 3 »Example 4	+	+	+	+
void **DrawVectorBitmap**(string file, int x, int y)	Loads the specified image file and renders it on the matrix starting at the relative position *(x, y)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawVectorBitmap3D** (string file, int x, int y, int z)	Loads the specified image file and renders it on the matrix starting at the relative position *(x, y, z)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawVectorLine**(color col, float x1, float y1, float x2, float y2)	Draws a line from the relative position (*x1, y1*) to the relative position (*x2, y2*) with the specified color.	+	+	+	+
void **DrawVectorLine3D**(color col, float x1, float y1, float z1, float x2, float y2, float z2)	Draws a line from the relative position (*x1, y1, z1*) to the relative position (*x2, y2, z2*) with the specified color *col*.	+	+	+	+
void **DrawVectorShape**(color col, int shape, float x, float y, float z, float w, float h, float d, int lineWidth, int drawMode, int lookAtType)	Draws a given *shape* at the relative position (*x, y, z*) with the relative size (width *w*, height *h*, depth *d*) and the specified color *col*. Learn more »Draw Shapes	+	+	+	+
void **DrawVectorText**(color c, font f, string t, float x, float y, int rotation)	Draws a vector text across the main output. »*color c* and »*font f* are structures. Valid values for rotation are **ROTATION_CCW_0**, **ROTATION_CCW_90**, **ROTATION_CCW_180**, **ROTATION_CCW_270** for counter-clockwise rotation and **ROTATION_CW_0**, **ROTATION_CW_90**, **ROTATION_CW_180**, **ROTATION_CW_270** for clockwise rotation. »Example 1 » Example 2	+	+	+	+
int **endswith**(string text, string substring)	This function checks if the string *text* ends with the given *substring*. If *text* ends with the specified *substring*, 1 (**TRUE**) is returned, otherwise 0 (**FALSE**). »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 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 *x*.	+	+	+	+

void **Filter**(int filter)	Renders a filter over the matrix. »**Valid values (Filters)** » **Description**	+	+	+	+
void **FilterColor**(color c)	Renders a color filter over the matrix. The values of the specified color will be passed through and all other colors will be filtered out.	+	+	+	+
int **findstring**(int startIndex, string text, string substring)	Searches for the *substring* within *text* starting at *startIndex*. The function returns an index that describes the position at which the *substring* begins. »**Description** Example: findstring(0, "`Hello World!`", "World") returns 6. If the *substring* is not found within *text*, *-1* is returned.	+	+	+	+
float **fmax**(float x, float y)	Returns the maximum value of the floating point numbers *x* and *y*.	+	+	+	+
float **fmin**(float x, float y)	Returns the minimum value of *x* and *y*.	+	+	+	+
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 array *buf* with the occurrences of each interval (buf[0] ... buf[10]). *buf[index]* is 1 (**TRUE**) if the specified interval was analyzed. »**Example**	+	+		
void **GetAllNoteValues**(int buf[])	Fills the array *buf* with the sound level values for each note (buf[0] ... buf[127]). *buf[index]* can differ from 0 to 127.	+	+		
string **GetApplicationPath**()	Locates the MADRIX.exe on your harddisk and returns the path as a *string*. »**Example**	+	+	+	+
int **GetAsync**()	Returns if asynchronous rendering is used. »**Description** This function is only available for some effects.	+	+		
int **GetBlind**()	Returns if ***Blind Mode*** is used. »**Description**	+	+		
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 **GetDayOfWeek**(int day, int month, int year)	Returns the day of the week of a specified date. Returned values include *0 = Sunday*, *1 = Monday*, ..., *6 = Saturday*. *-1* is returned if the provided parameters are invalid. Valid values for *day* range from *1* to *31*. Valid values for *month* range from *1* to *12*. Valid values for *year* start with value *1900*.	+	+	+	+

int **GetDmxFaderChannel**(int index)	Returns on which DMX channel the specified fader sends data. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 511 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
int **GetDmxFaderCount**()	Returns how many faders are available in the DMX Fader Tool.	+	+	+	+
int **GetDmxFaderUniverse**(int value)	Returns on which DMX universe the specified fader sends data. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
int **GetDmxFaderValue**(int value)	Returns the DMX value the specified fader sends. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
void **GetDmxIn**(int DmxValues[], int startchannel, int channels, int universe)	Stores the DMX-IN status of several DMX channels in an array (*DmxValues[]*). To use the function, choose a DMX start channel and specify the number of channels that should be scanned. Valid values for *startchannel* range from *0* to *511*. Valid values for *channels* range from *0* to *511*. Please note that a maximum of 512 channels can be scanned. This means that the sum of *startchannel* + *channels* must be less or equal to 512. In addition, specify the universe that should be scanned. Valid values for *universe* range from *0* to *255*. The default *universe* 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 (*channel*). Valid values for *channel* range from *0* to *511*. In addition, specify the universe that should be scanned. Valid values for *universe* range from *0* to *255*. The default value of *universe* is *0*, i.e. DMX Universe 1.	+	+	+	+
int **GetFilter**()	Returns which ***Filter (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 *SetFrameCount* for MAS Script. 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 **GetMatrixDepth**()	Returns the pixel count of the rendered matrix regarding the depth. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Matrix Size, since they can be mapped via Map Settings.	+	+	+	+

int **GetMatrixHeight**()	Returns the vertical pixel count of the rendered matrix. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Ma trix Size, since they can be mapped via Map Settings.	+	+	+	+
int **GetMatrixWidth**()	Returns the horizontal pixel count of the rendered matrix. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Matrix Size, since they can be mapped via Map Settings.	+	+	+	+
void **GetMidiInControl**(int midivalues[], int startcontrol, int controlcount, int midichannel, int device)	Stores the MIDI values for multiple controls in an array (*midivalues[]*). To use the function, choose a start control (*startcontrol*) and the number of controls (*controlcount*), as well as the MIDI channel (*midichannel*) and the *device*. The default value of *startnote*, *midichannel*, and *device* is *0*, i.e. the first control, channel, and device. The default and maximum value of *controlcount* 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 *control change (midicontrol)* and *midichannel* for the specified *device*. The default value of *control*, *midichannel*, and *device* 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 (*midivalues[]*). To use the function, choose a start note (*startnote*) and the number of notes (*notecount*), as well as the MIDI channel (*midichannel*) and the *device*. The default value of *startnote*, *midichannel*, and *device* is *0*, i.e. the first control, channel, and device. The default and maximum value of *notecount* 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 (*midinote*) and *midichannel* for the specified *device*. The default value of *note*, *midichannel*, and *device* 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 values (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*).	+	+	+	+
color **GetPixel3D**(int x, int y, int z)	Returns the color of the voxel at position (*x, y, z*).	+	+	+	+
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 array of colors. »**Description**	+	+	+	+

void **GetPixelArea3D**(color matrix [][][], int xSrc, int ySrc, int zSrc, int w, int h, int d, int xDst, int yDst, int zDst)	Retrieves data from the matrix and stores it into a 3-dimensional array of colors. »**Description**	+	+	+	+
string **GetScriptEngineVersion**()	Returns the script engine version and returns the value as a *string*. »**Example**	+	+	+	+
string **GetSoftwareVersion**()	Returns the MADRIX software version and returns the value as a *string*. »**Example**	+	+	+	+
int **GetSolo**()	Returns if ***Solo Mode*** is used. »**Description**	+	+		
float **GetSoundLevel**(int channel)	Returns a value, ranging from *0.0* to *1.0*, representing the sound level of the stereo channel: 0 = left channel, 1 = right channel. »**Example**	+	+		
int **GetStep**()	Returns if stepped rendering is used. »**Description** This function is only available for some effects.	+	+		
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.	+	+	+	+
time **GetTimeCode**()	Returns a time structure with the currently used ***Time Code***. »**Example**	+	+	+	+
time **GetTimeSunrise**(struct date, int latitude_degrees, int latitude_minutes, int longitude_degrees, int longitude_minutes, float time_zone_UTC_offset)	Returns the sunrise time of a geographic location at the specified *date*. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunriseCity**(struct date, int city)	Returns the sunrise time of a specified *city* at the specified *date*. Valid values for city are »**city constants**. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunset**(struct date, int latitude_degrees, int latitude_minutes, int longitude_degrees, int longitude_minutes, float time_zone_UTC_offset)	Returns the sunset time of a geographic location at the specified *date*. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunsetCity**(struct date, int city)	Returns the sunset time of a specified *city* at the specified *date*. Valid values for city are »**city constants**. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**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 *string*. »**Example**	+	+	+	+

color **GetVectorPixel**(float x,float y)	Returns the *color* of the pixel drawn at vector coordinates *x* and *y*.	+	+	+	+
color **GetVectorPixel3D**(float x, float y,float z)	Returns the *color* of the voxel drawn at vector coordinates *x*, *y*, and *z*.	+	+	+	+
void **Grayscale**()	Converts the whole matrix to grayscaled values.	+	+	+	+
string **hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are lower-cased (a-f).	+	+	+	+
string **Hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are upper-cased (A-F).	+	+	+	+
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 1 (**TRUE**) if the given string text contains only characters and figures and if its length is greater then 0. Otherwise, 0 (**FALSE**) is returned. »**Description**	+	+	+	+
int **isalpha**(string text)	Returns 1 (**TRUE**) if the given string contains only characters and if its length is greater than 0. Otherwise, 0 (**FALSE**) is returned. »**Description**	+	+	+	+
int **IsDmxFaderEnable**(int index)	Returns if the specified fader is enabled (*1*) or not (*0*).Valid values for *index* range from 0 to 11 (Indexing starts with 0).	+	+	+	+
int **IsDmxInEnabled**()	Retrieves if DMX-IN is activated (*1*) or not (*0*).	+	+	+	+
int **IsInterval**(int index)	Returns 1 (**TRUE**) if the specified interval (at the *index* position, ranging from *0* to *10*) was analyzed.	+	+		
int **IsMidiInEnabled**()	Retrieves if MIDI-IN is activated (*1*) or not (*0*).	+	+	+	+
int **IsNote**(int note)	Returns 1 (**TRUE**) if the specified note was analyzed. (0 = C = 8.25 Hz, ..., 127 = G = 12.67 kHz)	+	+		
int **isnum**(string text)	Returns 1 (**TRUE**) if the given text represents a number. This may be an integer value or a floating point number. Otherwise it returns 0 (**FALSE**). »**Description**	+	+	+	+
int **IsTonality**()	Returns 1 (**TRUE**) if the tonality could be determined.	+	+		
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*.	+	+	+	+
int **MapDlgGetAntiAliasing**()	Retrieves the ***Anti-Aliasing*** mode of the Layer. See »**here** for a list of Anti-Aliasing Mode constants.	+	+		
int **MapDlgGetMapMode**()	Retrieves the currently used ***Map Mirror Mode*** of the Layer. See »**here** for a list of constants.	+	+		

void **MapDlgGetMapPixel**(int map[])	Retrieves the map settings for the Layer using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgGetMapPixel3D**(int map[])	Retrieves the map settings for the Layer in 3D using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgGetMapVector**(float map[])	Retrieves the map settings for the Layer using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgGetMapVector3D** (float map[])	Retrieves the map settings for the Layer in 3D using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		

void **MapDlgGetRotationDegree** (int rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgGetRotationMode** (int mode[])	Retrieves the ***Rotation*** mode for axes *x, y, and z* of the Layer. See »here for a list of Rotation Mode constants. The values are saved in an array (*mode[]*). ▪ mode[0] = X-coordinate (X-Axis) ▪ mode[1] = Y-coordinate (Y-Axis) ▪ mode[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgGetRotationVector** (float rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values. The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
int **MapDlgGetRotationXDegree** ()	Retrieves the ***Rotation*** for the X-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationXMode**()	Retrieves the ***Rotation*** mode for the X-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationXVector**()	Retrieves the ***Rotation*** for the X-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationYDegree** ()	Retrieves the ***Rotation*** for the Y-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationYMode**()	Retrieves the ***Rotation*** mode for the Y-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationYVector**()	Retrieves the ***Rotation*** for the Y-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationZDegree** ()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationZMode**()	Retrieves the ***Rotation*** mode for the Z-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationZVector**()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
int **MapDlgGetTileMode**()	Retrieves the currently used ***Tile Mode*** of the Layer. See »here for a list of Tile Mode constants.	+	+		

void **MapDlgGetTileOffsetPixel** (int offset[])	Retrieves the ***Tiling Offset*** of the Layer using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgGetTileOffsetPixel3D**(int offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgGetTileOffsetVector** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgGetTileOffsetVector3D** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgGetTilePixel**(int tile [])	Retrieves the ***Tiling*** settings for the Layer using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		

void **MapDlgGetTilePixel3D**(int tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgGetTileVector**(float tile[])	Retrieves the ***Tiling*** settings for the Layer using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgGetTileVector3D** (float tile[])	Retrieves the tile settings for the Layer in 3D using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
int **MapDlgIsMapped**()	Retrieves if the Layer is mapped.	+	+		
void **MapDlgSetAntiAliasing**(int aaLevel)	Sets the ***Anti-Aliasing*** mode of the Layer. See »here for a list of Anti-Aliasing Mode constants.	+	+		
void **MapDlgSetMapMode**(int mode)	Sets the ***Map Mirror Mode*** for the Layer. See »here for a list of constants.	+	+		
void **MapDlgSetMapPixel**(int x, int y, int w, int h)	Maps the Layer to a certain area of the matrix using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgSetMapPixel3D**(int x, int y, int z, int w, int h, int d)	Maps the Layer to a certain area of the matrix in 3D using pixel values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgSetMapVector**(float x, float y, float w, float h)	Maps the Layer to a certain area of the matrix using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgSetMapVector3D** (float x, float y, float z, float w, float h, float d)	Maps the Layer to a certain area of the matrix in 3D using relative values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgSetRotationDegree** (int x, int y, int z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.	+	+		
void **MapDlgSetRotationMode** (int x, int y, int z)	Sets the ***Rotation*** mode for axes *x, y, and z* of the Layer. See »**here** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationVector** (float x, float y, float z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values.	+	+		
void **MapDlgSetRotationXDegree**(int value)	Sets the ***Rotation*** *value* for the X-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationXMode** (int mode)	Sets the ***Rotation*** mode for the X-axis of the Layer. See »**here** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationXVector** (float value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using degree values.	+	+		
void **MapDlgSetRotationYDegree**(int value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationYMode** (int mode)	Sets the ***Rotation*** mode for the Y-axis of the Layer. See »**here** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationYVector** (float value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationZDegree**(int value)	Sets the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationZMode** (int mode)	Sets the ***Rotation*** mode for the Z-axis of the Layer. See »**here** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationZVector** (float value)	Sets the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
void **MapDlgSetTileMode**(int mode)	Sets the ***Tile Mode*** of the Layer. See »**here** for a list of Tile Mode constants.	+	+		
void **MapDlgSetTileOffsetPixel** (int x, int y)	Sets the ***Tiling Offset*** of the Layer using pixel values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgSetTileOffsetPixel3D**(int x, int y, int z)	Sets the ***Tiling Offset*** of the Layer in 3D using pixel values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
void **MapDlgSetTileOffsetVector** (float x, float y)	Sets the ***Tiling Offset*** of the Layer using relative values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgSetTileOffsetVector3D** (float x, float y, float z)	Sets the ***Tiling Offset*** of the Layer in 3D using relative values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
void **MapDlgSetTilePixel**(int x, int y, int w, int h)	Tiles the Layer in a certain area of the mapping using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgSetTilePixel3D**(int x, int y, int z, int w, int h, int d)	Tiles the Layer in a certain area of the mapping using pixel values. *x, y and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgSetTileVector**(float x, float y, float w, float h)	Tiles the Layer to a certain area of the mapping using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgSetTileVector3D** (float x, float y, float z, float w, float h, float d)	Tiles the Layer in a certain area of the mapping in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y,* and *z* describe the coordinates. *w, h,* and *d* describe width, height, and depth.	+	+		
int **max**(int x, int y)	Returns the maximum value of the integer numbers *x* and *y*.	+	+	+	+
int **min**(int x, int y)	Returns the minimum value of *x* and *y*.	+	+	+	+
void **PixelFloodFill**(color col, int x, int y, int z)	Fills a single-colored area with the specified color *col*, starting at the given position *x, y* and *z*. The area is restricted to the given *z*-plane for which the default value is *0*.	+	+	+	+
float **pow**(float x, float y)	Returns the result of the calculation of *x* to the power of *y* .	+	+	+	+
float **rad2deg**(float a)	Converts the angle *a* from radian measure to degrees.	+	+	+	+
int **random**(int min, int max)	Returns a random integer number in the value range of *min* to *max*.	+	+	+	+
int **ReadAsync**(string file, string txt, int encoding)	Reads the content from a *file* with a certain *encoding* (default: automatic detection) as text into the string *txt*. » **Description And Examples**	+	+	+	+
void **RenderShape**(struct color, int shapeType, float positionX, float positionY, float positionZ, float sizeX, float sizeY, float sizeZ, struct shape, int unitType)	Renders 2D or 3D shapes onto the matrix. »**Render Shapes**	+	+	+	+
void **replace**(string src, string old, string new)	Replaces any appearances of *old* within *src* with *new*. » **Description**	+	+	+	+
int **rfindstring**(int startIndex, string text, string substring)	This function looks for the *substring* in the given *text* from its ending to the beginning. The search starts at the given *startIndex*, while the last charachter has the index 0. The function returns the index where the substring occurs for the first time after the specified *startIndex*. 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 **SetAsync**(int value)	Sets the asynchronous rendering mode. »**Description** This function is only available for some effects.	+	+		
void **SetBlind**(int value)	Sets the ***Blind Mode***. »**Description**	+	+		
void **SetChromaKey**(color c, int alpha)	Sets the specified alpha value for the specified color c. ***color c*** is a structure. Valid values for *alpha* range from *0* to *255*. **Note:** To be mainly used in *PostRenderEffect()*.	+	+	+	+
void **SetDmxFaderValue**(int index, int value)	Sets the value for a fader of the DMX Fader Tool. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255. **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+

void **SetFilter**(int filter)	Applies a ***Filter (FX)*** to the matrix. Valid values for *filter* are »Filters »Description	+	+	+	+
void **SetFrameId**(float id)	Sets a new Frame ID. If the given *id* is lower than 0, it is set to *0*. »Description	+	+		
void **SetLink**(int enable)	Enables or disables the option to link effect Layers. Valid values are 0 (**FALSE**) or 1 (**TRUE**).	+	+		
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. »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 And Examples	+	+	+	+
void **SetPixel3D**(color col, int x, int y, int z)	Sets the voxel at position (*x*, *y*, *z*) to the specified color.	+	+	+	+
void **SetPixelGrayscale**(int x, int y)	Renders the selected pixel at position (*x*, *y*) in grayscale. » Description	+	+	+	+
void **SetPixelGrayscale3D**(int x, int y, int z)	Renders the selected voxel at position (*x*, *y*, *z*) in grayscale.	+	+	+	+
void **SetPixelTransposeEntry**(int idx, int srcX, int srcY, int destX, int destY)	Adds one pixel transpose table entry defined by *idx*. *srcX* and *srcY* are the coordinates of the source in x and y. *destX* and *destY* are the destination coordinates. » Description	+	+	+	+
void **SetPixelTransposeEntry3D** (int idx, int srcX, int srcY, int srcZ, int destX, int destY, int destZ)	Adds one pixel transpose table entry defined by *idx*. *srcX*, *srcY* and *srcZ* are the coordinates of the source in x, y, and z. *destX*, *destY*, and *destZ* are the destination coordinates. »Description	+	+	+	+
int **SetReadAsyncInterval**(string file, int interval)	Sets the reading *interval* for a certain *file*. To be used in combination with ReadAsync. »Description And Examples	+	+	+	+
void **SetSolo**(int value)	Sets the ***Solo Mode***. »Description	+	+		
void **SetStep**(int value)	Sets the stepped rendering mode. »Description This function is only available for some effects.	+	+		
void **SetSubMaster**(int value)	Sets the ***Submaster*** value of the effect. Valid values range from *0* to *255*.	+	+		
void **SetVectorPixel**(color, float x, float y)	Draws one pixel at the given vector coordinates. In addition to the coordinates, a *color* must be specified.	+	+	+	+
void **SetVectorPixel3D**(color, float x, float y, float z)	Draws one voxel at the given vector coordinates. In addition to the coordinates, a *color* 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 *x*, *y*, *w*, and *h*) *step* pixel into the given direction. Exemplary values for *dir* are **SHIFT_UP**, **SHIFT_DOWN**, or **SHIFT_UL**. »List Of Global Constants **Note:** After shifting the pixels, the data is lost. It is not possible to undo a shifting operation. *x*, *y*, *w*, *h* describe the rectangle that should be shifted. *x* and *y* describe the upper left corner. *w* and *h* are the width and height of the rectangle. Learn more »ShiftMatrix	+	+	+	+

void **ShiftVectorMatrix**(float x, float y, float w, float h, int dir, float step)	Moves the area of the matrix (defined by *x*, *y*, *w*, and *h*) *step* pixel into the given direction. Exemplary values for *dir* are **SHIFT_UP**, **SHIFT_DOWN**, or **SHIFT_UL**. »**List Of Global Constants** **Note:** After shifting the pixels, the data is lost. It is not possible to undo a shifting operation. *x*, *y*, *w*, *h* describe the rectangle that should be shifted. *x* and *y* describe the upper left corner. *w* and *h* are the width and height of the rectangle. Learn more »**ShiftMatrix**	+	+	+	+
float **sin**(float a) float **sinDeg**(float a)	Returns the 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 **sqrt**(float x)	Returns the square root of *x*.	+	+	+	+
int **startswith**(string text, string substring)	This function checks if the *text* string starts with the given *substring*. If *text* starts with *substring*, 1 (**TRUE**) is returned. Otherwise 0 (**FALSE**) is returned. »**Description**	+	+	+	+
int **strcmp**(string str1, string str2)	Compares the two given strings each other. If they are equal, *0* is returned. A value of *-1* is returned if *str1* is less than *str2*. A value of *1* is returned if *str1* is greater than *str2*. »**Description**	+	+	+	+
void **strip**(string text)	Removes leading and ending white spaces, such as space, tabulator, line feeds, etc., from the given string *text*. » **Description**	+	+	+	+
string **substring**(string text, int start, int count)	Extracts a certain number of characters (*count*) beginning with *start* from the given *text*. If *count* is *-1*, all characters of the string starting at *start* are returned. »**Description** E.g.: substring("Hello", 0, 2) returns "He".	+	+	+	+
float **tan**(float a) float **tanDeg**(float a)	Returns the 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.	+	+	+	+
void **ToggleAsync**()	Toggles the asynchronous rendering mode. »**Description** This function is only available for some effects.	+	+		
void **ToggleBlind**()	Toggles ***Blind Mode***. »**Description**	+	+		
void **ToggleLink**()	Toggles ***Link Mode***, the option to link effect Layers.	+	+		
void **ToggleSolo**()	Toggles ***Solo Mode***. »**Description**	+	+		
void **ToggleStep**()	Toggles the stepped rendering mode. »**Description** This function is only available for some effects.	+	+		
void **tokenize**(string src, string delimiter, string reslist[])	Strips the string *src* into smaller pieces delimited by characters within *delemiter*. The result is returned in *reslist*. »**Description**	+	+	+	+
void **tolower**(string text)	Converts each character of the given string *text* into a lowercase character. »**Description**	+	+	+	+
void **toupper**(string text)	Converts each character of the given string *text* 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, float z)	Fills a single-colored area with the specified color *col*, starting at the given position *x*, *y* and *z*. The area is restricted to the given *z*-plane for which the default value is *0*. In contrast to PixelFloodFill(), 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**	+	+	+	+
void **WriteTextClear**()	Clears the Script Output window from all messages.	+	+	+	+

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:

- **SetFrameCount**

- **BlackTraxBeaconClear**
- **BlackTraxBeaconGetAccelerationX**
- **BlackTraxBeaconGetAccelerationY**
- **BlackTraxBeaconGetAccelerationZ**
- **BlackTraxBeaconGetEulerLatency**
- **BlackTraxBeaconGetEulerOrder**
- **BlackTraxBeaconGetEulerRotation1**
- **BlackTraxBeaconGetEulerRotation2**
- **BlackTraxBeaconGetEulerRotation3**
- **BlackTraxBeaconGetPositionCentroidLatency**
- **BlackTraxBeaconGetPositionCentroidX**
- **BlackTraxBeaconGetPositionCentroidY**
- **BlackTraxBeaconGetPositionCentroidZ**
- **BlackTraxBeaconGetPositionTrackedLatency**
- **BlackTraxBeaconGetPositionTrackedX**
- **BlackTraxBeaconGetPositionTrackedY**

- **BlackTraxBeaconGetPositionTrackedZ**
- **BlackTraxBeaconGetQuaterionsLatency**
- **BlackTraxBeaconGetQuaterniosW**
- **BlackTraxBeaconGetQuaterniosX**
- **BlackTraxBeaconGetQuaterniosY**
- **BlackTraxBeaconGetQuaterniosZ**
- **BlackTraxBeaconGetTimestamp**
- **BlackTraxBeaconGetVelocityX**
- **BlackTraxBeaconGetVelocityY**
- **BlackTraxBeaconGetVelocityZ**
- **BlackTraxBeaconIsExist**

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 corresponding chapters for each MADRIX Effect.

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:

- **GetDescription**
- **GetLayerCount**
- **GetPause**
- **GetSpeedMaster**
- **GetSubMaster**
- **LayerGetAsync**
- **LayerGetBlind**

- **LayerGetFilter**
- **LayerGetFrameCount**
- **LayerGetFrameId**
- **LayerGetFrameSteps**
- **LayerGetLink**
- **LayerGetMixMode**
- **LayerGetOpacity**
- **LayerGetSolo**
- **LayerGetStep**
- **LayerGetSubMaster**
- **LayerMapDlgGetMapMode**
- **LayerMapDlgGetMapPixel**
- **LayerMapDlgGetMapPixel3D**
- **LayerMapDlgGetMapVector**
- **LayerMapDlgGetMapVector3D**
- **LayerMapDlgGetRotationDegree**
- **LayerMapDlgGetRotationMode**
- **LayerMapDlgGetRotationVector**
- **LayerMapDlgGetRotationXDegree**
- **LayerMapDlgGetRotationXMode**
- **LayerMapDlgGetRotationXVector**
- **LayerMapDlgGetRotationYDegree**
- **LayerMapDlgGetRotationYMode**
- **LayerMapDlgGetRotationYMode**
- **LayerMapDlgGetRotationYVector**
- **LayerMapDlgGetRotationZDegree**
- **LayerMapDlgGetRotationZMode**
- **LayerMapDlgGetRotationZVector**
- **LayerMapDlgGetTileMode**
- **LayerMapDlgGetTileOffsetPixel**
- **LayerMapDlgGetTileOffsetPixel3D**

- **LayerMapDlgGetTileOffsetVector**
- **LayerMapDlgGetTileOffsetVector3D**
- **LayerMapDlgGetTilePixel**
- **LayerMapDlgGetTilePixel3D**
- **LayerMapDlgGetTileVector**
- **LayerMapDlgGetTileVector3D**
- **LayerMapDlgIsMapped**
- **LayerMapDlgSetMapMode**
- **LayerMapDlgSetMapPixel**
- **LayerMapDlgSetMapPixel3D**
- **LayerMapDlgSetMapVector**
- **LayerMapDlgSetMapVector3D**
- **LayerMapDlgSetRotationDegree**
- **LayerMapDlgSetRotationMode**
- **LayerMapDlgSetRotationVector**
- **LayerMapDlgSetRotationXDegree**
- **LayerMapDlgSetRotationXMode**
- **LayerMapDlgSetRotationXVector**
- **LayerMapDlgSetRotationYDegree**
- **LayerMapDlgSetRotationYMode**
- **LayerMapDlgSetRotationYMode**
- **LayerMapDlgSetRotationYVector**
- **LayerMapDlgSetRotationZDegree**
- **LayerMapDlgSetRotationZMode**
- **LayerMapDlgSetRotationZVector**
- **LayerMapDlgSetTileMode**
- **LayerMapDlgSetTileOffsetPixel**
- **LayerMapDlgSetTileOffsetPixel3D**
- **LayerMapDlgSetTileOffsetVector**
- **LayerMapDlgSetTileOffsetVector3D**
- **LayerMapDlgSetTilePixel**

- **LayerMapDlgSetTilePixel3D**
- **LayerMapDlgSetTileVector**
- **LayerMapDlgSetTileVector3D**
- **LayerSetAsync**
- **LayerSetBlind**
- **LayerSetFilter**
- **LayerSetFrameId**
- **LayerSetLink**
- **LayerSetMixMode**
- **LayerSetOpacity**
- **LayerSetSolo**
- **LayerSetStep**
- **LayerSetSubMaster**
- **LayerToggleAsync**
- **LayerToggleBlind**
- **LayerToggleLink**
- **LayerToggleSolo**
- **LayerToggleStep**
- **SetDescription**
- **SetPause**
- **SetSpeedMaster**
- **SetSubMaster**
- **TogglePause**

Functions Only Available For The Global Macro

You can find a detailed description in the chapter »**Global Macro: Functions**

Here is an overview:

- **CallGroupPreset**
- **CueAdd**

- **CueDelete**
- **CueDeleteAll**
- **CueDeleteCurrent**
- **CueGetDateDay**
- **CueGetDateMonth**
- **CueGetDateString**
- **CueGetDateWeekday**
- **CueGetDateYear**
- **CueGetDescription**
- **CueGetDurationFrame**
- **CueGetDurationHour**
- **CueGetDurationMinute**
- **CueGetDurationSecond**
- **CueGetDurationString**
- **CueGetFadeTime**
- **CueGetFadeTimeString**
- **CueGetFadeType**
- **CueGetFadeTypeString**
- **CueGetFollowCue**
- **CueGetGroupPreset**
- **CueGetPlace**
- **CueGetStorage**
- **CueGetTimeCodeFrame**
- **CueGetTimeCodeHour**
- **CueGetTimeCodeMinute**
- **CueGetTimeCodeSecond**
- **CueGetTimeCodeString**
- **CuelistBack**
- **CuelistCount**
- **CuelistCueAllOccupied**
- **CuelistCurrentCue**

- **CuelistGetTimecodeFormat**
- **CuelistGetTimecodeSource**
- **CuelistGo**
- **CuelistGoto**
- **CuelistNew**
- **CuelistPlay**
- **CuelistProgress**
- **CuelistSetTimecodeFormat**
- **CuelistSetTimecodeSource**
- **CuelistStop**
- **CueSetDate**
- **CueSetDateString**
- **CueSetDateWeekday**
- **CueSetDescription**
- **CueSetDuration**
- **CueSetDurationString**
- **CueSetFadeTime**
- **CueSetFadeType**
- **CueSetFollow**
- **CueSetGroupPreset**
- **CueSetPlace**
- **CueSetStorage**
- **CueSetTimeCode**
- **CueSetTimeCodeString**
- **GetAudioInputFader**
- **GetAudioInputMute**
- **GetAudioOutputFader**
- **GetAudioOutputMute**
- **GetAutoGainControl**
- **GetBlackout**
- **GetFadeTime**

- **GetFadeType**
- **GetFadeValue**
- **GetFilter**
- **GetFilterColor**
- **GetFreeze**
- **GetGroupCount**
- **GetGroupDefaultValue**
- **GetGroupDefaultValueByIndex**
- **GetGroupDisplayColor**
- **GetGroupDisplayColorByIndex**
- **GetGroupDisplayName**
- **GetGroupDisplayNameByIndex**
- **GetGroupFadeTime**
- **GetGroupFlashMode**
- **GetGroupFlashModeByIndex**
- **GetGroupIdByIndex**
- **GetGroupValue**
- **GetGroupValueByIndex**
- **GetMasterFader**
- **GetStorage**
- **GetStoragePause**
- **GetStoragePlace**
- **GetStoragePlaceFilter**
- **GetStoragePlaceFullState**
- **GetStoragePlaceSubMaster**
- **GetStorageSpeedMaster**
- **GetStorageSubMaster**
- **GetStrobe**
- **GetStrobeColor**
- **GetStrobeValue**
- **ImportFixtureGroupController**

- **ImportPatch**
- **ImportStorage**
- **ImportStoragePlace**
- **SetAudioInputFader**
- **SetAudioInputMute**
- **SetAudioOutputFader**
- **SetAudioOutputMute**
- **SetAutoGainControl**
- **SetBlackout**
- **SetFade**
- **SetFadeTime**
- **SetFadeType**
- **SetFadeValue**
- **SetFilter**
- **SetFilterColor**
- **SetFreeze**
- **SetGroupFadeTime**
- **SetGroupFlashMode**
- **SetGroupFlashModeByIndex**
- **SetGroupPreset**
- **SetGroupValue**
- **SetGroupValueByIndex**
- **SetMasterFader**
- **SetStorage**
- **SetStoragePause**
- **SetStoragePlace**
- **SetStoragePlaceFilter**
- **SetStoragePlaceSubMaster**
- **SetStorageSpeedMaster**
- **SetStorageSubMaster**
- **SetStrobe**

- **SetStrobeColor**
- **SetStrobeValue**
- **ToggleGroupFlashMode**
- **ToggleGroupFlashModeByIndex**

3.3 List Of Functions (Grouped)

Overview

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

Further information is provided in the chapter »**List Of Functions (Alphabetical Order)**.

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Global Macro
Draw Functions					
void **CaptureScreen**(color matrix [][], int xSrc, int ySrc, int w, int h)	Retrieves data from the screen and stores it into a 2-dimensional array of colors. »**Description**	+	+	+	+
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 *alpha* 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 *oldCol* by another one.	+	+	+	+
void **SetPixel**(color col, int x, int y)	Sets the pixel at position (*x*, *y*) to the specified color. » **Description And Examples**	+	+	+	+
void **SetPixel3D**(color col, int x, int y, int z)	Sets the voxel at position (*x*, *y*, *z*) to the specified color.	+	+	+	+
void **SetPixelGrayscale**(int x, int y)	Renders the selected pixel at position (*x*, *y*) in grayscale. » **Description And Examples**	+	+	+	+
void **SetPixelGrayscale3D**(int x, int y, int z)	Renders the selected voxel at position (*x*, *y*, *z*) in grayscale.	+	+	+	+
color **GetPixel**(int x, int y)	Returns the color of the pixel at position (*x*, *y*).	+	+	+	+
color **GetPixel3D**(int x, int y, int z)	Returns the color of the voxel at position (*x*, *y*, *z*).	+	+	+	+

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 array of colors. »**Description**	+	+	+	+
void **GetPixelArea3D**(color matrix[][][], int xSrc, int ySrc, int zSrc, int w, int h, int d, int xDst, int yDst, int zDst)	Retrieves data from the matrix and stores it into a 3-dimensional array 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 array of colors and renders it on the matrix. »**Description**	+	+	+	+
void **DrawPixelArea3D**(color matrix[][][], int xDst, int yDst, int zDst, int w, int h, int d, int xSrc, int ySrc, int zSrc, color filter)	Copies data from a 3-dimensional array of colors and renders it on the matrix. »**Description**	+	+	+	+
void **DrawPixelBitmap**(string file, int x, int y)	Loads the specified image file and renders it on the matrix starting at position *(x, y)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawVectorBitmap**(string file, int x, int y)	Loads the specified image file and renders it on the matrix starting at the relative position *(x, y)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawPixelBitmap3D**(string file, int x, int y, int z)	Loads the specified image file and renders it on the matrix starting at position *(x, y, z)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawVectorBitmap3D** (string file, int x, int y, int z)	Loads the specified image file and renders it on the matrix starting at the relative position *(x, y, z)*. **Note:** Please use cautiously! The higher the total pixel resolution of the image itself and the higher the pixel resolution of the virtual LED matrix in the software (Matrix Size), the higher are the performance requirements of this function. Certain file formats may have a better performance (*.bmp, *.jpg) than others (*.png).	+	+	+	+
void **DrawPixelLine**(color col, int x1, int y1, int x2, int y2)	Draws a line from pixel (*x1, y1*) to pixel (*x2, y2*) with the specified color.	+	+	+	+
void **DrawPixelLine3D**(color col, int x1, int y1, int z1, int x2, int y2, int z2)	Draws a line from voxel (*x1, y1, z1*) to voxel (*x2, y2, z2*) with the specified color *col*.	+	+	+	+
void **DrawVectorLine**(color col, float x1, float y1, float x2, float y2)	Draws a line from the relative position (*x1, y1*) to relative position (*x2, y2*) with the specified color.	+	+	+	+

void **DrawVectorLine3D**(color col, float x1, float y1, float z1, float x2, float y2, float z2)	Draws a line from the relative position (*x1, y1, z1*) to the relative position (*x2, y2, z2*) with the specified color *col*.	+	+	+	+
void **DrawPixelText**(color c, font f, string t, int x, int y, int rotation)	Draws a text across the main output. ***color c*** and ***font f*** are structures. Valid values for rotation are **ROTATION_CCW_0**, **ROTATION_CCW_90**, **ROTATION_CCW_180**, **ROTATION_CCW_270** for counter-clockwise rotation and **ROTATION_CW_0**, **ROTATION_CW_90**, **ROTATION_CW_180**, **ROTATION_CW_270** for clockwise rotation. »**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. ***color c*** and ***font f*** are structures. Valid values for rotation are **ROTATION_CCW_0**, **ROTATION_CCW_90**, **ROTATION_CCW_180**, **ROTATION_CCW_270** for counter-clockwise rotation and **ROTATION_CW_0**, **ROTATION_CW_90**, **ROTATION_CW_180**, **ROTATION_CW_270** for clockwise rotation. »**Example 1** » **Example 2**	+	+	+	+
void **DrawPixelShape**(color col, int shape, int x, int y, int z, int w, int h, int d, int lineWidth, int drawMode, int lookAtType)	Draws a given *shape* onto the matrix with the given size (width *w*, height *h*, depth *d*) at the given position (*x, y, z*) with the specified color *col*. Learn more »**Draw Shapes**	+	+	+	+
void **DrawVectorShape**(color col, int shape, float x, float y, float z, float w, float h, float d, int lineWidth, int drawMode, int lookAtType)	Draws a given *shape* at the relative position (*x, y, z*) with the relative size (width *w*, height *h*, depth *d*) and the specified color *col*. Learn more »**Draw Shapes**	+	+	+	+
void **ShiftPixelMatrix**(int x, int y, int w, int h, int dir, int step)	Moves the area of the matrix (defined by *x, y, w*, and *h*) *step* pixel into the given direction. Exemplary values for *dir* are **SHIFT_UP**, **SHIFT_DOWN**, or **SHIFT_UL**. »**List of Global Constants** **Note:** After shifting the pixels, the data is lost. It is not possible to undo a shifting operation. *x, y, w, h* describe the rectangle that should be shifted. *x* and *y* describe the upper left corner. *w* and *h* are the width and height of the rectangle. Learn more »**ShiftMatrix**	+	+	+	+
void **ShiftVectorMatrix**(float x, float y, float w, float h, int dir, float step)	Moves the area of the matrix (defined by *x, y, w*, and *h*) *step* pixel into the given direction. Exemplary values for *dir* are **SHIFT_UP**, **SHIFT_DOWN**, or **SHIFT_UL**. »**List of Global Constants** **Note:** After shifting the pixels, the data is lost. It is not possible to undo a shifting operation. *x, y, w, h* describe the rectangle that should be shifted. *x* and *y* describe the upper left corner. *w* and *h* are the width and height of the rectangle. Learn more »**ShiftMatrix**	+	+	+	+
void **VectorFloodFill**(color col, float x, float y, float z)	Fills a single-colored area with the specified color *col*, starting at the given position *x, y* and *z*. The area is restricted to the given *z*-plane for which the default value is *0*. In contrast to PixelFloodFill(), the coordinates are given as floating point values ranging from *0.0* to *1.0*.	+	+	+	+
void **PixelFloodFill**(color col, int x, int y, int z)	Fills a single-colored area with the specified color *col*, starting at the given position *x, y* and *z*. The area is restricted to the given *z*-plane for which the default value is *0*.	+	+	+	+

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 **Grayscale**()	Converts the whole matrix to grayscaled values.	+	+	+	+
void **RenderShape**(struct color, int shapeType, float positionX, float positionY, float positionZ, float sizeX, float sizeY, float sizeZ, struct shape, int unitType)	Renders 2D or 3D shapes onto the matrix. »**Render Shapes**	+	+	+	+
void **SetVectorPixel**(color, float x, float y)	Draws one pixel at the given vector coordinates. In addition to the coordinates, a *color* must be specified.	+	+	+	+
void **SetVectorPixel3D**(color, float x, float y, float z)	Draws one voxel at the given vector coordinates. In addition to the coordinates, a *color* must be specified.	+	+	+	+
color **GetVectorPixel**(float x,float y)	Returns the *color* of the pixel drawn at vector coordinates *x* and *y*.	+	+	+	+
color **GetVectorPixel3D**(float x, float y,float z)	Returns the *color* of the voxel drawn at vector coordinates *x, y, and z*.				
void **Filter**(int filter)	Renders a filter over the matrix. »**Valid values (Filters)** » **Description**	+	+	+	+
void **FilterColor**(color c)	Renders a color filter over the matrix. The values of the specified color will be passed through and all other colors will be filtered out.	+	+	+	+
void **SetFilter**(int filter)	Applies a ***Filter (FX)*** to the matrix. Valid values for *filter* are »**Filters** »**Description**	+	+	+	+
int **GetFilter**()	Returns which ***Filter (FX)*** is applied to the matrix.	+	+	+	+
void **SetChromaKey**(color c, int alpha)	Sets the specified alpha value for the specified color c. ***color c*** is a structure. Valid values for *alpha* range from *0* to *255*. **Note:** To be mainly used in *PostRenderEffect()*.	+	+	+	+
S2L Functions					
float **GetSoundLevel**(int channel)	Returns a value, ranging from *0.0* to *1.0*, 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 array *buf* with the sound level values for each note (buf[0] ... buf[127]). *buf[index]* can differ from 0 to 127.	+	+		
void **GetAllIntervals**(int buf[])	Fills the array *buf* with the occurrences of each interval (buf[0] ... buf[10]). *buf[index]* is 1 (**TRUE**) if the specified interval was analyzed.	+	+		
int **IsInterval**(int index)	Returns 1 (**TRUE**) if the specified interval (at the *index* position, ranging from *0* to *10*) was analyzed.	+	+		

int **IsNote**(int note)	Returns 1 (**TRUE**) 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 1 (**TRUE**) if the tonality could be determined.	+	+		
Mathematical Functions					
int **random**(int min, int max)	Returns a random integer number in the value range of *min* to *max*.	+	+	+	+
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 **arccot**(float a) float **arccotDeg**(float a)	Returns the arc cotangent of the angle *a* in radian measure or degrees, respectively.	+	+	+	+
float **cotan**(float a) float **cotanDeg**(float a)	Returns the cotangent of the angle *a* in radian measure or degrees, respectively.	+	+	+	+
float **arctan2**(float x, float y)	Returns the angle of polar coordinates from two Cartesian coordinates *x* and *y*.	+	+	+	+
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. ceil(2.00001) = 3.0	+	+	+	+
float **fmod**(float denominator, float divisor)	Calculates the remainder of the float division.	+	+	+	+
Mapping Functions					
int **MapDlgIsMapped**()	Retrieves if the Layer is mapped.	+	+		
void **MapDlgGetMapVector**(float map[])	Retrieves the map settings for the Layer using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapVector**(float x, float y, float w, float h)	Maps the Layer to a certain area of the matrix using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgGetMapVector3D** (float map[])	Retrieves the map settings for the Layer in 3D using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapVector3D** (float x, float y, float z, float w, float h, float d)	Maps the Layer to a certain area of the matrix in 3D using relative values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetMapPixel**(int map[])	Retrieves the map settings for the Layer using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapPixel**(int x, int y, int w, int h)	Maps the Layer to a certain area of the matrix using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetMapPixel3D**(int map[])	Retrieves the map settings for the Layer in 3D using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapPixel3D**(int x, int y, int z, int w, int h, int d)	Maps the Layer to a certain area of the matrix in 3D using pixel values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
int **MapDlgGetMapMode**()	Retrieves the currently used ***Map Mirror Mode*** of the Layer. See »here for a list of constants.	+	+		
void **MapDlgSetMapMode**(int mode)	Sets the ***Map Mirror Mode*** for the Layer. See »here for a list of constants.	+	+		

void **MapDlgGetTileVector**(float tile[])	Retrieves the ***Tiling*** settings for the Layer using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTileVector**(float x, float y, float w, float h)	Tiles the Layer to a certain area of the mapping using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetTileVector3D** (float tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTileVector3D** (float x, float y, float z, float w, float h, float d)	Tiles the Layer in a certain area in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetTilePixel**(int tile [])	Retrieves the ***Tiling*** settings for the Layer using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTilePixel**(int x, int y, int w, int h)	Tiles the Layer in a certain area using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgGetTilePixel3D**(int tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTilePixel3D**(int x, int y, int z, int w, int h, int d)	Tiles the Layer in a certain area using pixel values. *x, y and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetTileOffsetVector** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetVector** (float x, float y)	Sets the ***Tiling Offset*** of the Layer using relative values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgGetTileOffsetVector3D** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetVector3D** (float x, float y, float z)	Sets the ***Tiling Offset*** of the Layer in 3D using relative values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
void **MapDlgGetTileOffsetPixel** (int offset[])	Retrieves the ***Tiling Offset*** of the Layer using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetPixel** (int x, int y)	Sets the ***Tiling Offset*** of the Layer using pixel values. *x* describes Offset X, while *y* represents Offset Y.	+	+		

void **MapDlgGetTileOffsetPixel3D**(int offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetPixel3D**(int x, int y, int z)	Sets the ***Tiling Offset*** of the Layer in 3D using pixel values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
int **MapDlgGetTileMode**()	Retrieves the currently used ***Tile Mode*** of the Layer. See »here for a list of Tile Mode constants.	+	+		
void **MapDlgSetTileMode**(int mode)	Sets the ***Tile Mode*** of the Layer. See »here for a list of Tile Mode constants.	+	+		
void **MapDlgGetRotationVector** (float rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values. The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationVector** (float x, float y, float z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values.	+	+		
void **MapDlgGetRotationDegree** (int rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.The values are saved in an array *(rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationDegree** (int x, int y, int z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.	+	+		
float **MapDlgGetRotationXVector**()	Retrieves the ***Rotation*** for the X-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationXVector** (float value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using degree values.	+	+		
int **MapDlgGetRotationXDegree** ()	Retrieves the ***Rotation*** for the X-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationXDegree**(int value)	Sets the ***Rotation*** *value* for the X-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationXMode**()	Retrieves the ***Rotation*** mode for the X-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		

void **MapDlgSetRotationXMode** (int mode)	Sets the ***Rotation*** mode for the X-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationYVector**()	Retrieves the ***Rotation*** for the Y-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationYVector** (float value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationYDegree** ()	Retrieves the ***Rotation*** for the Y-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationYDegree**(int value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationYMode**()	Retrieves the ***Rotation*** mode for the Y-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationYMode** (int mode)	Sets the ***Rotation*** mode for the Y-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationZVector**()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationZVector** (float value)	Sets the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationZDegree** ()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationZDegree**(int value)	Sets the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationZMode**()	Retrieves the ***Rotation*** mode for the Z-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationZMode** (int mode)	Sets the ***Rotation*** mode for the Z-axis of the Layer. See »here for a list of Rotation Mode constants.	+	+		
void **MapDlgGetRotationMode** (int mode[])	Retrieves the ***Rotation*** mode for axes *x*, *y*, and *z* of the Layer. See »here for a list of Rotation Mode constants. The values are saved in an array (*mode[]*). ▪ mode[0] = X-coordinate (X-Axis) ▪ mode[1] = Y-coordinate (Y-Axis) ▪ mode[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationMode** (int x, int y, int z)	Sets the ***Rotation*** mode for axes *x, y, and z* of the Layer. See »here for a list of Rotation Mode constants.	+	+		
int **MapDlgGetAntiAliasing**()	Retrieves the ***Anti-Aliasing*** mode of the Layer. See »here for a list of Anti-Aliasing Mode constants.	+	+		
void **MapDlgSetAntiAliasing**(int aaLevel)	Sets the ***Anti-Aliasing*** mode of the Layer. See »here for a list of Anti-Aliasing Mode constants.	+	+		
String Functions					

int **findstring**(int startIndex, string text, string substring)	Searches for the *substring* within *text* starting at *startIndex*. The function returns an index that describes the position at which the *substring* begins. »**Description** Example: findstring(0, "`Hello World!`", "World") returns 6. If the *substring* is not found within *text*, *-1* is returned.	+	+	+	+
string **substring**(string text, int start, int count)	The function extracts *count* characters from the given *text* starting with *start*. If *count* is *-1*, all characters of the string starting at *start* are returned. »**Description** E.g.: substring("Hello", 0, 2) returns "He".	+	+	+	+
int **rfindstring**(int startIndex, string text, string substring)	This functions looks for the *substring* in the given text from its end to the beginning. The function starts its search at a specified position of the entire *text* using *startIndex* and returns an index that describes the position at which the *substring* begins. If the substring was not found, *-1* is returned.	+	+	+	+
int **startswith**(string text, string substring)	This function checks if the string *text* starts with the given *substring*. If *text* starts with *substring*, 1 (**TRUE**) is returned, otherwise 0 (**FALSE**).	+	+	+	+
int **endswith**(string text, string substring)	This function checks if the string *text* ends with the given *substring*. If *text* ends with *substring*, 1 (**TRUE**) is returned, otherwise 0 (**FALSE**).	+	+	+	+
int **isalnum**(string text)	Returns 1 (**TRUE**) if the given string contains only characters and figures and if its length is greater then 0. Otherwise, 0 (**FALSE**) is returned.	+	+	+	+
int **isalpha**(string text)	Returns 1 (**TRUE**) if the given string contains only characters and if its length is greater than 0, otherwise 0 (**FALSE**) is returned.	+	+	+	+
int **isnum**(string text)	Returns 1 (**TRUE**) if the given *text* represents a number. This may be an integer number or a floating point number (e.g. 1.3). Otherwise, it returns 0 (**FALSE**).	+	+	+	+
void **tolower**(string text)	Converts each character of the given *string* into a lower-case character.	+	+	+	+
void **toupper**(string text)	Converts each character of the given *string* into an upper-case character.	+	+	+	+
string **hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are lower-cased (a-f).	+	+	+	+
string **Hex**(int value)	Formats the given *value* as a hexadecimal number. Letters are upper-cased (A-F).	+	+	+	+
void **strip**(string text)	Removes leading and ending white spaces, like space, tabulator, line feeds, etc. from the given *string*.	+	+	+	+
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 *delemiter*. The result is returned in *reslist*. »**Description**	+	+	+	+
Effect-Controlling Functions					

void **SetAsync**(int value)	Sets the asynchronous rendering mode. »Description This function is only available for some effects.	+	+		
int **GetAsync**()	Returns if asynchronous rendering is used. »Description This function is only available for some effects.	+	+		
void **ToggleAsync**()	Toggles the asynchronous rendering mode. »Description This function is only available for some effects.	+	+		
void **SetBlind**(int value)	Sets ***Blind Mode***. »Description	+	+		
int **GetBlind**()	Returns if ***Blind Mode*** is used. »Description	+	+		
void **ToggleBlind**()	Toggles ***Blind Mode***. »Description	+	+		
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* for MAS Script. The initial value is *1000.0*.	+	+		
void **SetSolo**(int value)	Sets ***Solo Mode***. »Description	+	+		
int **GetSolo**()	Returns if ***Solo Mode*** is used. »Description	+	+		
void **ToggleSolo**()	Toggles ***Solo Mode***. »Description	+	+		
void **SetStep**(int value)	Sets the stepped rendering mode. »Description This function is only available for some effects.	+	+		
int **GetStep**()	Returns if stepped rendering is used. »Description This function is only available for some effects.	+	+		
void **ToggleStep**()	Toggles the stepped rendering mode. »Description This function is only available for some effects.	+	+		
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.	+	+	+	+
void **WriteTextClear**()	Clears the Script Output window from all messages.	+	+	+	+
int **GetMatrixDepth**()	Returns the pixel count of the rendered matrix regarding the depth. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Matrix Size, since they can be mapped via Map Settings.				
int **GetMatrixWidth**()	Returns the horizontal pixel count of the rendered matrix. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Matrix Size, since they can be mapped via Map Settings.	+	+	+	+

int **GetMatrixHeight**()	Returns the vertical pixel count of the rendered matrix. In case of Global Macro and Storage Place Macro, this is the Matrix Size. When used as Effect Macro or MAS Script, the result may differ from the Matrix Size, since they can be mapped via Map Settings.	+	+	+	+
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 **GetDayOfWeek**(int day, int month, int year)	Returns the day of the week of a specified date. Returned values include *0 = Sunday*, *1 = Monday*, ..., *6 = Saturday*. *-1* is returned if the provided parameters are invalid. Valid values for *day* range from *1* to *31*. Valid values for *month* range from *1* to *12*. Valid values for *year* start with value *1900*.	+	+	+	+
int **GetMixMode**()	Returns the currently set ***Mix Mode***. »**Valid values (Mix Modes)** »**Description**	+	+		
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. »**Description**	+	+		
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 are *0* (**FALSE**) or *1* (**TRUE**).	+	+		
int **GetLink**()	Returns *1* (**TRUE**) if the option to link effect Layers is enabled, otherwise *0* (**FALSE**).	+	+		
void **ToggleLink**()	Toggles ***Link Mode***, the option to link effect Layers.	+	+		
int **ReadAsync**(string file, string txt, int encoding)	Reads the content from a *file* with a certain *encoding* (default: automatic detection) as text into the string *txt*. » **Description And Examples**	+	+	+	+
int **SetReadAsyncInterval**(string file, int interval)	Sets the reading *interval* for a certain *file*. To be used in combination with **ReadAsync**. »**Description And Examples**	+	+	+	+
string **GetApplicationPath**()	Locates the MADRIX.exe on your harddisk and returns the path as a *string*. »**Example**	+	+	+	+
string **GetUserProfileDirectory**()	Locates the directory of the current user profile on the harddisk and returns the path as a *string*. »**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.	+	+	+	+
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 *string*. »**Example**	+	+	+	+
string **GetSoftwareVersion**()	Returns the MADRIX software version and returns the value as a *string*. »**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 *major* and *minor*. Or else *0* is returned. The current Script Engine Version is 3.12. 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 *major*, *minor*, *subminor*, and *subsubminor*. Or else *0* is returned. The current MADRIX version is 5.3.2.30. 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 (*DmxValues[]*). To use the function, choose a DMX start channel and specify the number of channels that should be scanned. Valid values for *startchannel* range from *0* to *511*. Valid values for *channels* range from *0* to *511*. Please note that a maximum of 512 channels can be scanned. This means that the sum of *startchannel* + *channels* must be less or equal to 512. In addition, specify the universe that should be scanned. Valid values for *universe* range from *0* to *255*. The default *universe* 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 (*channel*). Valid values for *channel* range from *0* to *511*. In addition, specify the universe that should be scanned. Valid values for *universe* range from *0* to *255*. The default value of *universe* 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 *size* and *growsize*.	+	+	+	+
void **SetPixelTransposeEntry**(int idx, int srcX, int srcY, int destX, int destY)	Adds one pixel transpose table entry defined by *idx*. *srcX* and *srcY* are the x and y coordinates of the source. *destX* and *destY* are the destination coordinates.	+	+	+	+
void **SetPixelTransposeEntry3D** (int idx, int srcX, int srcY, int srcZ, int destX, int destY, int destZ)	Adds one pixel transpose table entry defined by *idx*. *srcX*, *srcY* and *srcZ* are the coordinates of the source in x, y, and z. *destX*, *destY*, and *destZ* are the destination coordinates. »**Description**	+	+	+	+
void **AddPixelTransposeEntry** (int srcX, int srcY, int destX, int destY)	Adds one entry to the pixel transpose table and resizes the table if necessary. *srcX* and *srcY* are the coordinates in X and Y of the source. *destX* and *destY* are the destination coordinates.	+	+	+	+

void **AddPixelTransposeEntry3D** (int srcX, int srcY, int srcZ, int destX, int destY, int destZ)	Adds one entry to the pixel transpose table and resizes the table if necessary. *srcX, srcY*, and srcZ are the coordinates in X, Y, and Z of the source. *destX, destY*, and *destZ* are the coordinates of the destination. »**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.	+	+	+	+
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 (*midinote)* and *midichannel* for the specified *device*. The default value of *note*, *midichannel*, and *device* 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 *control change (midicontrol)* and *midichannel* for the specified *device*. The default value of *control*, *midichannel*, and *device* 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 (*midivalues[]*). To use the function, choose a start note (*startnote*) and the number of notes (*notecount*), as well as the MIDI channel (*midichannel*) and the *device*. The default value of *startnote*, *midichannel*, and *device* is *0*, i.e. the first control, channel, and device. The default and maximum value of *notecount* 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 (*midivalues[]*). To use the function, choose a start control (*startcontrol*) and the number of controls (*controlcount*), as well as the MIDI channel (*midichannel*) and the *device*. The default value of *startnote*, *midichannel*, and *device* is *0*, i.e. the first control, channel, and device. The default and maximum value of *controlcount* 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 *value* range from *0.0* to *1.0*.	+	+	+	+
void **DimPixel**(float value, int x, int y)	Reduces the brightness of an individual pixel. *x* and *y* are the coordinates of the pixel. Valid values for *value* range from *0.0* to *1.0*.	+	+	+	+
void **DimPixel3D**(float value, int x, int y, int z)	Reduces the brightness of an individual voxel. *x, y*, and z are the coordinates of the voxel. Valid values for *value* 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. *x* and *y* are the coordinates of the area (upper left corner). *width* and *height* specify the width and height of the area. Valid values for *value* range from *0.0* to *1.0*.	+	+	+	+
void **DimPixelArea3D**(float value, int x, int y, int z, int width, int height, int depth)	Reduces the brightness of a certain area of the virtual 3D matrix. *x*, *y*, and z are the coordinates of the area (front upper left corner). *width*, *height*, and *depth* specify the width, height, and depth of the area. Valid values for *value* range from *0.0* to *1.0*.	+	+	+	+
Sunset And Sunrise Functions					
time **GetTimeSunrise**(struct date, int latitude_degrees, int latitude_minutes, int longitude_degrees, int longitude_minutes, float time_zone_UTC_offset)	Returns the sunrise time of a geographic location at the specified *date*. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunriseCity**(struct date, int city)	Returns the sunrise time of a specified *city* at the specified *date*. Valid values for city are »**city constants**. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunset**(struct date, int latitude_degrees, int latitude_minutes, int longitude_degrees, int longitude_minutes, float time_zone_UTC_offset)	Returns the sunset time of a geographic location at the specified *date*. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
time **GetTimeSunsetCity**(struct date, int city)	Returns the sunset time of a specified *city* at the specified *date*. Valid values for city are »**city constants**. **Note**: Please use cautiously! Due to the performance requirements of this function, it is not recommended to call it often. Please call this function seldomly in your script/macro. »**Example**	+	+	+	+
DMX Fader Tool					
int **GetDmxFaderChannel**(int index)	Returns on which DMX channel the specified fader sends data. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 511 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
int **GetDmxFaderCount**()	Returns how many faders are available in the DMX Fader Tool.	+	+	+	+
int **GetDmxFaderUniverse**(int value)	Returns on which DMX universe the specified fader sends data. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
int **GetDmxFaderValue**(int value)	Returns the DMX value the specified fader sends. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255 (Counting starts with 0). **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+
int **IsDmxFaderEnable**(int index)	Returns if the specified fader is enabled (*1*) or not (*0*). Valid values for *index* range from 0 to 11 (Indexing starts with 0).	+	+	+	+

void **SetDmxFaderValue**(int index, int value)	Sets the value for a fader of the DMX Fader Tool. Valid values for *index* range from 0 to 11 (Indexing starts with 0). Valid values for *value* range from 0 to 255. **Note:** Make sure to activate the fader in the DMX Fader Tool first!	+	+	+	+

3.4 List Of Global Variables And Constants

Variable/Constant	Description
Math	
float **PI**	Represents the number PI with a value of 3.141592.
S2L - Sound2Light	
float **SOUND_DATA_LEFT[]**	Contains the sound values of the left audio channel. 511 bands/values in a range from *0.0* to *1.0*.
float **SOUND_DATA_RIGHT[]**	Contains the sound values of the right audio channel. 511 bands/values in a range from *0.0* to *1.0*.
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}
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}

Variable/Constant	Description
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}
color **SKY**	Sky color without alpha. {0,191,255,0,0}
color **GOLD**	Gold color without alpha. {238,201,0,0,0}

Variable/Constant	Description
color **WHITE_ALPHA**	White color with alpha. {255,255,255,255,255}
color **BLACK_ALPHA**	Black color with alpha. {0,0,0,0,255}
Mix Modes	
int **MIXMODE_NORMAL**	The ***Normal*** mix mode.
int **MIXMODE_DARKEN**	The ***Darken*** mix mode.
int **MIXMODE_MULTIPLY**	The ***Multiply*** 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_LINEARDODGE**	The ***Linear Dodge*** mix mode.
int **MIXMODE_OVERLAY**	The ***Overlay*** mix mode.
int **MIXMODE_SOFTLIGHT**	The ***Soft Light*** 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 ***Hard Mix*** mix mode.
int **MIXMODE_DIFFERENCE**	The ***Difference*** mix mode.
int **MIXMODE_EXCLUSION**	The ***Exclusion*** mix mode.
int **MIXMODE_MASK**	The ***Mask*** mix mode.
int **MIXMODE_NEGATIVE_MASK**	The ***Negative Mask*** mix mode.
Filters (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.

Variable/Constant	Description
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_SHARPEN**	This filter sharpens the output.
Brightness Graph Filters	
int **FILTER_BRIGHTNESS_GRAPH_XYZ**	This filter displays the distribution of the color intensity on the y-axis.
int **FILTER_BRIGHTNESS_GRAPH_XZY**	This filter displays the distribution of the color intensity on the z-axis.
int **FILTER_BRIGHTNESS_GRAPH_YXZ**	This filter displays the distribution of the color intensity on the x-axis.
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_GRAYSCALE**	The grayscale filter to render the matrix grayscale, i.e. in gray colors.
int **FILTER_INVERT_COLOR**	The invert color filter to invert every color channel.
Color Swap Filters	
int **FILTER_RGB_TO_BGR**	The filter reorders the color channels to blue, green, red. The white color channel remains unchanged.
int **FILTER_RGB_TO_BRG**	The filter reorders the color channels to blue, red, green. The white color channel remains unchanged.
int **FILTER_RGB_TO_GRB**	The filter reorders the color channels to green, red, blue. The white color channel remains unchanged.
int **FILTER_RGB_TO_GBR**	The filter reorders the color channels to green, blue, red. The white color channel remains unchanged.
int **FILTER_RGB_TO_RBG**	The filter reorders the color channels to red, blue, green. The white color channel remains unchanged.
int **FILTER_RGBW_TO_WBGR**	The filter reorders the color channels to white, blue, green, red.
int **FILTER_RGBW_TO_WBRG**	The filter reorders the color channels to white, blue, red, green.
int **FILTER_RGBW_TO_WGRB**	The filter reorders the color channels to white, green, red, blue.
int **FILTER_RGBW_TO_WGBR**	The filter reorders the color channels to white, green, blue, red.

Variable/Constant	Description
int **FILTER_RGBW_TO_WRBG**	The filter reorders the color channels to white, red, blue, green.
int **FILTER_RGBW_TO_WRGB**	The filter reorders the color channels to white, red, green, blue.
int **FILTER_RGBW_TO_BWGR**	The filter reorders the color channels to blue, white, green, red.
int **FILTER_RGBW_TO_BWRG**	The filter reorders the color channels to blue, white, red, green.
int **FILTER_RGBW_TO_GWRB**	The filter reorders the color channels to green, white, red, blue.
int **FILTER_RGBW_TO_GWBR**	The filter reorders the color channels to green, white, blue, red.
int **FILTER_RGBW_TO_RWBG**	The filter reorders the color channels to red, white, blue, green.
int **FILTER_RGBW_TO_RWGB**	The filter reorders the color channels to red, white, green, blue.
int **FILTER_RGBW_TO_BGWR**	The filter reorders the color channels to blue, green, white, red.
int **FILTER_RGBW_TO_BRWG**	The filter reorders the color channels to blue, red, white, green.
int **FILTER_RGBW_TO_GRWB**	The filter reorders the color channels to green, red, white, blue.
int **FILTER_RGBW_TO_GBWR**	The filter reorders the color channels to green, blue, white, red.
int **FILTER_RGBW_TO_RBWG**	The filter reorders the color channels to red, blue, white, green.
int **FILTER_RGBW_TO_RGWB**	The filter reorders the color channels to red, green, white, blue.
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.
int **FILTER_RED_BLUE**	The red/blue filter to filter out every color except the red and the blue color channel.

Variable/Constant	Description
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.
Kaleidoscope Filters	
int **FILTER_KALEIDOSCOPE_6X**	The ***6x*** mix mode.
int **FILTER_KALEIDOSCOPE_8X**	The ***8x*** mix mode.
int **FILTER_KALEIDOSCOPE_12X**	The ***12x*** mix mode.
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_EMBOSS**	The emboss filter to create an image with just highlights and shadows.
int **FILTER_EMBOSS_POPUP**	The emboss popup filter to create an image with just highlights and shadows depending on the motif.
Mirror 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°.

Variable/Constant	Description
int **FILTER_INVERT_D_MATRIX**	The filter flips the matrix regarding the depth.
int **FILTER_INVERT_HD_MATRIX**	The filter flips the matrix horizontally and regarding the depth.
int **FILTER_INVERT_VD_MATRIX**	The filter flips the matrix vertically and regarding the depth.
int **FILTER_INVERT_HVD_MATRIX**	The filter flips the matrix horizontally, vertically, and regarding the depth.
Swap Filters	
int **FILTER_SWAP_H_1X**	This filter divides the matrix into 2 columns and transposes them.
int **FILTER_SWAP_H_2X**	This filter divides the matrix into 4 columns and transposes them.
int **FILTER_SWAP_H_3X**	This filter divides the matrix into 8 columns and transposes them.
int **FILTER_SWAP_H_4X**	This filter divides the matrix into 16 columns and transposes them.
int **FILTER_SWAP_H_5X**	This filter divides the matrix into 32 columns and transposes them.
int **FILTER_SWAP_V_1X**	This filter divides the matrix into 2 rows and transposes them.
int **FILTER_SWAP_V_2X**	This filter divides the matrix into 4 rows and transposes them.
int **FILTER_SWAP_V_3X**	This filter divides the matrix into 8 rows and transposes them.
int **FILTER_SWAP_V_4X**	This filter divides the matrix into 16 rows and transposes them.
int **FILTER_SWAP_V_5X**	This filter divides the matrix into 32 rows and transposes them.
int **FILTER_SWAP_HV_1X**	This filter divides the matrix into 2 columns and rows, and transposes them.
int **FILTER_SWAP_HV_2X**	This filter divides the matrix into 4 columns and rows, and transposes them.
int **FILTER_SWAP_HV_3X**	This filter divides the matrix into 8 columns and rows, and transposes them.
int **FILTER_SWAP_HV_4X**	This filter divides the matrix into 16 columns and rows, and transposes them.
int **FILTER_SWAP_HV_5X**	This filter divides the matrix into 32 columns and rows, and transposes them.
int **FILTER_SWAP_D_1X**	This filter divides the matrix into 2 depth segments and transposes them.

Variable/Constant	Description
int **FILTER_SWAP_D_2X**	This filter divides the matrix into 4 depth segments and transposes them.
int **FILTER_SWAP_D_3X**	This filter divides the matrix into 8 depth segments and transposes them.
int **FILTER_SWAP_D_4X**	This filter divides the matrix into 16 depth segments and transposes them.
int **FILTER_SWAP_D_5X**	This filter divides the matrix into 32 depth segments and transposes them.
int **FILTER_SWAP_HD_1X**	This filter divides the matrix into 2 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_2X**	This filter divides the matrix into 4 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_3X**	This filter divides the matrix into 8 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_4X**	This filter divides the matrix into 16 columns and depth segments, and transposes them.
int **FILTER_SWAP_HD_5X**	This filter divides the matrix into 32 columns and depth segments, and transposes them.
int **FILTER_SWAP_VD_1X**	This filter divides the matrix into 2 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_2X**	This filter divides the matrix into 4 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_3X**	This filter divides the matrix into 8 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_4X**	This filter divides the matrix into 16 rows and depth segments, and transposes them.
int **FILTER_SWAP_VD_5X**	This filter divides the matrix into 32 rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_1X**	This filter divides the matrix into 2 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_2X**	This filter divides the matrix into 4 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_3X**	This filter divides the matrix into 8 columns, rows and depth segments, and transposes them.

Variable/Constant	Description
int **FILTER_SWAP_HVD_4X**	This filter divides the matrix into 16 columns, rows and depth segments, and transposes them.
int **FILTER_SWAP_HVD_5X**	This filter divides the matrix into 32 columns, rows and depth segments, and transposes them.
Shifting Directions	
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.
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.

Variable/Constant	Description
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_MIRROR_D**	Mirrors the content regarding the depth.
int **MAP_MIRROR_HD**	Mirrors the content of the matrix horizontally and regarding the depth.
int **MAP_MIRROR_VD**	Mirrors the content of the matrix vertically and regarding the depth.
int **MAP_MIRROR_HVD**	Mirrors the content of the matrix horizontally, vertically, and regarding the depth.
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.
int **MAP_TILE_MIRROR_D**	Mirrors tiles regarding the depth.
int **MAP_TILE_MIRROR_HD**	Mirrors tiles horizontally and regarding the depth.
int **MAP_TILE_MIRROR_VD**	Mirrors tiles vertically and regarding the depth.
int **MAP_TILE_MIRROR_HVD**	Mirrors tiles horizontally, vertically, and regarding the depth.
int **MAP_ROTATION_FIXED**	Sets a fixed rotation.
int **MAP_ROTATION_LOOP**	Sets a lopped rotation animation.
int **MAP_AA_NONE**	Sets no anti-aliasing mode.
int **MAP_AA_2X**	Sets simple anti-aliasing (requires some performance).
int **MAP_AA_4X**	Sets complex anti-aliasing (requires a lot performance).
Distributions	
int **DIST_UNIFORM**	Sets the distribution to ***Uniform***.
int **DIST_LINEAR**	Sets the distribution to ***Linear***.
int **DIST_LINEAR_DECREASING**	Sets the distribution to ***Linear Decreasing***.
int **DIST_TRIANGLE**	Sets the distribution to ***Triangle***.
int **DIST_QUADRATIC**	Sets the distribution to ***Quadratic***.

Variable/Constant	Description
int **DIST_QUADRATIC_DECREASING**	Sets the distribution to ***Quadratic Decreasing***.
int **DIST_SQRT**	Sets the distribution to ***Square Root***.
int **DIST_SQRT_DECREASING**	Sets the distribution to ***Square Root Decreasing***.
int **DIST_CUBIC**	Sets the distribution to ***Cubic***.
int **DIST_CUBIC_DECREASING**	Sets the distribution to ***Cubic Decreasing***.
int **DIST_RND**	Sets the distribution to ***Random***.
Wave Types / Function Types	
int **WAVE_TYPE_SINE**	Sets the wave type to ***Sine***.
int **WAVE_TYPE_SINE_ABSOLUTE**	Sets the wave type to ***Absolute Sine***.
int **WAVE_TYPE_SAWTOOTH_DOWNWARDS**	Sets the wave type to ***Sawtooth Downwards***.
int **WAVE_TYPE_SAWTOOTH_UPWARDS**	Sets the wave type to ***Sawtooth Upwards***.
int **WAVE_TYPE_TRIANGLE**	Sets the wave type to ***Triangle***.
int **WAVE_TYPE_COSINE**	Sets the wave type to ***Cosine***.
int **WAVE_TYPE_SQUARE**	Sets the wave type to ***Square***.
Miscellaneous	
string **WHITE_SPACES**	A string which contains white spaces, like tab, new line, or space. This can be used as delimiter string for »Tokenizing
int **TRUE**	Represents value 1.
int **FALSE**	Represents value 0.

Cities	
A - L	**M - Z**
CITY_AACHEN **CITY_AALBORG** **CITY_AARHUS** **CITY_ABERDEEN** **CITY_ABIDJAN** **CITY_ABILENE** **CITY_ABU_DHABI** **CITY_ACAPULCO** **CITY_ACCRA** **CITY_ADANA** **CITY_ADDIS_ABABA** **CITY_ADELAIDE** **CITY_ADEN** **CITY_AGADEZ**	**CITY_MAASTRICHT** **CITY_MACAO** **CITY_MACHALA** **CITY_MACHAZE** **CITY_MACON** **CITY_MADISON** **CITY_MADRAS** **CITY_MADRID** **CITY_MAGDEBURG** **CITY_MAINZ** **CITY_MAJUNGA** **CITY_MALABO** **CITY_MALAGA** **CITY_MALANG**

Cities	
CITY_AGADIR	CITY_MALDEN_BRIDGE
CITY_AHMEDABAD	CITY_MALE
CITY_AKRON	CITY_MALMOE
CITY_AKUREYRI	CITY_MANAGUA
CITY_AKYAB	CITY_MANAMA
CITY_ALBANY	CITY_MANAUS
CITY_ALBANY_NEW_YORK_USA	CITY_MANCHESTER
CITY_ALBUQUERQUE	CITY_MANCHESTER_NEW_HAMPSHIRE_USA
CITY_ALEPPO	CITY_MANDALAY
CITY_ALEXANDRIA	CITY_MANILA
CITY_ALEXANDRIA_LOUISIANA_USA	CITY_MANIZALES
CITY_ALEXANDRIA_VIRGINIA_USA	CITY_MANNHEIM
CITY_ALGIERS	CITY_MANTA
CITY_ALLENTOWN	CITY_MAPUTO
CITY_ALTOONA	CITY_MARACAIBO
CITY_AL_MAHALLA_AL_KUBRA	CITY_MARACAY
CITY_AL_MOBARRAZ	CITY_MARADI
CITY_AMADORA	CITY_MAROUA
CITY_AMARILLO	CITY_MARRAKESH
CITY_AMBATO	CITY_MARSEILLE
CITY_AMMAN	CITY_MAR_DEL_PLATA
CITY_AMSTERDAM	CITY_MASERU
CITY_AMSTETTEN	CITY_MASHHAD
CITY_ANAHEIM	CITY_MAZAR_I_SHARIF
CITY_ANCHORAGE	CITY_MBABANE
CITY_ANDORRA_LA_VELLA	CITY_MBALE
CITY_ANGELES	CITY_MBUJI_MAYI
CITY_ANKARA	CITY_MECCA
CITY_ANNABA	CITY_MECHELEN
CITY_ANNAPOLIS	CITY_MEDAN
CITY_ANN_ARBOR	CITY_MEDELLIN
CITY_ANTANANARIVO	CITY_MEDINA
CITY_ANTOFAGASTA	CITY_MEKNES
CITY_ANTSIRABE	CITY_MELBOURNE
CITY_ANTSIRANANA	CITY_MELBOURNE_FLORIDA_USA
CITY_ANTWERP	CITY_MEMPHIS
CITY_APELDOORN	CITY_MENDOZA
CITY_APIA	CITY_MERIDA
CITY_AREQUIPA	CITY_MERIDIAN
CITY_ARIANA	CITY_MEXICALI
CITY_ARICA	CITY_MEXICO_CITY
CITY_ARLINGTON	CITY_MIAMI
CITY_ARUSHA	CITY_MIDDLETOWN
CITY_ASHEVILLE	CITY_MIDLAND
CITY_ASMARA	CITY_MILAN
CITY_ASSIUT	CITY_MILWAUKEE
CITY_ASUNCION	CITY_MINDELO
CITY_AS_SMARA	CITY_MINNEAPOLIS
CITY_ATBARA	CITY_MINSK
CITY_ATHENS	CITY_MISKOLC
CITY_ATLANTA	CITY_MOBILE
CITY_ATLANTIC_CITY	CITY_MOENCHENGLADBACH
CITY_AUCKLAND	

Cities	
CITY_AUGSBURG	CITY_MOERS
CITY_AUGUSTA	CITY_MOGADISHU
CITY_AUSTIN	CITY_MOMBASA
CITY_BACOLOD	CITY_MONACO
CITY_BADEN	CITY_MONROE
CITY_BAGHDAD	CITY_MONROVIA
CITY_BAHIA_BLANCA	CITY_MONS
CITY_BAKERSFIELD	CITY_MONTEGO_BAY
CITY_BAKHTARAN	CITY_MONTERREY
CITY_BAKU	CITY_MONTEVIDEO
CITY_BALTIMORE	CITY_MONTE_CARLO
CITY_BAMAKO	CITY_MONTGOMERY
CITY_BANDAR_SERI_BEGAWAN	CITY_MONTPELLIER
CITY_BANDUNG	CITY_MONTREAL
CITY_BANGALORE	CITY_MOPTI
CITY_BANGKOK	CITY_MORATUWA
CITY_BANGOR	CITY_MORONI
CITY_BANGUI	CITY_MOSCOW
CITY_BANJUL	CITY_MOSUL
CITY_BARCELONA	CITY_MOULMEIN
CITY_BARCELONA_VENEZUELA	CITY_MOUNT_VERNON
CITY_BARI	CITY_MUELHEIM
CITY_BARQUISIMETO	CITY_MUKALLA
CITY_BARRANQUILLA	CITY_MULTAN
CITY_BARREIRO	CITY_MUNCIE
CITY_BASEL	CITY_MUNICH
CITY_BASRA	CITY_MUNSTER
CITY_BASSEIN	CITY_MUSCAT
CITY_BASSE_TERRE	CITY_MWANZA
CITY_BATA	CITY_NACALA
CITY_BATON_ROUGE	CITY_NAGASAKI
CITY_BEAUMONT	CITY_NAGOYA
CITY_BEERSHEBA	CITY_NAGPUR
CITY_BEIJING	CITY_NAIROBI
CITY_BEIRA	CITY_NAJAF
CITY_BEIRUT	CITY_NAKHON_RATHISAMA
CITY_BELEM	CITY_NAKHON_SI_THAMMARAT
CITY_BELFAST	CITY_NAKURU
CITY_BELGRADE	CITY_NAMUR
CITY_BELIZE_CITY	CITY_NANJING
CITY_BELMOPAN	CITY_NANTES
CITY_BELO_HORIZONTE	CITY_NAPLES
CITY_BENGHAZI	CITY_NASHUA
CITY_BERBERA	CITY_NASHVILLE
CITY_BERGEN	CITY_NASSAU
CITY_BERGISCH_GLADBACH	CITY_NATCHEZ
CITY_BERLIN	CITY_NATITINGOU
CITY_BERN	CITY_NDJAMENA
CITY_BETHLEHEM	CITY_NDOLA
CITY_BIEL	CITY_NEE_SOON
CITY_BIELEFELD	CITY_NEUCHATEL
CITY_BILBAO	CITY_NEUSS
CITY_BILE	CITY_NEWARK

Cities	
CITY_BILLINGS	CITY_NEWCASTLE
CITY_BILOXI	CITY_NEWPORT
CITY_BINGHAMTON	CITY_NEWPORT_NEWS
CITY_BIRMINGHAM	CITY_NEW_BEDFORD
CITY_BIRMINGHAM_UNITED_KINGDOM	CITY_NEW_DELHI
CITY_BISMARCK	CITY_NEW_HAVEN
CITY_BISSAU	CITY_NEW_ORLEANS
CITY_BIZERTA	CITY_NEW_YORK
CITY_BLANTYRE	CITY_NHA_TRANG
CITY_BLIDA	CITY_NIAGARA_FALLS
CITY_BNEI_BRAK	CITY_NIAMEY
CITY_BO	CITY_NICE
CITY_BOBO_DIOULASSO	CITY_NICOSIA
CITY_BOCHUM	CITY_NIJMEGEN
CITY_BOGATA	CITY_NISH
CITY_BOISE	CITY_NIZHNY_NOVGOROD
CITY_BOLOGNA	CITY_NORFOLK
CITY_BOLZANO	CITY_NORRKOEPING
CITY_BOMBAY	CITY_NORWICH
CITY_BONN	CITY_NOUADHIBOU
CITY_BORAS	CITY_NOUAKCHOTT
CITY_BORDEAUX	CITY_NOVA_IGUACU
CITY_BOSTON	CITY_NOVI_SAD
CITY_BOTTROP	CITY_NOVOSIBIRSK
CITY_BOULDER	CITY_NUKUALOFA
CITY_BOULOGNE	CITY_NUREMBERG
CITY_BRADFORD	CITY_NZEREKORE
CITY_BRAGA	CITY_OAKLAND
CITY_BRAILA	CITY_OBERHAUSEN
CITY_BRASILIA	CITY_ODENSE
CITY_BRATISLAVA	CITY_ODESSA
CITY_BRAUNAU	CITY_ODESSA_TEXAS_USA
CITY_BRAUNSCHWEIG	CITY_OFFENBACH
CITY_BRAZZAVILLE	CITY_OGBOMOSHO
CITY_BREGENZ	CITY_OGDEN
CITY_BREMEN	CITY_OKLAHOMA_CITY
CITY_BREMERHAVEN	CITY_OLDENBURG
CITY_BRESLAU	CITY_OLOMOUC
CITY_BRIDGEPORT	CITY_OMAHA
CITY_BRIDGETOWN	CITY_OMDURMAN
CITY_BRISBANE	CITY_OMSK
CITY_BRISTOL	CITY_ORAN
CITY_BRNO	CITY_ORANGE_WALK
CITY_BROMBERG	CITY_ORANJESTAD
CITY_BROWNSVILLE	CITY_OREBRO
CITY_BRUCK_AN_DER_MUR	CITY_ORLANDO
CITY_BRUEGGE	CITY_ORURO
CITY_BRUSSELS	CITY_OSAKA
CITY_BUAK	CITY_OSHKOSH
CITY_BUCARAMANGA	CITY_OSIJEK
CITY_BUCHAREST	CITY_OSLO
CITY_BUDAPEST	CITY_OSNABRUECK
CITY_BUENOS_AIRES	CITY_OSTEND

Cities	
CITY_BUFFALO	CITY_OSTRAVA
CITY_BUJUMBURA	CITY_OTTAWA
CITY_BUKAVU	CITY_OUAGADOUGOU
CITY_BULAWAYO	CITY_OUJDA
CITY_BURAIDAH	CITY_OULU
CITY_BURGAS	CITY_OWENSBORO
CITY_BURLINGTON	CITY_OXFORD
CITY_BURSA	CITY_PADANG
CITY_BUTARE	CITY_PADERBORN
CITY_BUTTE	CITY_PAKSE
CITY_CAGAYAN_DE_ORO	CITY_PALEMBANG
CITY_CAIRO	CITY_PALERMO
CITY_CALCUTTA	CITY_PALMA_DE_MALLORCA
CITY_CALGARY	CITY_PALM_SPRINGS
CITY_CALI	CITY_PANAMA_CITY
CITY_CALLAO	CITY_PANCHIAO
CITY_CALOOCAN	CITY_PAPEETE
CITY_CAMAGUEY	CITY_PARAMARIBO
CITY_CAMDEN	CITY_PARIS
CITY_CAM_RANH	CITY_PASADENA
CITY_CANBERRA	CITY_PASAY
CITY_CANTON	CITY_PATERSON
CITY_CAPE_TOWN	CITY_PATRAS
CITY_CAP_HAITIEN	CITY_PAU
CITY_CARACAS	CITY_PAYSANDU
CITY_CARDIFF	CITY_PECS
CITY_CARTAGENA	CITY_PEDRO_JUAN_CABALLERO
CITY_CASABLANCA	CITY_PENSACOLA
CITY_CASPER	CITY_PEORIA
CITY_CASTRIES	CITY_PEREIRA
CITY_CATANIA	CITY_PERM
CITY_CEBU	CITY_PETACH_TIKVA
CITY_CEDAR_RAPIDS	CITY_PETANGE
CITY_CHAMPAIGN	CITY_PETERSBURG
CITY_CHANIA	CITY_PFORZHEIM
CITY_CHARLEROI	CITY_PHILADELPHIA
CITY_CHARLESTON	CITY_PHNOM_PENH
CITY_CHARLESTON_WEST_VIRGINIA_USA	CITY_PHOENIX
CITY_CHARLOTTE	CITY_PINANG
CITY_CHARLOTTESVILLE	CITY_PINE_BLUFF
CITY_CHATTANOOGA	CITY_PIRAEUS
CITY_CHEMNITZ	CITY_PITTSBURGH
CITY_CHENGDU	CITY_PITTSFIELD
CITY_CHEYENNE	CITY_PIURA
CITY_CHIANG_MAI	CITY_PLOVDIV
CITY_CHIAYI	CITY_PLYMOUTH
CITY_CHICAGO	CITY_PLZEN
CITY_CHICLAYO	CITY_POCATELLO
CITY_CHIHUAHUA	CITY_POINTE_NOIRE
CITY_CHILLAN	CITY_POONA
CITY_CHIMBOTE	CITY_PORI
CITY_CHINGOLA	CITY_PORTLAND
CITY_CHITTAGONG	CITY_PORTLAND_OREGON_USA

Cities	
CITY_CHITUNGWIZA	CITY_PORTO
CITY_CHOLUTECA	CITY_PORTOVIEJO
CITY_CHONGJIN	CITY_PORTO_ALEGRE
CITY_CHONGQING	CITY_PORTO_NOVO
CITY_CHON_BURI	CITY_PORTSMOUTH
CITY_CHOYBALSAN	CITY_PORT_ARTHUR
CITY_CHRISTCHURCH	CITY_PORT_AU_PRINCE
CITY_CHUNGLI	CITY_PORT_ELIZABETH
CITY_CHUR	CITY_PORT_GENTIL
CITY_CINCINNATI	CITY_PORT_LOUIS
CITY_CIRCLEVILLE	CITY_PORT_MORESBY
CITY_CIUDAD_GUAYANA	CITY_PORT_OF_SPAIN
CITY_CIUDAD_JUAREZ	CITY_PORT_SAID
CITY_CLARKSVILLE	CITY_PORT_SUDAN
CITY_CLEVELAND	CITY_PORT_VILA
CITY_CLUJ_NAPOCA	CITY_POSES
CITY_COCHABAMBA	CITY_POTOSI
CITY_COIMBRA	CITY_POTSDAM
CITY_COLOGNE	CITY_POUGHKEEPSIE
CITY_COLOMBO	CITY_PRAGUE
CITY_COLON	CITY_PRAIA
CITY_COLORADO_SPRINGS	CITY_PRETORIA
CITY_COLUMBIA	CITY_PROVIDENCE
CITY_COLUMBIA_SOUTH_CAROLINA_USA	CITY_PROVO
CITY_COLUMBUS	CITY_PUEBLA
CITY_COLUMBUS_OHIO_USA	CITY_PUEBLA_DE_ZARAGOZA
CITY_CONAKRY	CITY_PUEBLO
CITY_CONCEPCION	CITY_PUERTO_BARRIOS
CITY_CONCEPCION_PARAGUAY	CITY_PUERTO_PLATA
CITY_CONCORD	CITY_PUERTO_STROESSNER
CITY_CONSTANTINE	CITY_PUNTARENAS
CITY_CONSTANZA	CITY_PUSAN
CITY_COPENHAGEN	CITY_PYONGYANG
CITY_CORDOBA	CITY_PYUTHAN
CITY_CORK	CITY_QUEBEC
CITY_COROZAL	CITY_QUEZALTENANGO
CITY_CORPUS_CHRISTI	CITY_QUEZON_CITY
CITY_COTONOU	CITY_QUITO
CITY_COTTBUS	CITY_QUI_NHON
CITY_CRAIOVA	CITY_RABAT
CITY_CUCUTA	CITY_RABAUL
CITY_CUENCA	CITY_RALEIGH
CITY_CURITIBA	CITY_RAMAT_GAN
CITY_CUZCO	CITY_RANCAGUA
CITY_DAKAR	CITY_RANDERS
CITY_DAKHLA	CITY_RANGOON
CITY_DALLAS	CITY_RAPID_CITY
CITY_DALOA	CITY_RAQQA
CITY_DAMASCUS	CITY_RAWALPINDI
CITY_DANBURY	CITY_RECIFE
CITY_DANGRIGA	CITY_RECKLINGHAUSEN
CITY_DANZIG	CITY_REGENSBURG
CITY_DARKHAN	CITY_REGINA

Cities	
CITY_DARMSTADT	CITY_REMICH
CITY_DAR_ES_SALAAM	CITY_REMSCHEID
CITY_DAVAO	CITY_RENO
CITY_DAVID	CITY_REYKJAVIK
CITY_DAYTON	CITY_RICHMOND
CITY_DAYTONA_BEACH	CITY_RIGA
CITY_DA_NANG	CITY_RIO_DE_JANEIRO
CITY_DEBRECEN	CITY_RIVERA
CITY_DEHIWALA	CITY_RIVERSIDE
CITY_DEIR_EZ_ZOR	CITY_RIYADH
CITY_DENVER	CITY_ROANOKE
CITY_DESSAU	CITY_ROCHESTER
CITY_DES_MOINES	CITY_ROCHESTER_NEW_YORK_USA
CITY_DETROIT	CITY_ROCKFORD
CITY_DHAKA	CITY_ROCK_ISLAND
CITY_DIFFERDANGE	CITY_ROME
CITY_DJERBA	CITY_ROODEPOORT
CITY_DJIBOUTI	CITY_ROSARIO
CITY_DODOMA	CITY_ROSEAU
CITY_DOHA	CITY_ROSKILDE
CITY_DONETSK	CITY_ROSTOCK
CITY_DORNBIRN	CITY_ROSTOV
CITY_DORTMUND	CITY_ROTTERDAM
CITY_DOUALA	CITY_RUSE
CITY_DRAMMEN	CITY_SAARBRUECKEN
CITY_DRESDEN	CITY_SACRAMENTO
CITY_DUBAI	CITY_SAFI
CITY_DUBLIN	CITY_SAGINAW
CITY_DUDELANGE	CITY_SALEM
CITY_DUESSELDORF	CITY_SALEM_OREGON_USA
CITY_DUISBURG	CITY_SALINAS
CITY_DULUTH	CITY_SALIYAN
CITY_DUNEDIN	CITY_SALTA
CITY_DURBAN	CITY_SALTO
CITY_DURHAM	CITY_SALT_LAKE_CITY
CITY_DURRES	CITY_SALVADOR
CITY_EDINBURGH	CITY_SALZBURG
CITY_EDMONTON	CITY_SALZGITTER
CITY_EERDENET	CITY_SANA
CITY_EINDHOVEN	CITY_SANKT_POELTEN
CITY_EISENSTADT	CITY_SANTA_ANA
CITY_ELIZABETH	CITY_SANTA_BARBARA
CITY_ELMIRA	CITY_SANTA_CLARA
CITY_ELOBIED	CITY_SANTA_CRUZ
CITY_ELSINORE	CITY_SANTA_FE
CITY_EL_ALAIUN	CITY_SANTA_FE_NEW_MEXICO_USA
CITY_EL_MANSOURA	CITY_SANTA_ROSA
CITY_EL_PASO	CITY_SANTIAGO
CITY_ENCARNACION	CITY_SANTIAGO_DE_CHILE
CITY_ENSCHEDE	CITY_SANTIAGO_DE_CUBA
CITY_ENTEBBE	CITY_SANTIAGO_PANAMA
CITY_ERFURT	CITY_SANTO_DOMINGO
CITY_ERIE	CITY_SAN_ANTONIO

Cities	
CITY_ESBJERG	CITY_SAN_BERNARDINO
CITY_ESCH	CITY_SAN_CRISTOBAL
CITY_ESCUINTLA	CITY_SAN_DIEGO
CITY_ESMERALDAS	CITY_SAN_FRANCISCO
CITY_ESPOO	CITY_SAN_JOSE
CITY_ESSEN	CITY_SAN_JOSE_COSTA_RICA
CITY_EUGENE	CITY_SAN_JUAN
CITY_EVANSVILLE	CITY_SAN_JUAN_DOMINICAN_REPUBLIC
CITY_FAIRBANKS	CITY_SAN_JUAN_PUERTO_RICO
CITY_FAISALABAD	CITY_SAN_LORENZO
CITY_FARGO	CITY_SAN_LUIS_POTOSI
CITY_FAYETTEVILLE	CITY_SAN_MARINO
CITY_FELDKIRCH	CITY_SAN_MIGUEL
CITY_FERNANDO_DE_LA_MORA	CITY_SAN_MIGUEL_DE_TUCUMAN
CITY_FEZ	CITY_SAN_PEDRO_SULA
CITY_FIANAR	CITY_SAN_SALVADOR
CITY_FLINT	CITY_SAO_PAULO
CITY_FLORENCE	CITY_SAO_TOME
CITY_FORTALEZA	CITY_SAPPORO
CITY_FORT_DE_FRANCE	CITY_SARAJEVO
CITY_FORT_LAUDERDALE	CITY_SARH
CITY_FORT_LAUDERDALE_FLORIDA_USA	CITY_SAVANNAH
CITY_FORT_SMITH	CITY_SAVANNAKHET
CITY_FORT_SMITH_ARKANSAS_USA	CITY_SCHAFFHAUSEN
CITY_FORT_WAYNE	CITY_SCHENECTADY
CITY_FORT_WAYNE_INDIANA_USA	CITY_SCHWERIN
CITY_FORT_WORTH	CITY_SCRANTON
CITY_FORT_WORTH_TEXAS_USA	CITY_SEATTLE
CITY_FRANCISTOWN	CITY_SEGOU
CITY_FRANKFURT	CITY_SEMARANG
CITY_FREEPORT	CITY_SEOUL
CITY_FREETOWN	CITY_SERANGOON
CITY_FREIBURG	CITY_SERKA
CITY_FREISING	CITY_SERRES
CITY_FRESNO	CITY_SETIF
CITY_FRIBOURG	CITY_SETUBAL
CITY_FUKUOKA	CITY_SEVILLE
CITY_FUNAFUTI	CITY_SFAX
CITY_FUNCHAL	CITY_SHANGHAI
CITY_GABORONE	CITY_SHARJAH
CITY_GAIN	CITY_SHEFFIELD
CITY_GAINESVILLE	CITY_SHENYANG
CITY_GALATZ	CITY_SHIRAZ
CITY_GALVESTON	CITY_SHKODER
CITY_GALWAY	CITY_SHREVEPORT
CITY_GAO	CITY_SHUBRA_AL_KHAIMA
CITY_GARLAND	CITY_SIDI_BEL_ABBES
CITY_GARY	CITY_SIEGEN
CITY_GASIANTEP	CITY_SIGUIRI
CITY_GEELONG	CITY_SIKASSO
CITY_GELSENKIRCHEN	CITY_SINGAPORE
CITY_GENEVE	CITY_SIOUX_CITY
CITY_GENOA	

Cities	
CITY_GEORGETOWN	CITY_SIOUX_FALLS
CITY_GERA	CITY_SKOPJE
CITY_GERMISTON	CITY_SLIEMA
CITY_GHENT	CITY_SOFIA
CITY_GIBRALTAR	CITY_SOLINGEN
CITY_GIZA	CITY_SONGKHLA
CITY_GLASGOW	CITY_SOSNOWIEC
CITY_GOETEBORG	CITY_SOUTHAMPTON
CITY_GOETTINGEN	CITY_SOUTH_BEND
CITY_GOLD_COAST	CITY_SPARTANBURG
CITY_GOTHAM_CITY	CITY_SPLIT
CITY_GRANADA	CITY_SPOKANE
CITY_GRAND_FORKS	CITY_SPRINGFIELD
CITY_GRAND_RAPIDS	CITY_SPRINGFIELD_MASSACHUSETTS_USA
CITY_GRAZ	CITY_SPRINGFIELD_MISSOURI_USA
CITY_GREAT_FALLS	CITY_STAMFORD
CITY_GREENSBORO	CITY_STARA_ZAGORA
CITY_GREENVILLE	CITY_STAVANGER
CITY_GREEN_BAY	CITY_STETTIN
CITY_GRONINGEN	CITY_STEYR
CITY_GUADALAJARA	CITY_STOCKHOLM
CITY_GUANGZHOU	CITY_STOCKTON
CITY_GUANTANAMO	CITY_STRASBOURG
CITY_GUATEMALA_CITY	CITY_STUTTGART
CITY_GUAYAQUIL	CITY_ST_AUGUSTINE
CITY_GUJRANWALA	CITY_ST_CATHARINES
CITY_GULF_PORT	CITY_ST_ETIENNE
CITY_GWERU	CITY_ST_GALLEN
CITY_GYOER	CITY_ST_GEORGES
CITY_HAARLEM	CITY_ST_GEORGES_GRENADA
CITY_HAGEN	CITY_ST_JOHNS
CITY_HAIFA	CITY_ST_JOHNS_ANTIGUA
CITY_HAIPHONG	CITY_ST_JOSEPH
CITY_HALLE	CITY_ST_LAURENT
CITY_HALLEIN	CITY_ST_LOUIS
CITY_HAMA	CITY_ST_LOUIS_SENEGAL
CITY_HAMBURG	CITY_ST_PAUL
CITY_HAMILTON	CITY_ST_PETERSBURG
CITY_HAMILTON_CANADA	CITY_ST_PETERSBURG_FLORIDA_USA
CITY_HAMILTON_NEW_ZEALAND	CITY_SUCRE
CITY_HAMM	CITY_SUDBURY
CITY_HAMPTON	CITY_SUEZ
CITY_HANOI	CITY_SUNNYVALE
CITY_HANOVER	CITY_SURABAYA
CITY_HARARE	CITY_SURAKARTA
CITY_HARBIN	CITY_SUVA
CITY_HARGEISA	CITY_SVERDLOVSK
CITY_HARRISBURG	CITY_SYDNEY
CITY_HARTFORD	CITY_SYRACUSE
CITY_HAVANA	CITY_SZEGED
CITY_HEIDELBERG	CITY_TABOUK
CITY_HEILBRONN	CITY_TABRIZ
CITY_HELENA	

Cities	
CITY_HELSINGBORG	CITY_TACOMA
CITY_HELSINKI	CITY_TAEGU
CITY_HERAT	CITY_TAEJEON
CITY_HERNE	CITY_TAHOUA
CITY_HIGH_POINT	CITY_TAICHUNG
CITY_HILO	CITY_TAIF
CITY_HIROSHIMA	CITY_TAINAN
CITY_HOBART	CITY_TAIPEI
CITY_HODEIDA	CITY_TAIS
CITY_HOFUF	CITY_TALCA
CITY_HOLGUIN	CITY_TALCAHUANO
CITY_HOLON	CITY_TALLAHASSEE
CITY_HOMS	CITY_TAMALE
CITY_HONG_KONG	CITY_TAMPA
CITY_HONIARA	CITY_TAMPERE
CITY_HONOLULU	CITY_TAMPICO
CITY_HORSENS	CITY_TANGA
CITY_HOSPITALET	CITY_TANGIER
CITY_HOUSTON	CITY_TANTA
CITY_HO_CHI_MINH_CITY	CITY_TARAWA
CITY_HSINCHU	CITY_TASHKENT
CITY_HUAMBO	CITY_TAUNGGYI
CITY_HUNGNAM	CITY_TBILISI
CITY_HUNTINGTON	CITY_TEGUCIGALPA
CITY_HUNTSVILLE	CITY_TEHRAN
CITY_HYDERABAD	CITY_TEL_AVIV
CITY_HYDERABAD_PAKISTAN	CITY_TEMPE
CITY_IASI	CITY_TEMUCO
CITY_IBADAN	CITY_TERNITZ
CITY_IBAGUE	CITY_TERRE_HAUTE
CITY_IDAHO_FALLS	CITY_TETOUAN
CITY_ILOILO	CITY_THESSALONIKI
CITY_INCHON	CITY_THE_HAGUE
CITY_INDEPENDENCE	CITY_THIES
CITY_INDIANAPOLIS	CITY_THIMPHU
CITY_INNSBRUCK	CITY_THUN
CITY_IPOH	CITY_TIANJIN
CITY_IQUITOS	CITY_TIJUANA
CITY_IRAKLION	CITY_TILBURG
CITY_IRBID	CITY_TIMBUKTU
CITY_IRKUTSK	CITY_TIMISOARA
CITY_ISFAHAN	CITY_TIRANA
CITY_ISLAMABAD	CITY_TLEMCEN
CITY_ISTANBUL	CITY_TOAMASINA
CITY_IZMIR	CITY_TOKYO
CITY_JACKSON	CITY_TOLEDO
CITY_JACKSONVILLE	CITY_TOLIARY
CITY_JAFFNA	CITY_TOPEKA
CITY_JAIPUR	CITY_TORONTO
CITY_JAKARTA	CITY_TORREON
CITY_JALALABAD	CITY_TOULOUSE
CITY_JEDDAH	CITY_TOURS
CITY_JEFFERSON_CITY	CITY_TRAUN

Cities	
CITY_JENA	CITY_TRENTON
CITY_JERSEY_CITY	CITY_TRIPOLI
CITY_JERUSALEM	CITY_TRIPOLI_LIBYA
CITY_JINJA	CITY_TRONDHEIM
CITY_JOENKOEPING	CITY_TROY
CITY_JOHANNESBURG	CITY_TRUJILLO
CITY_JOHNSTOWN	CITY_TSCHOUDJO
CITY_JOHORE_BHARU	CITY_TSUMEB
CITY_JUMLA	CITY_TUCSON
CITY_JURONG	CITY_TULSA
CITY_KABUL	CITY_TUNIS
CITY_KAESONG	CITY_TURIN
CITY_KAISERSLAUTERN	CITY_TURKU
CITY_KALAMAZOO	CITY_TUSCALOOSA
CITY_KAMPALA	CITY_UJUNG_PADANG
CITY_KANANGA	CITY_ULAN_BATOR
CITY_KANDAHAR	CITY_ULM
CITY_KANDY	CITY_UMHLAZI
CITY_KANKAN	CITY_UPPSALA
CITY_KANO	CITY_UTICA
CITY_KANPUR	CITY_UTRECHT
CITY_KANSAS_CITY	CITY_VADUZ
CITY_KANSAS_CITY_MISSOURI_USA	CITY_VAESTERAS
CITY_KAOLACK	CITY_VALENCIA
CITY_KAPFENBERG	CITY_VALENCIA_VENEZUELA
CITY_KARACHI	CITY_VALLADOLID
CITY_KARLSRUHE	CITY_VALLETTA
CITY_KARONGA	CITY_VALPARAISO
CITY_KASSEL	CITY_VANCOUVER
CITY_KATMANDU	CITY_VANTAA
CITY_KATOWICE	CITY_VARNA
CITY_KAVALA	CITY_VEJLE
CITY_KAWASAKI	CITY_VENICE
CITY_KAYES	CITY_VERA_CRUZ
CITY_KAZAN	CITY_VICTORIA
CITY_KEELUNG	CITY_VICTORIA_CANADA
CITY_KEETMANSHOOP	CITY_VICTORIA_SEYCHELLES
CITY_KENITRA	CITY_VIENNA
CITY_KEY_WEST	CITY_VIENTIANE
CITY_KHAMIS_MUSHAIT	CITY_VILA_NOVA_DE_GAIA
CITY_KHAOSIUNG	CITY_VILLACH
CITY_KHARKOV	CITY_VINA_DEL_MAR
CITY_KHARTOUM	CITY_VINELAND
CITY_KIEL	CITY_VLADIVOSTOK
CITY_KIEV	CITY_VOLGOGRAD
CITY_KIGALI	CITY_VOLOS
CITY_KINGSTON	CITY_WACO
CITY_KINGSTOWN	CITY_WADI_MEDANI
CITY_KINSHASA	CITY_WARSAW
CITY_KIRKUK	CITY_WARWICK
CITY_KISANGANI	CITY_WASHINGTON
CITY_KISMAYU	CITY_WATERBURY
CITY_KISUMU	CITY_WATERFORD

Cities	
CITY_KITAKIUSHU CITY_KITCHENER CITY_KITWE CITY_KLAGENFURT CITY_KLOSTERNEUBURG CITY_KNOXVILLE CITY_KOBE CITY_KOBLENZ CITY_KOENIZ CITY_KOLDING CITY_KORTRIJK CITY_KOSICE CITY_KOTA_BHARU CITY_KOWLOON CITY_KRAKOW CITY_KREFELD CITY_KREMS CITY_KRISTIANSAND CITY_KRONSTADT CITY_KUALA_LUMPUR CITY_KUALA_TRENGGANU CITY_KUIBYSHEV CITY_KUMASI CITY_KUOPIO CITY_KUWAIT_CITY CITY_KWANGJU CITY_KYOTO CITY_LABE CITY_LAE CITY_LAGOS CITY_LAHORE CITY_LAHTI CITY_LANCASTER CITY_LANSING CITY_LAREDO CITY_LARISSA CITY_LAS_CRUCES CITY_LAS_PALMAS CITY_LAS_PIEDRAS CITY_LAS_VEGAS CITY_LATAKIA CITY_LAUSANNE CITY_LAWRENCE CITY_LAWRENCE_MASSACHUSETTS_USA CITY_LA_CEIBA CITY_LA_CHAUX_DE_FONDS CITY_LA_CHORRERA CITY_LA_CROSSE CITY_LA_PAZ CITY_LA_PLATA CITY_LA_ROMANA CITY_LEEDS CITY_LEIPZIG	CITY_WATERLOO CITY_WELLINGTON CITY_WELS CITY_WEST_PALM_BEACH CITY_WHEELING CITY_WICHITA CITY_WICHITA_FALLS CITY_WIENER_NEUSTADT CITY_WIESBADEN CITY_WILKES_BARRE CITY_WILLEMSTAD CITY_WILLIAMSBURG CITY_WILLISTON CITY_WILMINGTON CITY_WILMINGTON_NORTH_CAROLINA_USA CITY_WINDHOEK CITY_WINDSOR CITY_WINNIPEG CITY_WINSTON_SALEM CITY_WINTERTHUR CITY_WITTEN CITY_WOLFSBURG CITY_WOLF_MOUNTAIN CITY_WOLLONGONG CITY_WORCESTER CITY_WUERZBURG CITY_WUHAN CITY_WUPPERTAL CITY_XIAN CITY_YAKIMA CITY_YAOUNDE CITY_YAREN CITY_YEREVAN CITY_YOKOHAMA CITY_YORK CITY_YORK_PENNSYLVANIA_USA CITY_YOUNGSTOWN CITY_YUNGHU CITY_ZAGAZIG CITY_ZAGREB CITY_ZAMBOANGA CITY_ZANZIBAR CITY_ZARAGOZA CITY_ZIGUINCHOR CITY_ZINDER CITY_ZOMBA CITY_ZURICH CITY_ZWICKAU

Cities	
CITY_LEOBEN CITY_LEON CITY_LEVERKUSEN CITY_LEWISTON CITY_LEXINGTON CITY_LE_HAVRE CITY_LIBEREC CITY_LIBREVILLE CITY_LIEGE CITY_LILONGWE CITY_LIMA CITY_LIMASSOL CITY_LIMERICK CITY_LIMON CITY_LINCOLN CITY_LINDEN CITY_LINKOEPING CITY_LINZ CITY_LISBON CITY_LITTLE_ROCK CITY_LIVERPOOL CITY_LJUBLJANA CITY_LODZ CITY_LOME CITY_LONDON CITY_LONG_BEACH CITY_LOS_ANGELES CITY_LOUISVILLE CITY_LOWELL CITY_LUANDA CITY_LUANG_PRABANG CITY_LUBANGO CITY_LUBBOCK CITY_LUBLIN CITY_LUBUMBASHI CITY_LUCERNE CITY_LUDWIGSHAFEN CITY_LUEBECK CITY_LUSAKA CITY_LUXEMBOURG CITY_LYNCHBURG CITY_LYON	

List Of Extra Script Information

Identifier	Description
`@scriptname`	A name for the script. »Description
`@author`	The name of the author who wrote the script. »Description
`@version`	The current version of this script; if there is any. »Description
`@description`	Any text that describes the script. »Description

3.5 List Of Operations

Arithmetical Operations

Operator	Operand/ Data Type	Results In Data Type	Description
++, -- (as prefix, e.g. ++i)	int	int	Adds/subtracts 1 to/from the value of the operand. Superior expressions are evaluated after the value of the operand is changed.
++, -- (as suffix, e.g. i++)	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, string	Depends on operand data types. It will be converted into the more precise data type.	Calculates the sum of two operands or concatenates two character strings.
-, *, /	int, float	Depends on operand data types. It will be converted into the more precise data type.	Calculates the difference/product/quotient of two operands.
%	int	int	Calculates the remainder of an integer division.

»Description

Logical Operations

Operator	Operand/ Data Type	Results In Data Type	Description
!	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.
<, <=, >, >=, ==, ! =	int, float, bool, string	bool	Compares two operands.

»Description

Bit Operations

Operator	Operand/ Data Type	Results In Data Type	Description
~	int	int	This is the bitwise NOT. It negates each bit of the following operand.
^	int	int	This is the bitwise XOR. Each bit of the result is set where the corresponding bits of the operands are different.
|	int	int	This is the bitwise OR. Each bit of the result is set where at least one of the corresponding bits of the operands is set.
&	int	int	This is the bitwise AND. Each bit of the result is set where both corresponding bits of the operands are set.
<<	int	int	Shifts the bits of the first operand by offset *n* to the left (*n* is the second operand). The rightmost (least significant) bits of the result are set to 0.
>>	int	int	Shifts the bits of the first operand by offset *n* to the right (*n* is the second operand). The leftmost (most significant) bits of the result are set to 0.
>>>	int	int	Shifts the bits of the first operand by offset *n* to the right (*n* is the second operand). The leftmost (most significant) bits of the result depend on the first operand which sign is preserved (i.e. negative numbers will not become positive by shifting to the right).

»Description

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**.
^=	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**.
>>>=	For example: **i >>>= 4** corresponds to **i = i >>> 4**.

»Description

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	▪ int r ▪ int g ▪ int b ▪ int w ▪ int a	*color* stores a color value. There are 5 channels (red, green, blue, white, alpha) with values between 0 and 255. Example: color c = {255, 255, 0, 0}; Member examples: c.r, c.g, c.b, c.w, c.a Learn more »List Of Global Variables And Constants »Script Example

date	▪ int day ▪ int weekday ▪ int month ▪ int year	*date* stores a date. Values for *day* include 1 to 31 for a single day of the month. Values for *weekday* include: 0 = Sunday, 1 = Monday, ..., 6 = Saturday. Values for *month* include 1 to 12 for every single month of the year. Values for *year* include year dates. Example: date d = {24, 11, 1980}; Member examples: d.day, d.weekday, d.month, d.year »**Script Example**
time	▪ int hour ▪ int min ▪ int sec	*time* stores a certain time. Valid values are: hours: 0 .. 23, minutes: 0 .. 59, seconds: 0 .. 59. Example: time t = {12, 05, 00}; Member examples: t.hour, t.min, t.sec »**Script Example**
font	▪ 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	*font* stores a specific font face. *height* specifies the size of the font and requires an *integer* value. *width* specifies the wideness of the font and requires an *integer* value. *escapement* specifies the desired rotation angle in tenths of a degree and requires an *integer* value. *orientation* should be set to the same value as *escapement* and requires an *integer* value. *weight* specifies the weight of the font. Valid values are: **FONT_WEIGHT_DONTCARE** **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** *italic* specifies the sloping of the font and requires an *integer* value: *0* (off) or *1* (on). *underline* draws a line under the font and requires an *integer* value: *0* (off) or *1* (on). *strikeOut* draws a line through the middle of the font and requires an *integer* value: *0* (off) or *1* (on). *charset* 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** *outprecision* 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** *clipprecision* 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** **PRECIS_CLIP_EMBEDDED** *quality* specifies the quality of the font. Valid values are: **QUALITY_DEFAULT** **QUALITY_DRAFT** **QUALITY_PROOF** **QUALITY_NONANTIALIASED** **QUALITY_ANTIALIASED** **QUALITY_CLEARTYPE** **QUALITY_CLEARTYPE_NATURAL**

		pitch specifies the pitch of the font. Valid values for are: **PITCH_DEFAULT** **PITCH_FIXED** **PITCH_VARIABLE** **PITCH_MONO_FONT** *family* 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** *fontname* requires a *string*. For example "MS Sans Serif". »**Script Example**

shape	▪ int renderingMode ▪ int shapeAlignment ▪ int shapeRotation ▪ int blendingMode ▪ int originType ▪ float border ▪ float innerGlow ▪ float outerGlow ▪ int innerGlowInterpolation ▪ int outerGlowInterpolation ▪ float proportion ▪ float diagonalLength	*shape* stores specific information for shapes. Valid values for *renderingMode* are: **RENDERING_MODE_EXTENDED** **RENDERING_MODE_SIMPLE** Valid values for *shapeAlignment* are: **LOOKAT_FRONT** **LOOKAT_BACK** **LOOKAT_LEFT** **LOOKAT_RIGHT** **LOOKAT_TOP** **LOOKAT_BOTTOM** **LOOKAT_RANDOM** Valid values for *shapeRotation* are: **ROTATION_CCW_0** **ROTATION_CCW_90** **ROTATION_CCW_180** **ROTATION_CCW_270** **ROTATION_CW_0** **ROTATION_CW_90** **ROTATION_CW_180** **ROTATION_CW_270** *blendingMode* is only available for RENDERING_MODE_EXTENDED. Valid values are: **BLENDING_MODE_NONE** **BLENDING_MODE_ALPHA** Valid values for *originType* are: **ORIGIN_CENTER** **ORIGIN_GEOMETRIC_CENTER** **ORIGIN_FRONT** **ORIGIN_BACK** **ORIGIN_LEFT** **ORIGIN_RIGHT** **ORIGIN_TOP** **ORIGIN_BOTTOM** **ORIGIN_TOP_LEFT** **ORIGIN_TOP_RIGHT** **ORIGIN_BOTTOM_LEFT** **ORIGIN_BOTTOM_RIGHT** **ORIGIN_FRONT_LEFT** **ORIGIN_FRONT_RIGHT** **ORIGIN_BACK_LEFT** **ORIGIN_BACK_RIGHT** **ORIGIN_FRONT_TOP** **ORIGIN_FRONT_BOTTOM** **ORIGIN_BACK_TOP** **ORIGIN_BACK_BOTTOM** **ORIGIN_FRONT_TOP_LEFT** **ORIGIN_FRONT_TOP_RIGHT** **ORIGIN_FRONT_BOTTOM_LEFT** **ORIGIN_FRONT_BOTTOM_RIGH** **ORIGIN_BACK_TOP_LEFT**

		ORIGIN_BACK_TOP_RIGHT **ORIGIN_BACK_BOTTOM_LEFT** **ORIGIN_BACK_BOTTOM_RIGHT** *border* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *innerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. *outerGlow* is only available for RENDERING_MODE_EXTENDED. Valid values range from *0.01* to *1.00*. Valid values for *innerGlowInterpolation* are: **INTERPOLATION_TYPE_LINEAR** **INTERPOLATION_TYPE_EASE_BOUNCE_IN** **INTERPOLATION_TYPE_EASE_BOUNCE_OUT** **INTERPOLATION_TYPE_EASE_BOUNCE_INOUT** **INTERPOLATION_TYPE_EASE_CIRC_IN** **INTERPOLATION_TYPE_EASE_CIRC_OUT** **INTERPOLATION_TYPE_EASE_CIRC_INOUT** **INTERPOLATION_TYPE_EASE_CUBIC_IN** **INTERPOLATION_TYPE_EASE_CUBIC_OUT** **INTERPOLATION_TYPE_EASE_CUBIC_INOUT** **INTERPOLATION_TYPE_EASE_SINE_IN** **INTERPOLATION_TYPE_EASE_SINE_OUT** **INTERPOLATION_TYPE_EASE_SINE_INOUT** **INTERPOLATION_TYPE_EASE_EXPO_IN** **INTERPOLATION_TYPE_EASE_EXPO_OUT** **INTERPOLATION_TYPE_EASE_EXPO_INOUT** Valid values for *outerGlowInterpolation* are: See *innerGlowInterpolation* *proportion* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_CROSS_STRAIGHT, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, SHAPE_TYPE_3D_CROSS_STRAIGHT, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*. *diagonalLength* is only available for RENDERING_MODE_EXTENDED and for SHAPE_TYPE_CROSS, SHAPE_TYPE_STAR, SHAPE_TYPE_3D_CROSS, and SHAPE_TYPE_3D_STAR. Valid values range from *0.01* to *1.00*. »**Script Example**

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
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

Index	Spectrum In Hz	Index	Spectrum In Hz	Index	Spectrum In Hz
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
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

Index	Spectrum In Hz	Index	Spectrum In Hz	Index	Spectrum In Hz
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
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

Index	Spectrum In Hz	Index	Spectrum In Hz	Index	Spectrum In Hz
106	899.01	277	4,026.71	447	9,200.06
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
141	1,367.36	312	4,920.34	482	10,529.74
142	1,383.51	313	4,947.25	483	10,567.42

Index	Spectrum In Hz	Index	Spectrum In Hz	Index	Spectrum In Hz
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#	B

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	37.128	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	105.60	110.00	118.80	123.75
Index	48	49	50	51	52	53	54	55	56	57	58	59
Freq. (Hz)	132.00	137.00	148.50	158.40	465.00	176.00	185.60	198.00	211.20	220.00	237.60	247.50
Index	60	61	62	63	64	65	66	67	68	69	70	71
Freq. (Hz)	264.00	275.00	297.00	316.80	330.00	352.00	371.25	396.00	422.40	440.00	475.20	495.00
Index	72	73	74	75	76	77	78	79	80	81	82	83
Freq. (Hz)	528.00	550.00	594.00	633.60	660.00	704.00	742.50	792.00	844.80	880.00	950.40	990.00
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	10,137	10,560	11,264	11,880	12,672				

3.9 Examples

Script Examples for Specific Functions

GetApplicationPath

Copy and paste this Macro into the Global Macro Editor, for example, 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 this Macro into the Global Macro Editor, for example, 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 this Macro into the Global Macro Editor, for example, 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 this Macro into the Global Macro Editor, for example, 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 this Macro into the Global Macro Editor, for example, 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 this Macro into the Global Macro Editor, for example, 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 this Macro into the Macro Editor of the effect SCE Ticker and monitor the Previews to see the result.

```
@scriptname="local ticker time";
@author="inoage";
@version="MADRIX 2.10";
@description="Set time in SCE_Ticker text with GetTime and pm and 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

Copy and paste this Macro into the Global Macro Editor, for example. It 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,
         "MS Sans Serif"};

int i=0;

void InitEffect()
{
    i=0;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
   f.height=i%40;
   DrawPixelText(WHITE,f,"Hello",0,0,ROTATION_CCW_0);
}

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

»Description

DrawPixelText - Font Color

Copy and paste this Macro into the Gobal Macro Editor, for example. It 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,
         "MS Sans Serif"};

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_CCW_0);
}

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

»Description

DrawPixelText - Moving Text

Copy and paste this Macro into the Global Macro Editor, for example It 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,
         "MS Sans Serif"};

int i=0;

void InitEffect()
{
    i=0;
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
    i=i%50;
   DrawPixelText(RED,f,"Hello",i,i,ROTATION_CCW_0);
}

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

»Description

DrawPixelText - Rotating Text

Copy and paste this Macro into the Global Macro Editor, for example. 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,
         "MS Sans Serif"};

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_CCW_0);break;
         case 1:DrawPixelText(RED,f,"Hello",25,25,ROTATION_CCW_90);break;
         case 2:DrawPixelText(RED,f,"Hello",25,25,ROTATION_CCW_180);break;
         case 3:DrawPixelText(RED,f,"Hello",25,25,ROTATION_CCW_270);break;
    }
}

void MatrixSizeChanged()
```

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

»Description

GetTimeCode

Retrieves the currently used Time Code. This Macro/Script works in all four Script locations.

```
@scriptname="GetTimeCode";
@author="inoage";
@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 Macro/Script works in all four Script locations.

```
@scriptname="DmxInToColor";
@author="inoage";
@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[]; // array 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. This Script works in the Global Macro.

```
@scriptname="MIDItoMaster";
@author="inoage";
@version="";
@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)
   {
      // MIDI NOTES 0x9, 0x8
      const float Value = (float)GetMidiInNoteValue(NOTE,CHANNEL,DEVICE_ID)/127.0;
      // MIDI CONTROLLER 0xb
      //const float Value = (float)GetMidiInControlValue(NOTE,CHANNEL,DEVICE_ID)/127.0;
      SetMasterFader(Value*255);
   }
}

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

»Description

GetMidiInNote And GetMidiInControl

Uses incoming MIDI-IN data to control the Master and the Audio Input Level. This Script works in the Global Macro.

```
@scriptname="MIDItoMasterandAudio";
@author="inoage";
@version="";
@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)
   {
      // MIDI NOTES 0x9, 0x8
      GetMidiInNote(MidiData, NOTE, NOTE_COUNT, CHANNEL, DEVICE_ID);
      // MIDI NOTES 0xb
      //GetMidiInControl(MidiData, NOTE, NOTE_COUNT, CHANNEL, DEVICE_ID);
      SetMasterFader(MidiData[0]*255/127);
      SetAudioInputFader(MidiData[1]*255/127);
   }
}

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

»Description

DrawVectorText - Font Size

Copy and paste this Macro into the Global Macro Editor, for example. It 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,
         "MS Sans Serif"};

int i=0;

void InitEffect()
{
    i=0;
```

```
}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
    i++;
   f.height=i%40;
   DrawVectorText(WHITE,f,"Hello",0.0,0.0,ROTATION_CCW_0);
}

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

»Description

DrawVectorText - Font Color

Copy and paste this Macro into the Global Macro Editor, for example. It 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,
         "MS Sans Serif"};

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;
   DrawVectorText(col,f,"Hello",0.0,0.0,ROTATION_CCW_0);
}

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

»Description

GetTimeSunrise / GetTimeSunset

Copy and paste this Macro into the Global Macro Editor, for example. It provides the sunrise and sunset times of Dresden of today in the ***Script Output***.

```
@scriptname="";
@author="";
@version="3.3";
@description="get sunrise/sunset of Dresden of today";

void InitEffect()
{
    date d = GetDate();
    time t = GetTimeSunrise(d,51,3,13,44, 1.0); // DRESDEN
    WriteText(t.hour+":"+t.min+":"+t.sec);
    t = GetTimeSunriseCity(d, CITY_DRESDEN);
    WriteText(t.hour+":"+t.min+":"+t.sec);
    t = GetTimeSunset(d,51,3,13,44, 1.0);  // DRESDEN
    WriteText(t.hour+":"+t.min+":"+t.sec);
    t = GetTimeSunsetCity(d, CITY_DRESDEN);
    WriteText(t.hour+":"+t.min+":"+t.sec);
}

void PreRenderEffect()
{}

void PostRenderEffect()
{}

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

»Description

GetTimeSunriseCity / GetTimeSunsetCity

Copy and paste this Macro into the Global Macro Editor, for example. It provides the sunrise and sunset times of Berlin of today in the ***Script Output***.

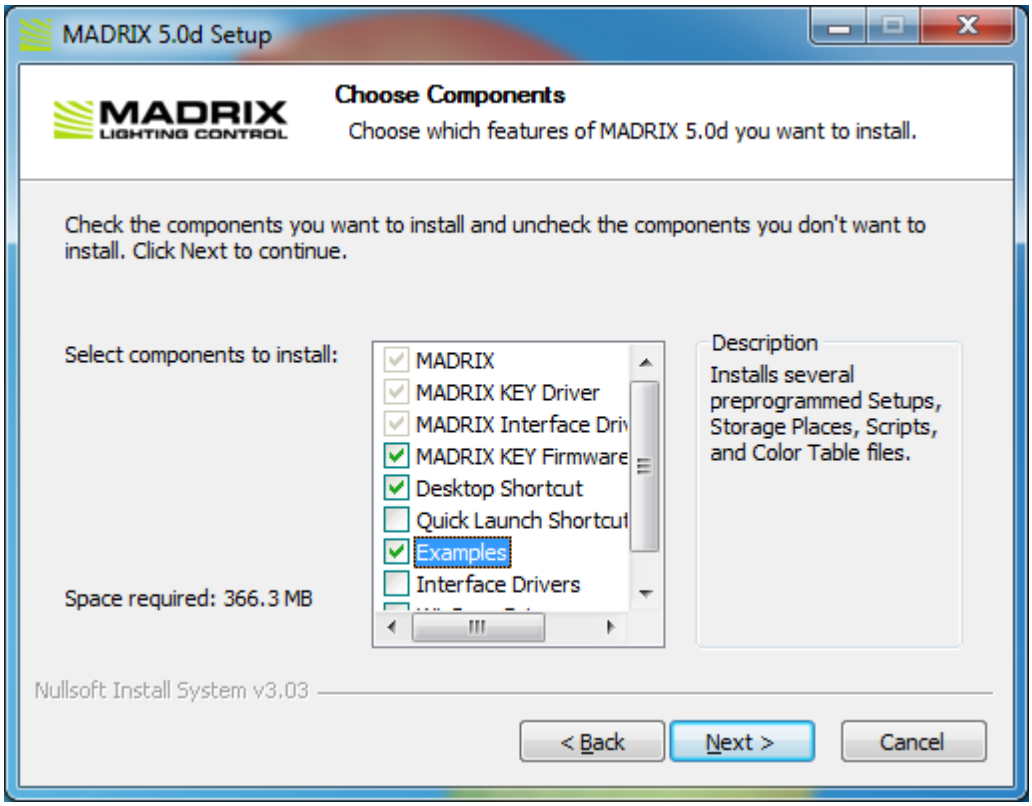
```
@scriptname="";
@author="";
@version="3.3";
@description="get sunrise/sunset of Berlin of today";

void InitEffect()
{
    date d =GetDate();// today
    time tr =GetTimeSunriseCity(d,CITY_BERLIN);
    WriteText("Sunrise: "+(string)tr.hour+":"+(string)tr.min);
    time ts =GetTimeSunsetCity( d,CITY_BERLIN);
    WriteText("Sunset : "+(string)ts.hour+":"+(string)ts.min);
}

void PreRenderEffect()
{}

void PostRenderEffect()
{}

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 harddisk. Please navigate to *C:\Users\Public\Documents\MADRIX5 Samples\scripts*

It contains various examples.

MADRIX LIGHTING CONTROL

//PART D

MAS Script Effect

4 MAS Script Effect

4.1 Overview (MAS Script Effect)

Introduction

- The MAS Script Effect provides the potential to create your own effects. That could be a completely new effect that MADRIX does not offer in its stock library of effects.
- 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.
- Scripts are stored as part of the effect. This means they are part of a stored Storage Place file or Setup file.
- It is possible to save scripts as separate files. The file extension of a MAS script is **.mas*. The extension of a compiled script is **.macs*.
- The MAS Script Effect is an effect of MADRIX. For that reason, its result can be controlled and manipulated by a »**Macro For Effects** like all the other effects as well.
- The MAS Script Effect can be selected from the effect list like all the other effects.

Functions Called By MADRIX

Overview

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 array 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 arrays, 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. This is the right place to calculate the effect and draw it onto the matrix.

MartrixSizeChanged

(Automatically included in a new script)

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 Script Editor, the empty standard script will look like this:

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

void InitEffect()
{

}

void RenderEffect()
{

}

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

4.2 Functions (MAS Script Effect)

Specific Resources

- »Functions called by MADRIX
- MAS Script Effect: Available Functions

- »Using Frames

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

- For non-specific functions, see »List of Functions (Alphabetical Order)
- In addition to that, there are several other functions available for the MAS Script Effect which are not available for macros.

MAS Script-Specific Functions

1) General

Function	Description
void **SetFrameCount**(float fc)	Set the number of frames the effect produces before it gets repeated. See »here for further details.

Learn more »Using Frames

2) BlackTrax

Learn more »BlackTrax

4.3 Using Frames

Introduction

- The MADRIX user interface provides several possibilities to change the speed of effects, such as Speed Master Left/Right, Speed Pitch, and BPM.
- In MADRIX Script, additional functions are provided that allow for much more control.

Functions

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Global Macro
void **SetAsync**(int value)	Sets the asynchronous rendering mode. This function is only available for some effects.	+	+		
int **GetAsync**()	Returns if asynchronous rendering is used. This function is only available for some effects.	+	+		
void **ToggleAsync**()	Toggles the asynchronous rendering mode. This function is only available for some effects.	+	+		

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.	+	+		
float **GetFrameSteps**()	Returns the number of frames between this and the last call.	+	+		
float **GetFrameCount**()	Retrieves the number of frames the effect produces before it gets repeated. This is the same value which was set by *SetFrameCount* for MAS Script. The initial value is 1000.0.	+	+		
void **SetFrameCount**(float fc)	Sets the number of frames the effect produces before it gets repeated.	+			
void **SetStep**(int value)	Sets the stepped rendering mode. This function is only available for some effects.	+	+		
int **GetStep**()	Returns if stepped rendering is used. This function is only available for some effects.	+	+		
void **ToggleStep**()	Toggles the stepped rendering mode. This function is only available for some effects.	+	+		

Applying Asynchronous Rendering

Overview

Normally, MADRIX Effects are synchronized to the Main Mixing Frame Rate, which is set to 50 Hz by default. But for the »**MAS Script** Effect there is the option to control the render frequency independently.

- **`void SetAsync(int value)`**
- **`int GetAsync()`**
- **`void ToggleAsync()`**

Valid values for *value* are *0* (*Off*) or *1* (*On*). If asynchronous rendering is activated, the render frequency directly corresponds to the BPM value set by the slider or the macro. For example, a BPM value of 1200 causes a render frequency of 20 Hz. In other words, the effect renders 20 frames per second. Thus, you can decrease the render frequency in order to save performance by slowing down the effect. However, there is an upper limit for the render frequency, which is 50 Hz by default according to the Main Mixing.

There is another scenario to use asynchronous rendering in the MAS Script Effect: Imagine, you want to draw a line step by step, one pixel per frame. Straight forwardly, you can set one pixel after another with each call of » **RenderEffect**. In order to control the speed of drawing with the BPM slider, it would be smart to activate asynchronous rendering. That way, RenderEffect is called more or less often, depending on the BPM value. Yet another example would be to write a speed-sensitive counter which draws sequential numbers according to the BPM value.

Summary

- During execution of an MAS Script Effect script, you can use the BPM slider.
- If ***Async*** is deactivated (***Sync*** is activated), the script runs with the Main Mixing frequency. The render frequency is fixed. The BPM slider controls the Frame ID / Frame Step.
  (Go to ***Preferences > Options... > Performance*** to change the Main Mixing FPS if really necessary.)
- If ***Async*** is activated, the MAS Script Effect calculates its render frequency according to the BPM slider (60 BPM = 1 FPS; 1200 BPM = 20 FPS; 3000 BPM = 50 FPS; etc.). The render frequency is variable. The maximum render frequency is still set up by the Main Mixing frequency.
  (Go to ***Preferences > Options... > Performance*** to change the Main Mixing FPS if really necessary.)
- When Async is activated, use the following formulas to calculate the speed or render frequency:

```
BPM = FPS * 60
FPS = BPM / 60
```

Applying Stepped Rendering

Normally, the effects use floating-point frames which is represented by the rational return values of the functions GetFrameId and GetFrameSteps. That way, the effects are able to render very precisely and smooth. However, sometimes it is desired to let the effects "step" pixel by pixel. This option is only available for some effects, for example »**SCE Color Scroll** and »**MAS Script**

- **`void SetStep(int value)`**

- **`int GetStep()`**
- **`void ToggleStep()`**

Valid values for *value* are *0* (*Off*) or *1* (*On*). If stepped rendering is activated, effects like SCE Color Scroll look hard-stepping. In the MAS Script effect, stepped rendering causes GetFrameId() and GetFrameSteps() to return only integer values.

- If stepped rendering is deactivated, Frame IDs will be returned as float values.
  - For example, SCE Color Scroll will be rendered smoothly.
- If stepped rendering is activated, Frame IDs will be returned as integer values.
  - For example, SCE Color Scroll will not look as smooth.

GetFrameSteps

- GetFrameSteps() - Defines the number of steps between Frames / Frame IDs.

`Frame Steps * Main Mixer FPS * 60 = BPM`

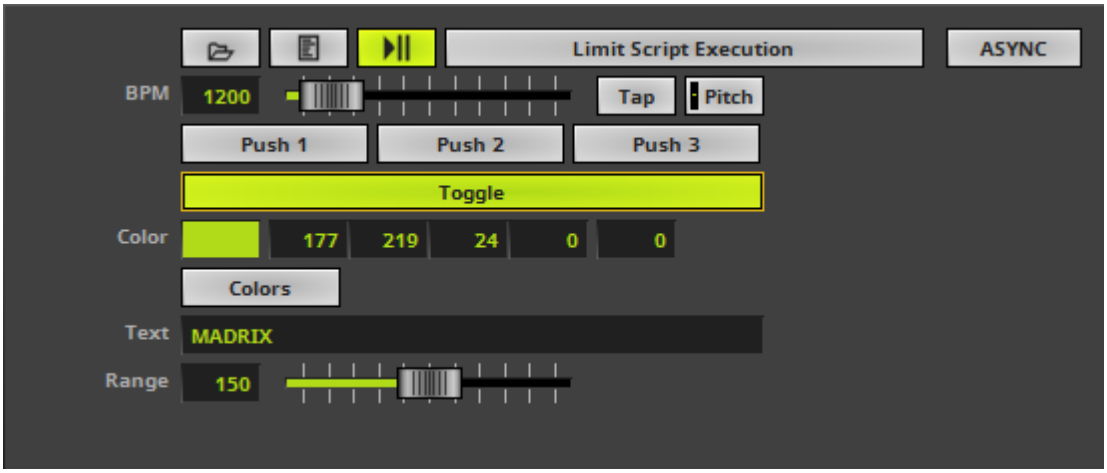
Using Frame Count And Frame ID

Note: The script of the MAS Script effect itself is able to control the Frame ID and the Frame Count. Setting the Frame ID within both locations, a script and an additional macro, may result in undesired behavior.

4.4 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 further. 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 additional elements. For example, there are buttons to be pushed or toggled, color controls to define one or more colors, a text field to enter text, and a slider. Each different element will 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:

```
ctrlbutton myButton;
ctrlcheckbutton myCheckButton;
ctrlcolor myColor;
ctrlcolortable myColorTable;
ctrledit myEdit;
ctrlslider mySlider;
```

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 with global variables which creates the GUI shown above:

```
ctrlbutton3 btnPush = { "Push 1", "OnPushed1",
                        "Push 2", "OnPushed2",
                        "Push 3", "OnPushed3",
                        "starts the first action",
                        "starts the second action",
                        "starts the third action" };
ctrlcheckbutton btnToggle = { "Toggle", "OnToggled", "activates or deactivates the behaviour", 1 };
ctrlcolor clrColor = { "Color", MADRIX_GREEN };
ctrlcolortable tableColors = { "Colors", { RED, GREEN, BLUE }, "opens the Color Table"};
ctrledit editText = { "Text", EDIT_ALIGN_LEFT, "MADRIX", "enter any text" };
ctrlslider sliderRange = { "Range", 100, 150, 200, "defines the value" };
```

Explanation:

- This source code creates each GUI element and initializes it 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 starts with a value of 150.

Using GUI Elements

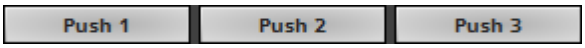
Overview

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:

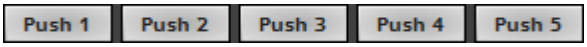
List Of Available Elements

Structure	Entries			GUI Element / Description
ctrlbutton Push	string	label	The label for the button.	*ctrlbutton* provides a button. A function must be assigned, which is called whenever the button has been pressed. This is done by assigning the name of the function to the *proc* element of a *ctrlbutton* variable. The function set must be of type **`void function()`**. The *tooltip* member holds the tooltip for the button, which is shown when the user holds the mouse over the button for some seconds. Description 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.	*ctrlbutton2* 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 *proc[n]* element of a *ctrlbutton* variable. The function set must be of the type **`void function()`**. The *tooltip* members hold the tooltips for the buttons, 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	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.	*ctrlbutton3* 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 *proc[n]* element of a *ctrlbutton* variable. The function set must be of the type **`void function()`**. The *tooltip* members hold the tooltips for the buttons, 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  	string	label1	The label for the first button.	*ctrlbutton4* 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 *proc[n]* element of a *ctrlbutton* variable. The function set must be of type **`void function()`**. The *tooltip* members hold the tooltips for the buttons, 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	label4	Label for the fourth button.	
	string	proc4	Function handler for the fourth 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
	string	tooltip4	Tooltip for the fourth button.	
ctrlbutton5  	string string string string string string string string string string	label1 proc1 label2 proc2 label3 proc3 label4 proc4 label5 proc5	The label for the first button. Function handler for the first button. Label for the second button. Function handler for the second button. Label for the third button. Function handler for the third button. Label for the fourth button. Function handler for the fourth button. Label for the fifth button. Function handler for the fifth button.	*ctrlbutton5* provides five 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 *proc[n]* element of a *ctrlbutton* variable. The function set must be of type **`void function()`**. The *tooltip* members hold the tooltips for the buttons, which are shown when the user holds the mouse over a button for some seconds. **Description** **Example 1** **Example 2** **Example 3**

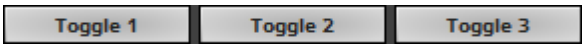
Structure	Entries			GUI Element / Description
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	
	string	tooltip4	Tooltip for the fourth button.	
	string	tooltip5	Tooltip for the fifth button.	

Structure	Entries			GUI Element/ Description
ctrlbutton6  	string	label1	The label for the first button.	*ctrlbutton6* provides six 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 *proc[n]* element of a *ctrlbutton* variable. The function set must be of type **`void function()`**. The *tooltip* members hold the tooltips for the buttons, 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	label4	Label for the fourth button.	
	string	proc4	Function handler for the fourth button.	
	string	label5	Label for the fifth button.	
	string	proc5	Function handler for the fifth button.	
	string	label6	Label for the sixth button.	

Structure	Entries			GUI Element / Description
	string	proc6	Function handler for the sixth button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	
	string	tooltip4	Tooltip for the fourth button.	
	string	tooltip5	Tooltip for the fifth button.	
	string	tooltip6	Tooltip for the sixth button.	

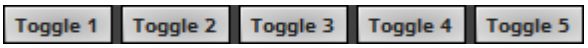
Structure	Entries			GUI Element/ Description
ctrlcheckbutton Toggle	string	label	The label for the first button.	*ctrlcheckbutton* provides a check button. A function must be assigned, which is called whenever the check button has been toggled. This is done by assigning the name of the function to the *proc* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check button can be queried and changed by using the *checked* element of a *ctrlcheckbutton* variable. The *tooltip* member holds the tooltip for the check button, which is shown when the user holds the mouse over the check button for some seconds. Example 1 Example 2
	string	proc	Function handler for the first button.	
	string	tooltip	Tooltip for the first button.	
	int	checked	State of the first button.	

Structure	Entries			GUI Element / Description
ctrlcheckbutton2  	string	label1	The label for the first button.	*ctrlcheckbutton2* provides two check buttons in one line. For each check button a function must be assigned, which is called whenever the corresponding check button has been toggled. This is done by assigning the name of the function to the *proc[n]* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check buttons can be queried and changed by using the *checked[n]* elements of a *ctrlcheckbutton* variable. The *tooltip* members hold the tooltips for the check buttons, which are shown when the user holds the mouse over a check button for some seconds. **Example 1** **Example 2**
	string	label2	Label for the second button.	
	string	proc1	Function handler for the first button.	
	string	proc2	Function handler for the second button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	int	checked1	State of the first button.	
	int	checked2	State of the second button.	

Structure	Entries			GUI Element / Description
ctrlcheckbutton3  	string	label1	The label for the first button.	*ctrlcheckbutton3* provides three check buttons in one line. For each check button a function must be assigned, which is called whenever the corresponding check button has been toggled. This is done by assigning the name of the function to the *proc[n]* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check buttons can be queried and changed by using the *checked[n]* elements of a *ctrlcheckbutton* variable. The *tooltip* members hold the tooltips for the check buttons, which are shown when the user holds the mouse over a check button for some seconds. Example 1 Example 2
	string	label2	Label for the second button.	
	string	label3	Label for the third button.	
	string	proc1	Function handler for the first button.	
	string	proc2	Function handler for the second 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.	
	int	checked1	State of the first button.	
	int	checked2	State of the second button.	
	int	checked3	State of the third button.	

Structure	Entries			GUI Element / Description
ctrlcheckbutton4  	string	label1	The label for the first button.	*ctrlcheckbutton4* provides four check buttons in one line. For each check button a function must be assigned, which is called whenever the corresponding check button has been toggled. This is done by assigning the name of the function to the *proc[n]* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check buttons can be queried and changed by using the *checked[n]* elements of a *ctrlcheckbutton* variable. The *tooltip* members hold the tooltips for the check buttons, which are shown when the user holds the mouse over a check button for some seconds. Example 1 Example 2
	string	label2	Label for the second button.	
	string	label3	Label for the third button.	
	string	label4	Label for the fourth button.	
	string	proc1	Function handler for the first button.	
	string	proc2	Function handler for the second button.	
	string	proc3	Function handler for the third button.	
	string	proc4	Function handler for the fourth button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	

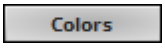
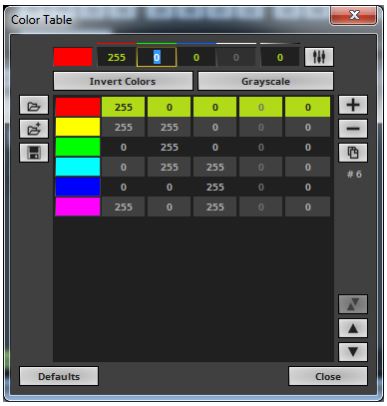
Structure	Entries			GUI Element / Description
	string	tooltip4	Tooltip for the fourth button.	
	int	checked1	State of the first button.	
	int	checked2	State of the second button.	
	int	checked3	State of the third button.	
	int	checked4	State of the fourth button.	

Structure	Entries			GUI Element / Description
ctrlcheckbutton5  	string	label1	The label for the first button.	*ctrlcheckbutton5* provides five check buttons in one line. For each check button a function must be assigned, which is called whenever the corresponding check button has been toggled. This is done by assigning the name of the function to the *proc[n]* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check buttons can be queried and changed by using the *checked[n]* elements of a *ctrlcheckbutton* variable. The *tooltip* members hold the tooltips for the check buttons, which are shown when the user holds the mouse over a check button for some seconds. Example 1 Example 2
	string	label2	Label for the second button.	
	string	label3	Label for the third button.	
	string	label4	Label for the fourth button.	
	string	label5	Label for the fifth button.	
	string	proc1	Function handler for the first button.	
	string	proc2	Function handler for the second button.	
	string	proc3	Function handler for the third button.	
	string	proc4	Function handler for the fourth button.	
	string	proc5	Function handler for the fifth button.	
	string	tooltip1	Tooltip for the first button.	

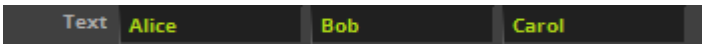
Structure	Entries			GUI Element / Description
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	
	string	tooltip4	Tooltip for the fourth button.	
	string	tooltip5	Tooltip for the fifth button.	
	int	checked1	State of the first button.	
	int	checked2	State of the second button.	
	int	checked3	State of the third button.	
	int	checked4	State of the fourth button.	
	int	checked5	State of the fifth button.	

Structure	Entries			GUI Element / Description
ctrlcheckbutton6  	string	label1	The label for the first button.	*ctrlcheckbutton6* provides six check buttons in one line. For each check button a function must be assigned, which is called whenever the corresponding check button has been toggled. This is done by assigning the name of the function to the *proc[n]* element of a *ctrlcheckbutton* variable. The function set must be of type **`void function()`**. The state of the check buttons can be queried and changed by using the *checked[n]* elements of a *ctrlcheckbutton* variable. The *tooltip* members hold the tooltips for the check buttons, which are shown when the user holds the mouse over a check button for some seconds. Example 1 Example 2
	string	label2	Label for the second button.	
	string	label3	Label for the third button.	
	string	label4	Label for the fourth button.	
	string	label5	Label for the fifth button.	
	string	label6	Label for the sixth button.	
	string	proc1	Function handler for the first button.	
	string	proc2	Function handler for the second button.	
	string	proc3	Function handler for the third button.	
	string	proc4	Function handler for the fourth button.	
	string	proc5	Function handler for the fifth button.	

Structure	Entries			GUI Element / Description
	string	proc6	Function handler for the sixth button.	
	string	tooltip1	Tooltip for the first button.	
	string	tooltip2	Tooltip for the second button.	
	string	tooltip3	Tooltip for the third button.	
	string	tooltip4	Tooltip for the fourth button.	
	string	tooltip5	Tooltip for the fifth button.	
	string	tooltip6	Tooltip for the sixth button.	
	int	checked1	State of the first button.	
	int	checked2	State of the second button.	
	int	checked3	State of the third button.	
	int	checked4	State of the fourth button.	
	int	checked5	State of the fifth button.	
	int	checked6	State of the sixth button.	

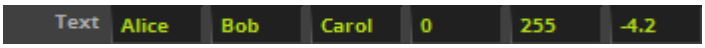
Structure	Entries			GUI Element / Description
ctrlcolor  	string	label	The label for the element.	*ctrlcolor* provides a color input control with the Color Picker and five edit fields to enter the red, green, blue, white, and alpha values. The alpha value will not be provided/ set by the Color Picker. **Example**
	color	value	The value set by the element.	
ctrlcolortable   	string	label	Label of the button.	*ctrlcolortable* 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. The *tooltip* is shown when the user holds the mouse over the button that opens the color table for some seconds. **Example**
	color[]	value	A field of values which holds the entries of the color table.	
	string	tooltip	The tooltip for the button to open the color table.	

Structure	Entries			GUI Element / Description
ctrledit 	string	label	The label for the element.	*ctrledit* provides an edit control which allows to enter a text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	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.	
ctrledit2 	string	label	The label for the element.	*ctrledit2* provides two edit controls in one line to enter text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	int	align	Alignment of the edit fields	
	string	value1	The first value.	
	string	value2	The second value.	
	string	tooltip1	Tooltip for the first edit control.	
	string	tooltip2	Tooltip for the second edit control.	

Structure	Entries			GUI Element / Description
ctrledit3  	string	label	The label for the element.	*ctrledit3* provides three edit controls in one line to enter text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	int	align	Alignment of the edit fields	
	string	value1	The first value.	
	string	value2	The second value.	
	string	value3	The third 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.	

Structure	Entries			GUI Element / Description
ctrledit4 	string	label	The label for the element.	*ctrledit4* provides four edit controls in one line to enter text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	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
ctrledit5 	string	label	The label for the element.	*ctrledit5* provides five edit controls in one line to enter text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	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	value5	The fifth 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.	
	string	tooltip5	Tooltip for the fifth edit control.	

Structure	Entries			GUI Element / Description
ctrledit6  	string	label	The label for the element.	*ctrledit6* provides six edit controls in one line to enter text or values. The variable *align* can be set to one of the following values: **EDIT_ALIGN_LEFT**, **EDIT_ALIGN_CENTER**, or **EDIT_ALIGN_RIGHT**. Example 1 Example 2 Example 3
	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	value5	The fifth value.	
	string	value6	The sixth 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.	
	string	tooltip5	Tooltip for the fifth edit control.	

Structure	Entries			GUI Element / Description
	string	tooltip6	Tooltip for the sixth edit control.	
ctrlslider  	string int int int string	label rangeMin value rangeMax tooltip	The label for the element The minimum value the slider provides. This value must be greater than or equal to 0. The start value set by the slider. The maximum value the slider provides. This value must be greater than the minimum value. Holds the tooltip for the slider.	*ctrlslider* 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. You can set a range for the slider. The minimum value (*rangeMin*) has to be greater or equal to 0, while the maximum value (*rangeMax*) must be greater than the minimum value. If *value* itself is set to a value outside the range, it is automatically set to the minimum or maximum value. The *tooltip* is shown when the user holds the mouse over the slider for some seconds. **Example**

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 for the MAS Script Effect 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 by the slider
    color chg = {0, 0, 0, 0, chgB.value}; // increase alpha value continuously
    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 control 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. Its Alpha value is set by the *value* provided by the slider *chgB*. 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 for the MAS Script Effect:

```
@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 Compiler Messages of the Script Editor.

Description

Examples

Button Example 1 (*ctrlbutton*)

The following example for the MAS Script Effect creates four buttons. Pressing a button will activate the color indicated by the button (Red, Green, Blue, or White).

```
@scriptname="ColorTestWithButton";
@author="inoage 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 for the MAS Script Effect 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 = {"Blinking Off", "OnBlinkOff",
"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 On";
    g_btnMode.proc = "OnBlinkOn";
    g_btnMode.tooltip = "Enable blinking mode";
    WriteText("Blink disabled");
}

void OnBlinkOn()
{
    g_mode = MODE_BLINK;
    g_btnMode.label = "Blinking Off";
    g_btnMode.proc = "OnBlinkOff";
```

```
    g_btnMode.tooltip = "Disable blinking mode";
    WriteText("Blinking enabled");
}
```

Description

Button Example 3 (*ctrlbutton, ctrlbutton2, ctrlbutton3, ctrlbutton4*)

The following example for the MAS Script Effect draws continuing circles and 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="inoage";
@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","set blue color"};
ctrlbutton4 Button4 ={"Red","OnRed","Green","OnGreen","Blue","OnBlue","White","OnWhite",
 "set red color", "set green color","set blue color","set white color"};

void InitEffect()
{
  height = GetMatrixHeight();
 width = GetMatrixWidth();
 Clear();
 i=0;
 SetAsync(true);
 SetBpm(100.0);
}

void RenderEffect()
{
 i++;
 if(i>width/2 || i>height/2)
 {
        i=1;
        Clear();
 }

 DrawPixelShape(col, DRAWSHAPE_CIRCLE, width/2-i, height/2-i, 0, i*2, i*2, 1);
}

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

```
}

void OnRed()
{
 col=RED;
}

void OnGreen()
{
 col=GREEN;
}

void OnBlue()
{
 col=BLUE;
}

void OnWhite()
{
 col=WHITE;
}
```

Description

Check Button Example 1 (*ctrlcheckbutton*)

The following example for the MAS Script Effect creates a check button. The check button toggles if the matrix boundaries are taken into account for the continually moving pixel. By default, this is not the case.

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

ctrlcheckbutton btnToggle = { "Mind The Walls!",
                              "OnMindTheWalls",
                              "activates or deactivates bouncing off the matrix boundaries",
                              0 };

float x, y, dx, dy;

void InitEffect()
{
    x = frandom();
    y = frandom();
    dx = frandom();
    dy = frandom();

    if (dx < 0.5)
        dx = -dx - 0.5;

    if (dy < 0.5)
```

```
        dy = -dy - 0.5;
}

void RenderEffect()
{
    const float fSteps = GetFrameSteps();
    const color fade = { 0, 0, 0, 0, 20 };

    ChangeBrightness(fade);

    x += dx * fSteps;
    y += dy * fSteps;

    if (btnToggle.checked == true)
    {
        // Bounce off
        if (x < 0.0 || x > 1.0)
        {
            dx = -dx;
            x += dx * fSteps;
        }

        if (y < 0.0 || y > 1.0)
        {
            dy = -dy;
            y += dy * fSteps;
        }
    }
    else
    {
        // Go through
        if (x < 0.0)
            x = 1.0 - x;
        else if (x > 1.0)
            x = x - 1.0;

        if (y < 0.0)
            y = 1.0 - y;
        else if (y > 1.0)
            y = y - 1.0;
    }

    SetVectorPixel(ORANGE, x, y);
}

void OnMindTheWalls()
{
    // do nothing
}

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

Description

Check Button Example 2 (*ctrlcheckbutton2*)

The following example for the MAS Script Effect creates two check buttons. The check buttons toggle the display of horizontal and vertical lines, respectively. Note that at least one direction must be active due to the conditions in the check button functions.

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

ctrlcheckbutton2 btnToggle = { "Horizontal Lines",
                               "Vertical Lines",
                               "OnHorizontalLines",
                               "OnVerticalLines",
                               "activates or deactivates horizontal lines",
                               "activates or deactivates vertical lines",
                               1, 0 };

void InitEffect()
{
    SetBpm(60.0);
    SetFrameCount(1.0);
}

void RenderEffect()
{
    const float fId = GetFrameId();
    const color fade = { 0, 0, 0, 0, 20 };

    ChangeBrightness(fade);

    if (btnToggle.checked1 == true)
        DrawVectorLine(BLUE, 0.0, fId, 1.0, fId);

    if (btnToggle.checked2 == true)
        DrawVectorLine(ORANGE, fId, 0.0, fId, 1.0);
}

void OnHorizontalLines()
{
    // At least one direction must be active
    if (btnToggle.checked2 == false)
        btnToggle.checked1 = true;
}

void OnVerticalLines()
{
    // At least one direction must be active
    if (btnToggle.checked1 == false)
        btnToggle.checked2 = true;
```

```
}

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

Description

Color Table (*ctrlcolortable*)

The following example for the MAS Script Effect 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()
{
    SetAsync(true);
    SetBpm(60.0);
}

void RenderEffect()
{
    if(col.value.length > 0)
    {
        Clear(col.value[random(0, col.value.length - 1)]);
    }
}
```

Description

Edit Field (*ctrledit*)

The following example for the MAS Script Effect clears the matrix with the color white. It also creates an edit field to enter the color value 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="inoage";
@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;// value to color red via cast (int)
    Clear(col);// Set color to complete matrix
}

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

Description

Edit Fields (*ctrledit2*)

The following example for the MAS Script Effect creates an edit field ctrledit2 and assigns strings in InitEffect().

```
@scriptname="ctrledit2 example";
@author="inoage";
@version="MADRIX 3.1";
@description="";

ctrledit2 edit;

void InitEffect()
{
    edit.label = "EDIT";
    edit.align = EDIT_ALIGN_RIGHT;

    edit.value1 = "1";
    edit.tooltip1 = "First Edit Field";

    edit.value2 = "2";
    edit.tooltip2 = "Second Edit Field";

}

void RenderEffect()
{

}

void MatrixSizeChanged()
{
```

```
    InitEffect();
}
```

Description

Edit Fields (*ctrledit4*)

The following example for the MAS Script Effect clears the matrix with the color white. It also creates four 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="inoage";
@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

4.5 BlackTrax

Introduction

BlackTrax by CAST Software is a real-time, vision-based tracking system. The data the system provides can be used in MADRIX 5.

Currently, you need to use MADRIX 5 Script in order to let both systems, MADRIX 5 and BlackTrax, work together. MADRIX 5 is using the real-time tracking protocol for the integration in order to receive positional data (RTTrPM).

Available Functions

These functions are available in the MAS Script Effect as well as Macros For Effects.

Please note: Incoming values entirely depend on the configured sender. MADRIX 5 is only receiving the incoming values.

Function	Description
void **BlackTraxBeaconClear**()	Globally deletes the current set of recognized beacons in the MADRIX 5 Software.
int **BlackTraxBeaconIsExist**(string *beaconname*)	Allows you to check if the specified beacon currently exists in the set of recognized beacons in the MADRIX 5 Software. If the return value = 0, the beacon does not exist.
int **BlackTraxBeaconGetTimestamp**(string *beaconname*)	Returns the optional time stamp of the specified beacon. If the return value = 0, no time stamp exists.
int **BlackTraxBeaconGetPositionCentroidLatency**(string *beaconname*)	Returns the approximate time since the last measurement for the center position of the specified beacon in milliseconds.
float **BlackTraxBeaconGetPositionCentroidX**(string *beaconname*)	Returns the center position of the specified beacon in X.
float **BlackTraxBeaconGetPositionCentroidY**(string *beaconname*)	Returns the center position of the specified beacon in Y.
float **BlackTraxBeaconGetPositionCentroidZ**(string *beaconname*)	Returns the center position of the specified beacon in Z.
int **BlackTraxBeaconGetPositionTrackedLatency**(string *beaconname*, int *value*)	Returns the approximate time since the last measurement for the tracked position of the specified beacon in milliseconds. Specify which LED is used via *value* [0, 1, or 2].

float **BlackTraxBeaconGetPositionTrackedX**(string *beaconname*, int *value*)	Returns the position of the specified beacon in X, which can deviate from the center position. As you can specify an offset to the center for each LED, the offset will be automatically added here, which then represents center position plus offset equals tracked position. Specify which LED is used via *value* [0, 1, or 2].
float **BlackTraxBeaconGetPositionTrackedY**(string *beaconname*, int *value*)	Returns the position of the specified beacon in Y, which can deviate from the center position. As you can specify an offset to the center for each LED, the offset will be automatically added here, which then represents center position plus offset equals tracked position. Specify which LED is used via *value* [0, 1, or 2].
float **BlackTraxBeaconGetPositionTrackedZ**(string *beaconname*, int *value*)	Returns the position of the specified beacon in Z, which can deviate from the center position. As you can specify an offset to the center for each LED, the offset will be automatically added here, which then represents center position plus offset equals tracked position. Specify which LED is used via *value* [0, 1, or 2].
int **BlackTraxBeaconGetQuaterionsLatency**(string *beaconname*)	Returns the approximate time since the last measurement for the quaternions position of the specified beacon in milliseconds.
float **BlackTraxBeaconGetQuaternionsX**(string *beaconname*)	Returns the position of the specified beacon in X as expressed in quaternions; a number system for complex numbers and three-dimensional spaces.
float **BlackTraxBeaconGetQuaternionsY**(string *beaconname*)	Returns the position of the specified beacon in Y as expressed in quaternions; a number system for complex numbers and three-dimensional spaces.
float **BlackTraxBeaconGetQuaternionsZ**(string *beaconname*)	Returns the position of the specified beacon in Z as expressed in quaternions; a number system for complex numbers and three-dimensional spaces.
float **BlackTraxBeaconGetQuaternionsW**(string *beaconname*)	Returns the angle W of the specified beacon as expressed in quaternions; a number system for complex numbers and three-dimensional spaces.
int **BlackTraxBeaconGetEulerLatency**(string *beaconname*)	Returns the approximate time since the last measurement for the Euler position of the specified beacon in milliseconds.
int **BlackTraxBeaconGetEulerOrder**(string *beaconname*)	Returns the direction of the specified beacon as expressed in an Euler system.
float **BlackTraxBeaconGetEulerRotation1**(string *beaconname*)	Returns the first part of the rotation of the specified beacon as expressed in an Euler system.
float **BlackTraxBeaconGetEulerRotation2**(string *beaconname*)	Returns the second part of the rotation of the specified beacon as expressed in an Euler system.
float **BlackTraxBeaconGetEulerRotation3**(string *beaconname*)	Returns the third part of the rotation of the specified beacon as expressed in an Euler system.
float **BlackTraxBeaconGetAccelerationX**(string *beaconname*)	Returns the acceleration of the specified beacon in X.
float **BlackTraxBeaconGetAccelerationY**(string *beaconname*)	Returns the acceleration of the specified beacon in Y.
float **BlackTraxBeaconGetAccelerationZ**(string *beaconname*)	Returns the acceleration of the specified beacon in Z.
float **BlackTraxBeaconGetVelocityX**(string *beaconname*)	Returns the speed of the specified beacon in X.

float **BlackTraxBeaconGetVelocityY**(string *beaconname*)	Returns the speed of the specified beacon in Y.
float **BlackTraxBeaconGetVelocityZ**(string *beaconname*)	Returns the speed of the specified beacon in Z.

Example

```
@scriptname="BlackTrax Test Sample";
@author="inoage GmbH";
@version="1.0";
@description="An example of reading BlackTrax RTTrPM data";

void InitEffect()
{
 BlackTraxBeaconClear();
}

void RenderEffect()
{
 const int iCount = 85; // maximum classic beacon count
 for(int i=0; i<iCount; i++)
 {
        if(BlackTraxBeaconIsExist((string)(i)) != 0)
        {
               // POSITION CENTROID
               const int  lc = BlackTraxBeaconGetPositionCentroidLatency((string)(i));
               const float x = BlackTraxBeaconGetPositionCentroidX((string)(i));
               const float y = BlackTraxBeaconGetPositionCentroidY((string)(i));
               const float z = BlackTraxBeaconGetPositionCentroidZ((string)(i));
               WriteText("Beacon " + (string)(i) + " Position Centroid | x "
               + (string)x + " | y " +(string)y + " | z " +(string)z
               + " | Latency " +(string)lc);

               SetVectorPixel(RED, x, y);
               SetVectorPixel3D(WHITE, x, y, z);

               // POSITION TRACKED LED 1,2,3
               for(int j=0; j<3;j++)
               {
                      const int  lt = BlackTraxBeaconGetPositionTrackedLatency((string)(i), j);
                      const float x = BlackTraxBeaconGetPositionTrackedX((string)(i), j);
                      const float y = BlackTraxBeaconGetPositionTrackedY((string)(i), j);
                      const float z = BlackTraxBeaconGetPositionTrackedZ((string)(i), j);
                      WriteText("Beacon " + (string)(i) + " Pos Tracked Led "+(string)j
                      + " | x " + (string)x + " | y " +(string)y + " | z " +(string)z +
                      " | Latency " +(string)lt);
               }

               // TIMESTAMP
               const int   t = BlackTraxBeaconGetTimestamp(  (string)(i));
               WriteText("Beacon " + (string)(i) + " Timestamp    | " +(string)t);

               // QUATERNIONS
```

```
                const int   lq = BlackTraxBeaconGetQuaternionsLatency((string)(i));
                const float qx = BlackTraxBeaconGetQuaternionsX((string)(i));
                const float qy = BlackTraxBeaconGetQuaternionsY((string)(i));
                const float qz = BlackTraxBeaconGetQuaternionsZ((string)(i));
                const float qw = BlackTraxBeaconGetQuaternionsW((string)(i));
                WriteText("Beacon " + (string)(i) + " Quaternions | qx "
                + (string)qx + " | qy " +(string)qy + " | qz "
                +(string)qz + " | qw " +(string)qw + " | Latency " +(string)lq);

                // EULER
                const int   le = BlackTraxBeaconGetEulerLatency((string)(i));
                const int   eo = BlackTraxBeaconGetEulerOrder((string)(i));
                const float e1 = BlackTraxBeaconGetEulerRotation1((string)(i));
                const float e2 = BlackTraxBeaconGetEulerRotation2((string)(i));
                const float e3 = BlackTraxBeaconGetEulerRotation3((string)(i));
                WriteText("Beacon " + (string)(i) + " Euler | eo "
                + (string)eo + " | e1 " +(string)e1 + " | e2 " +(string)e2
                + " | e3 " +(string)e3 + " | Latency " +(string)le);

                // ACCELERATION
                const float ax = BlackTraxBeaconGetAccelerationX((string)(i));
                const float ay = BlackTraxBeaconGetAccelerationY((string)(i));
                const float az = BlackTraxBeaconGetAccelerationZ((string)(i));
                WriteText("Beacon " + (string)(i) + " Acceleration | ax "
                + (string)ax + " | ay " +(string)ay + " | az " +(string)z);

                // VELOCITY
                const float vx = BlackTraxBeaconGetVelocityX((string)(i));
                const float vy = BlackTraxBeaconGetVelocityY((string)(i));
                const float vz = BlackTraxBeaconGetVelocityZ((string)(i));
                WriteText("Beacon " + (string)(i) + " Velocity | vx "
                + (string)vx + " | vy " +(string)vy + " | vz " +(string)vz);
        }
  }
}

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

MADRIX LIGHTING CONTROL

//PART E

Macros For Effects

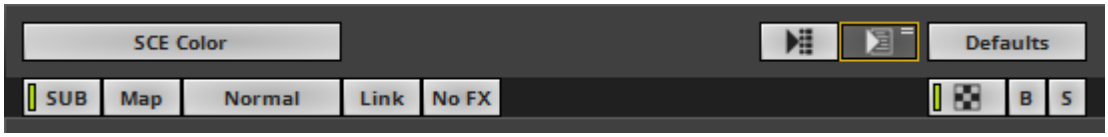
5 Macros For Effects

5.1 General Resources

5.1.1 Overview (Macros For Effects)

Introduction

- Macros for MADRIX Effects can be used to modify, manipulate, and affect the visual outcome of an effect or add automatization to it.
- Effect Macros are bound to a particular effect due to the different effects and its different functions and parameters.
- Effect Macros are available for every single MADRIX Effect, incl. SCE, S2L, M2L, TRI, as well as MAS.
- Effect Macros are stored as a part of the effect. This means they are part of a stored Storage Place file or Setup file.
- It is possible to save macros as separate files. The file extension of a macro is **.mms*. The extension of a crypted macro is **.mcm*.



Editor - Controls the corresponding Macro Editor.

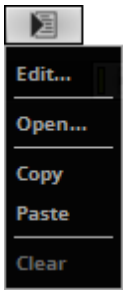
Macro - Allows you to configure and manage a Macro. At the same time this means that no Macro is running or included.

Deactivated [Macro Included] - A macro is included/inserted in the Macro Editor, but the macro is currently not running.

Left Mouse Click - Click once to activate the Macro.

Activated - A macro has been compiled and is currently running.

Left Mouse Click - Click once to deactivate the Macro.



Right Mouse Click - Opens the context menu.

Edit... - Opens the Macro Editor to write, edit, include, and compile Macros.

Open... - Allows you to load a Macro from an external file [of the file type *.mms and *.mcm]. Once loaded, the Macro will automatically be activated.

Copy - Allows you to copy the Macro to the clipboard of the computer.

Paste - Allows you to paste the Macro from the clipboard into the currently selected Macro Editor. If the copied Macro is running, the new Macro will be automatically activated as well. If the copied Macro is deactivated, the new Macro will be automatically deactivated as well.

Clear - Deletes all content of the Macro Editor and thereby deactivates and erases any Macro.

Functions Called By MADRIX

Overview

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()`

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 information for the developer of the script.

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 array 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 arrays, or whatever is necessary to (re)start the macro.

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.

PostRenderEffect

(Automatically included in a new macro)

This function is called after the effect has been rendered. Here, the result of the effect can be manipulated. You could use a gray filter on it, for example.

> **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.
>
> **Also note**: Mapping operations done in *PostRenderEffect* will affect the next frame, but not the current one. To control the current frame, please use *PreRenderEffect*.

MatrixSizeChanged

(Automatically included in a new macro)

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 Effect Macro Editor, the empty standard macro will look like this:

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

void InitEffect()
{

}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

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

5.1.2 Functions (Macros For Effects)

Specific Resources

- »Functions called by MADRIX
- Macros for Effects: Available Functions
- »Effect Parameter Chaser

- Using the Frame ID
- Applying Asynchronous Rendering
- Applying Stepped Rendering
- Applying Filter Effects (FX)
- Setting And Getting Blind Mode Or Solo Mode

- »Static Color Effects Overview (SCE)
- »Sound2Light Effects Overview (S2L)
- »Music2Light Effects Overview (M2L)

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

- For non-specific functions, see »List of Functions (Alphabetical Order)

Effect-Specific Functions Depending On The Effect

- Each MADRIX Effect is different. In addition to the non-specific functions, each effect has therefore its own settings and specific functions.

- Such functions are not working in other effects. A macro which uses such commands cannot be compiled in another effect.
- A separate topic is provided for each MADRIX Effect.

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(fmod(GetFrameId() + 2, GetFrameCount()));
}
```

However, it is not possible to set the Frame Count by a macro; except for a macro of the MAS Script Effect.

Learn more »**Using Frames**

Applying Asynchronous Rendering

- Normally, the effects are synchronized to the Main Mixing frame rate, which is set to 50 Hz by default.
- But for the »**MAS Script** effect, there is the option to control the render frequency independently.
- Learn more »**Using Frames**

Applying Stepped Rendering

Normally, the effects use floating-point frames which is represented by the rational return values of the functions GetFrameId and GetFrameSteps. That way, the effects are able to render very precisely and smooth. However, sometimes it is desired to let the effects "step" pixel by pixel. This option is only available for some effects, for example »**SCE Color Scroll** and »**MAS Script**

- **`void SetStep(int value)`**

- `int GetStep()`
- `void ToggleStep()`

Valid values for *value* are *0* (*Off*) or *1* (*On*). If stepped rendering is activated, effects like SCE Color Scroll look hard-stepping. In the MAS Script effect, stepped rendering causes GetFrameId and GetFrameSteps to return only integer values.

Learn more »**Using Frames**

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 And Getting Blind Mode Or Solo Mode

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.

- `void SetBlind(int value)`
- `int GetBlind()`
- `void ToggleBlind()`
- `void SetSolo(int value)`
- `int GetSolo()`
- `void ToggleSolo()`

Valid values for *value* are *0* (*Off*) or *1* (*On*).

5.1.3 Effect Parameter Chaser

Functions Provided For Using The Effect Parameter Chaser

Function	Description
int **ChaserGetPlaybackState**()	Returns the current playback state of the Chaser. See below for a list of constants.
void **ChaserPlay**()	Starts Chaser playback.
void **ChaserStop**()	Stops Chaser playback.
void **ChaserPause**()	Pauses Chaser playback.
int **ChaserGetCurrentStep**()	Returns the index of the Step that is currently active. Indexing starts with *0*.
void **ChaserCallCurrentStep**(int index)	Activates the Step specified with *index*. Indexing starts with *0*.
void **ChaserCallPreviousStep**(int offset)	Returns to the ***Previous Step***. The parameter *offset* allows to define how many Steps are skipped. By default, *offset* is set to *1* and does not need to be passed.
void **ChaserCallNextStep**(int offset)	Skips to the ***Next Step***. The parameter *offset* allows to define how many Steps are skipped. By default, *offset* is set to *1* and does not need to be passed.
void **ChaserGetStepCount**(int count)	Returns the current number of Steps.
void **ChaserSetLoopCount**(int count)	Sets the ***Loop Count***. Valid values range from *0* to *100*. A value of *0* means ***Endless***.
int **ChaserGetLoopCount**()	Returns the currently set ***Loop Count***. A value of *0* means ***Endless***.
int **ChaserGetCurrentLoop**()	Returns in which loop the Chaser is currently in.
float **ChaserGetStepProgress**()	Returns the progression of the current Step in percent. Valid values range from *0.0* to *100.0*.
float **ChaserGetLoopProgress**()	Returns the progression of the Loop in percent. Valid values range from *0.0* to *100.0*.
float **ChaserGetStepTime**(int index)	Returns the total ***Step Time*** of the Step specified with *index* in seconds. Indexing starts with *0*.
float **ChaserGetLoopTime**()	Returns the total ***Loop Time*** of the Chaser in seconds. Indexing starts with *0*.
void **ChaserSetStepDescription**(int index, string description)	Sets the ***Description*** of the Step specified with *index*. Indexing starts with *0*.
string **ChaserGetStepDescription**(int index)	Returns the ***Description*** of the Step specified with *index*. Indexing starts with *0*.
void **ChaserSetStepFadeType**(int index, int type)	Sets the ***Fade Type*** of the Step specified with *index*. Indexing starts with *0*. See below for a list of constants for *type*.
int **ChaserGetStepFadeType**(int index)	Returns the ***Fade Type*** of the Step specified with *index*. Indexing starts with *0*.
void **ChaserSetStepFadeTime**(int index, float time)	Sets the ***Fade Time*** of the Step specified with *index*, in seconds. Indexing starts with *0*.
float **ChaserGetStepFadeTime**(int index)	Returns the ***Fade Time*** of the Step specified with *index*, in seconds. Indexing starts with *0*.

void **ChaserSetStepWaitTime**(int index, float time)	Sets the ***Wait Time*** of the Step specified with *index*, in seconds. Indexing starts with *0*.
float **ChaserGetStepWaitTime**(int index)	Returns the ***Wait Time*** of the Step specified with *index*, in seconds. Indexing starts with *0*.
void **ChaserInvert**()	Inverts the Step list.
void **ChaserSwapSteps**(int index1, int index2)	Swaps the positions of the Steps specified with *index1* and *index2*. Indexing starts with *0*.
void **ChaserMoveStepUp**(int index)	Moves the Step specified with *index* one item up. Indexing starts with *0*.
void **ChaserMoveStepDown**(int index)	Moves the Step specified with *index* one item down. Indexing starts with *0*.
void **ChaserSetPlaybackMode**(int mode)	Sets the ***Playback Mode*** of the Chaser. See **below** for a list of constants for *mode*.
int **ChaserGetPlaybackMode**()	Returns the currently set ***Playback Mode***.
void **ChaserSetSpeedPitch**(float value)	Sets the ***Pitch*** *value* of the Chaser. Valid values range from *-10.00* to *10.00*.
float **ChaserGetSpeedPitch**()	Returns the currently set ***Pitch***.
void **ChaserSetAutostart**(int value)	Use 1 (**TRUE**) to activate the ***Automatic Start*** of the Chaser. Use 0 (**FALSE**) to deactivate it.
int **ChaserGetAutostart**()	Returns 1 (**TRUE**) if the ***Automatic Start*** of the Chaser is activated, otherwise 0 (**FALSE**).
void **ChaserToggleAutostart**()	Activates or deactivates the ***Automatic Start*** of the Chaser, depending on the current state.

Playback State Constants

Constant	Description
int **PLAYBACK_PLAYING**	Indicates that the Chaser is currently playing.
int **PLAYBACK_STOPPED**	Indicates that the Chaser is currently stopped.
int **PLAYBACK_PAUSED**	Indicates that the Chaser is currently paused.
int **PLAYBACK_FADING**	Indicates that the Chaser is currently transitioning from one Step to the next after having been triggered manually.

Interpolation Type Constants

Constant	Description

int **INTERPOLATION_TYPE_LINEAR**	Sets the Interpolation Type to ***Linear***.
int **INTERPOLATION_TYPE_EASE_BOUNCE_IN**	Sets the Interpolation Type to ***Ease In Bounce***.
int **INTERPOLATION_TYPE_EASE_BOUNCE_OUT**	Sets the Interpolation Type to ***Ease Out Bounce***.
int **INTERPOLATION_TYPE_EASE_BOUNCE_INOUT**	Sets the Interpolation Type to ***Ease In Out Bounce***.
int **INTERPOLATION_TYPE_EASE_CIRC_IN**	Sets the Interpolation Type to ***Ease In Circular***.
int **INTERPOLATION_TYPE_EASE_CIRC_OUT**	Sets the Interpolation Type to ***Ease Out Circular***.
int **INTERPOLATION_TYPE_EASE_CIRC_INOUT**	Sets the Interpolation Type to ***Ease In Out Circular***.
int **INTERPOLATION_TYPE_EASE_CUBIC_IN**	Sets the Interpolation Type to ***Ease In Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_OUT**	Sets the Interpolation Type to ***Ease Out Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_INOUT**	Sets the Interpolation Type to ***Ease In Out Cubic***.
int **INTERPOLATION_TYPE_EASE_SINE_IN**	Sets the Interpolation Type to ***Ease In Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_OUT**	Sets the Interpolation Type to ***Ease Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_INOUT**	Sets the Interpolation Type to ***Ease In Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_EXPO_IN**	Sets the Interpolation Type to ***Ease In Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_OUT**	Sets the Interpolation Type to ***Ease Out Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_INOUT**	Sets the Interpolation Type to ***Ease In Out Exponential*** .

Playback Mode Constants

The Chaser uses various playback modes. The function **ChaserSetPlaybackMode** can be used to change them. The following values can be used as parameter:

Constant	Description
int **PLAYBACK_MODE_FORWARD**	Sets the Playback Mode to ***Forward***.

int **PLAYBACK_MODE_BACKWARD**	Sets the Playback Mode to ***Backward***.
int **PLAYBACK_MODE_PING_PONG**	Sets the Playback Mode to ***Ping Pong***.
int **PLAYBACK_MODE_SHUFFLE**	Sets the Playback Mode to ***Shuffle***.

5.1.4 Using BPM Control

Functions Provided For Setting The Speed

The following table provides an overview of all functions to modify the speed:

Function	Description
void **SetBpm**(float value)	Sets the ***BPM*** *value*, which is the speed of the effect. Valid values for *value* range from *0.0* to *9999.0*. Setting the BPM to *0.0* means that the effect will be stopped immediately.
float **GetBpm**()	Returns the current ***BPM***.
void **SetSpeedPitch**(float value)	Sets the ***Pitch*** *value*, which defines an additional multiplicative factor for the speed of the effect. Valid values for *value* range from *-10.0* to *10.0*. Setting the pitch lower than *0.0* lets most effects run backwards.
float **GetSpeedPitch**()	Returns the current ***Pitch***.

Remarks

Each effect which uses the BPM Control will interpret the speed in its own way.

*- For example, 60 BPM in the »**SCE Color Change** effect will change the color 60 times per minute (once per second).*

*- 60 BPM in the »**SCE Pulse / Stroboscope** effect will flash the matrix 60 times per minute, too.*

*- For the »**SCE Shapes** and the »**SCE Tubes** effect, a BPM value of 60 means that 60 shapes/tubes will be created every minute by default.*

However, other effects apply the range of BPM values differently. Just try the BPM slider to get a feeling for how the effects interpret the speed.

*- For example, 3000 BPM in the »**M2L Drops** effect cause each drop to move over the whole matrix in one second.*

*- 3000 BPM in the »**SCE Graph** effect mean that the function will oscillate once per second.*

Please note that both the BPM and the BPM pitch are floating-point values. Thus, you can create very slow effects if you use values between 0.0 and 1.0.

Example

This sample macro causes the BPM slider to stick to multiples of 60 BPM. It works with the »SCE_Color_Change effect, for example.

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

const float step = 60.0; // use multiples of 60 BPM
float internalBpm = 0.0;
float previousBpm = 0.0;

void InitEffect()
{
 internalBpm = GetBpm();
 previousBpm = internalBpm;
}

void PreRenderEffect()
{
 internalBpm += (GetBpm() - previousBpm);
//      WriteText("internalBpm: " + (string)internalBpm);

 int times = (int)round(internalBpm / step);
 SetBpm((float)times * step);

 previousBpm = GetBpm();
}

void PostRenderEffect()
{

}

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

5.1.5 Using Center Control

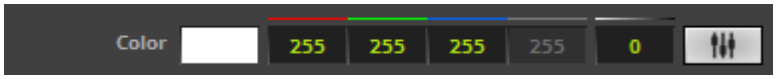
Functions Provided For Setting The Center

The following table provides an overview of all functions the effect can use to modify the center:

Function	Description
void **SetCenterX**(float value)	Sets the ***Center X*** *value* in percent of the matrix size.
float **GetCenterX**()	Returns the current ***Center X*** in percent of the matrix size.
void **SetCenterY**(float value)	Sets the ***Center Y*** *value* in percent of the matrix size.
float **GetCenterY**()	Returns the current ***Center Y*** in percent of the matrix size.
void **SetCenterZ**(float value)	Sets the ***Center Z*** *value* in percent of the matrix size.
float **GetCenterZ**()	Returns the current ***Center Z*** in percent of the matrix size.
void **SetCenter**(float x, float y, float z)	Sets the center values ***X, Y, Z*** in percent of the matrix size.
void **SetPixelCenterX**(int value)	Sets the ***Center X*** *value* in pixels.
int **GetPixelCenterX**()	Returns the current ***Center X*** in pixels.
void **SetPixelCenterY**(int value)	Sets the ***Center Y*** *value* in pixels.
int **GetPixelCenterY**()	Returns the current ***Center Y*** in pixels.
void **SetPixelCenterZ**(int value)	Sets the ***Center Z*** *value* in pixels.
int **GetPixelCenterZ**(void)	Returns the current ***Center Z*** in pixels.
void **SetPixelCenter**(int x, int y, int z)	Sets the center values ***X, Y, Z*** in pixels.
void **SetCenterMode**(int)	Sets the mode. Valid values include **CENTER_MODE_FIXED** and **CENTER_MODE_RANDOM**.
int **GetCenterMode**()	Returns the currently set center mode.

5.1.6 Using Color Controls

Functions Provided For Using Colors



Some effects, such as »SCE Color or »SCE Wave / Radial, 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***.

- For a detailed description of the non-primitive data type color, see »Using Data Types
- Learn more »List Of Global Variables And Constants

Example

This macro example sets a random generated color each time the effect is started.

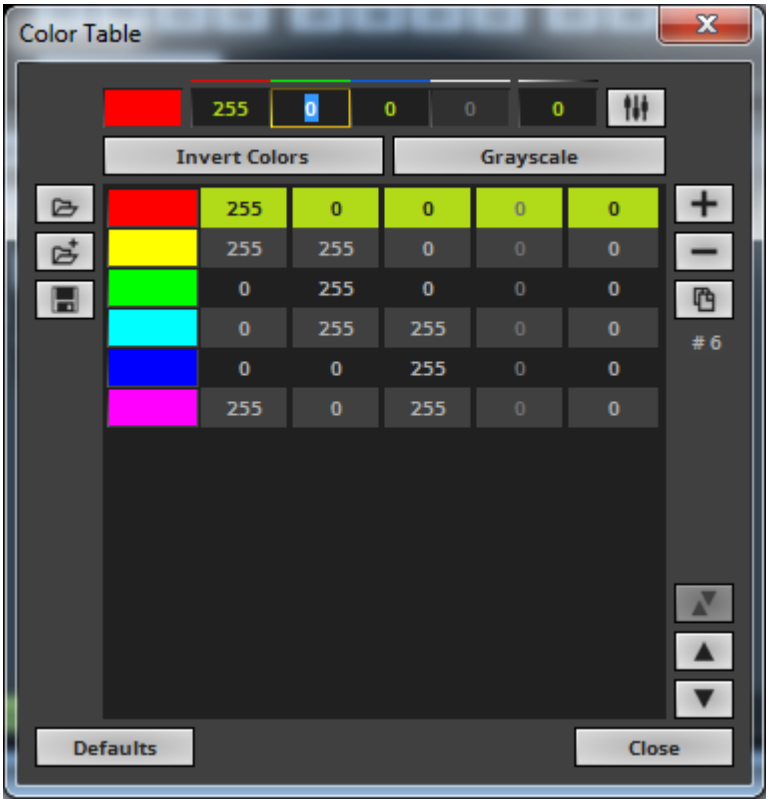
```
@scriptname="";
@author="";
@version="";
@description="";

void InitEffect()
{
 color c;
 c.r = random(0,255);
 c.g = random(0,255);
 c.b = random(0,255);
 c.w = 0;
 c.a = 0;

 SetColor(c);
}

void PreRenderEffect()
{

}
```

```
void PostRenderEffect()
{

}

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

5.1.7 Using Color Table

Functions Provided For Using The Color Table

Some effects provide more than one color. In this case the so-called Color Table (described here), »Color Gradient, »Color Gradient Dialog or »M2L Color Table is offered by the effect to take control of the colors.



The following table provides an overview of all functions the effect can use to modify the colors in a Color Table:

Function	Description
void **ColorTableSetColor**(int index, color c)	Sets the color with the specified *index* to the given color value. If the index is out of range, nothing happens.
color **ColorTableGetColor**(int index)	Returns the color with the specified *index* in the Color Table. If the index is out of range, black is returned.
int **ColorTableGetColorCount**()	Returns the current number of colors in the Color Table.

void **ColorTableInvert**()	Inverts the complete Color Table regarding the positions of the colors in the table.
void **ColorTableSwapColors**(int index1, int index2)	Swaps colors with their specified *indices* in the Color Table.
void **ColorTableAddColor**(int index, color c)	Adds another color to the Color Table at the specified *index* position. If the index is 0, the new color is added to the first position. If the index is equal to or greater than the current number of colors, the new color is added at the end.
void **ColorTableRemoveColor**(int index)	Removes the color at the specified *index*. If the given index is out of range, nothing happens.
void **ColorTableSetColorFade**(int fade)	Use 1 (**TRUE**) to activate the color ***Fade*** mode. Use 0 (**FALSE**) to deactivate it.
int **ColorTableGetColorFade**()	Returns 1 (**TRUE**) if the color ***Fade*** mode is activated, otherwise 0 (**FALSE**).
void **ColorTableToggleColorFade**()	Toggles the color ***Fade*** mode.
void **ColorTableSetColorMode**(int mode)	Sets the ***Color Mode***. Please use a constant **as described below** for *mode*.
int **ColorTableGetColorMode**()	Returns the current ***Color Mode***.

- For a detailed description of the non-primitive data type color, see »**Using Data Types**

Remarks

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 Macro. In some MADRIX effects you cannot add or delete the number of colors, only the colors itself.

Color Mode Constants

Constant	Description
int COLOR_MODE_LOOP	The colors of the list will be used one after another in a loop.
int COLOR_MODE_SHUFFLE	The colors of the list will be used without a specific order.
int COLOR_MODE_RANDOM	The effect creates random colors.

Example

This example ensures the amount of 6 colors in the Color Table and sets this 6 colors to defined values. The color fade and the loop mode are set. This example works with the »SCE Color Change or »SCE Color Scroll effect, for example.

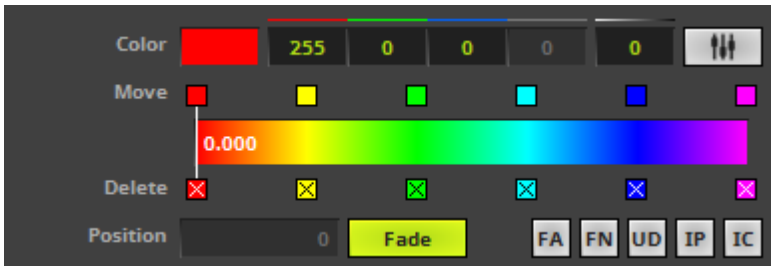
```
@scriptname="";
@author="";
@version="";
@description="";

//Colors
color c1 = {255,0,0}; //red
color c2 = {0,255,0}; //green
color c3 = {0,0,255}; //blue

void InitEffect()
{
 //remove all colors from list until the amount of colors is 6
 while(ColorTableGetColorCount() > 6)
        ColorTableRemoveColor(0);

 //add colors to list until the amount of colors is 6
 while(ColorTableGetColorCount() < 6)
        ColorTableAddColor(0,WHITE);
```

```
  //set colors
  ColorTableSetColor(0,c1);
  ColorTableSetColor(1,c2);
  ColorTableSetColor(2,c3);
  ColorTableSetColor(3,YELLOW);
  ColorTableSetColor(4,ORANGE);
  ColorTableSetColor(5,PINK);

  //activate color fade
  ColorTableSetColorFade(true);

  //set loop mode
  ColorTableSetColorMode(COLOR_MODE_LOOP);

}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
}

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

5.1.8 Using Color Gradient

Functions Provided For Using The Color Gradient

Some effects provide more than one color. In this case the so-called »Color Table, Color Gradient (described here), » Color Gradient Dialog or »M2L Color Table is offered by the effect to take control of the colors.



The following table provides an overview of all functions the effect can use to modify the colors in a Color Gradient:

Function	Description
void **GradientSetColor**(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 **GradientGetColor**(int idx)	Returns the color with the specified index. If the index is out of range, *black* is returned.
int **GradientGetColorCount**()	Returns the current number of colors in the Color Gradient.
void **GradientAddColor**(color c, float position, int fade)	Adds another color to the Color Gradient at the specified position. The position value have to be between *0.0* and *1.0*. The inserted color will be faded the color before if the fade value is 1 (**TRUE**).
void **GradientRemoveColor**(int idx)	Removes the color at the specified index. If the given index is out of range, nothing happens.
void **GradientSetColorPosition**(int idx, float pos)	Sets the color with the specified index to the given position value. The position value have to be between *0.0* and *1.0*. If the index is out of range, nothing happens.
float **GradientGetColorPosition**(int idx)	Returns the position of the color with the specified index. If the index is out of range, 0.0 is returned.
void **GradientSetColorFade**(int idx, int fade)	Use 1 (**TRUE**) to activate the color fade mode for the color with the specified index. Use 0 (**FALSE**) to deactivate it.
int **GradientGetColorFade**(int idx)	Returns 1 (**TRUE**) if the color fade mode is activated for the color with the specified index, otherwise 0 (**FALSE**).
void **GradientFadeAllColors**(void)	Activates the color fade mode for all colors in the Color Gradient.
void **GradientFadeNoneColors**(void)	Deactivates the color fade mode for all colors in the Color Gradient.

void **GradientSetUniformDistances**(void)	Sets the position values of each color in the Color Gradient to get uniform distances between each color.
void **GradientInvertColorPositions**(void)	Inverts the position values off all colors.
void **GradientInvertColors**(void)	Inverts all color values.

- For a detailed description of the non-primitive data type color, see »**Using Data Types**

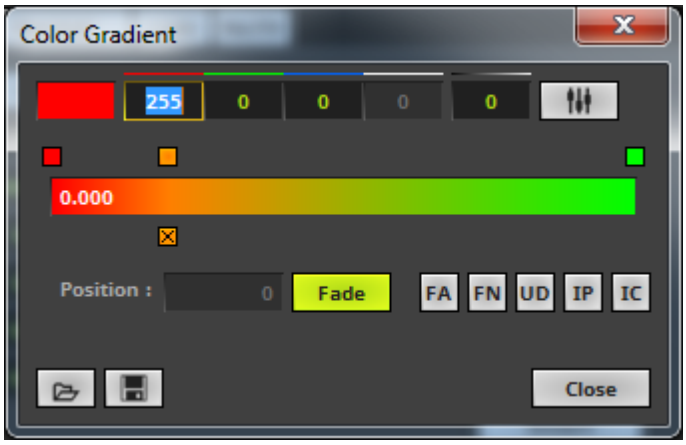
Remarks

The Color Gradient requires at least 2 colors. You will not be able to overwrite this requirement with a Macro. Some MADRIX effects require the same color at the first and the last position.

5.1.9 Using Color Gradient Dialog

Functions Provided For Setting Effect Directions

Some effects provide more than one color. In this case the so-called »Color Table, »Color Gradient, Color Gradient Dialog (described here) or »M2L Color Table is offered by the effect to take control of the colors.



The following table provides an overview of all functions the effect can use to modify the colors in a Color Gradient Dialog:

Function	Description
void **GradientDlgSetColor**(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 **GradientDlgGetColor**(int idx)	Returns the color with the specified index. If the index is out of range, *black* is returned.
int **GradientDlgGetColorCount**()	Returns the current number of colors in the Color Gradient Dialog.
void **GradientDlgAddColor**(color c, float position, int fade)	Adds another color to the Color Gradient Dialog at the specified position. The position value have to be between *0.0* and *1.0*. The inserted color will be faded the color before if the fade value is 1 (**TRUE**) .
void **GradientDlgRemoveColor**(int idx)	Removes the color at the specified index. If the given index is out of range, nothing happens.
void **GradientDlgSetColorPosition**(int idx, float pos)	Sets the color with the specified index to the given position value. The position value have to be between *0.0* and *1.0*. If the index is out of range, nothing happens.
float **GradientDlgGetColorPosition**(int idx)	Returns the position of the color with the specified index. If the index is out of range, 0.0 is returned.

void **GradientDlgSetColorFade**(int idx, int fade)	Use 1 (**TRUE**) to activate the color fade mode for the color with the specified index. Use 0 (**FALSE**) to deactivate it.
int **GradientDlgGetColorFade**(int idx)	Returns 1 (**TRUE**) if the color fade mode is activated for the color with the specified index, otherwise 0 (**FALSE**).
void **GradientDlgFadeAllColors**(void)	Activates the color fade mode for all colors in the Color Gradient Dialog.
void **GradientDlgFadeNoneColors**(void)	Deactivates the color fade mode for all colors in the Color Gradient Dialog.
void **GradientDlgSetUniformDistances**(void)	Sets the position values of each color in the Color Gradient Dialog to get uniform distances between each color.
void **GradientDlgInvertColorPositions**(void)	Inverts the position values off all colors.
void **GradientDlgInvertColors**(void)	Inverts all color values.

- For a detailed description of the non-primitive data type color, see »**Using Data Types**

Remarks

The Color Gradient Dialog requires at least 2 colors. You will not be able to overwrite this requirement with a Macro. Some MADRIX effects require the same color at the first and last position.

5.1.10 Using M2L Color Table

Functions Provided For Using The M2L Color Table

Some effects provide more than one color. In this case the so-called »Color_Table, »Color_Gradient, »Color Gradient Dialog or M2L Color Table (described here) is offered by the effect to take control of the colors.

The following table provides an overview of all functions the effect can use to modify the colors in a M2L Color Table:

Function	Description
void **M2LColorTableSetColor**(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 **M2LColorTableGetColor**(int idx)	Returns the color with the specified *index*. If the index is out of range, *black* is returned.
void **M2LColorTableSetPreset**(int preset)	Sets the M2L Color Table preset. Please use a constant as described below for *preset*.
void **M2LColorTableSetColorFade**(int enable)	Sets the ***Fade*** option. Use 1 (**TRUE**) to activate the fade. Use 0 (**FALSE**) to deactivate it.
int **M2LColorTableGetColorFade**()	Returns 1 (**TRUE**) if ***Fade*** is activated, otherwise 0 (**FALSE**).
void **M2LColorTableToggleColorFade**()	Toggles the ***Fade*** option.

- For a detailed description of the non-primitive data type color, see »Using Data Types

Remarks

Not every function might be available for each MADRIX effect. Therefore, the functions M2LColorTableSetColorFade, M2LColorTableGetColorFade, and M2LColorTableToggleColorFade are only available if the effect offers Fade Mode.

M2L Color Table Preset Constants

Constant	Description
int M2L_COLORTABLE_PRESET_C1	Sets preset ***C1***.
int M2L_COLORTABLE_PRESET_C2	Sets preset ***C2***.
int M2L_COLORTABLE_PRESET_R	Sets preset ***R***.
int M2L_COLORTABLE_PRESET_G	Sets preset ***G***.
int M2L_COLORTABLE_PRESET_B	Sets preset ***B***.

Example

This example sets the M2L Color Table preset C1 and sets the last 12 colors to color black.

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

void InitEffect()
{
 //Set Preset C1
 M2LColorTableSetPreset(M2L_COLORTABLE_PRESET_C1);

 //Sets the last 12 colors to black
 for(int i=12;i<24;i++)
        M2LColorTableSetColor(i,BLACK);
}

void PreRenderEffect()
```

```
{

}

void PostRenderEffect()
{

}

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

5.1.11 Using Image Table

Functions Provided For Using The Image List

The following table provides an overview of all functions the effect can use to modify the images in an Image Table:

Function	Description
int **ImageListGetCount**()	Returns the current number of images in the Image Table.
void **ImageListMoveImageUp**(int index)	Moves an image with its specified *index* up a place in the Image Table.
void **ImageListMoveImageDown**(int index)	Moves an image with its specified *index* down a place in the Image Table.
void **ImageListInvert**()	Inverts the complete Image Table regarding the positions of the images in the table.
void **ImageListSwapImage**(int index1, int index2)	Swaps images with their specified *indices* in the Image Table.
void **ImageListSetImageRotation**(int index, int rotation)	Sets the image ***Rotation*** mode for an image with the specified *index* in the Image Table. See **below** for details.
int **ImageListGetImageRotation**(int index)	Returns the image ***Rotation*** mode for an image with the specified *index* in the Image Table.
void **ImageListSetImageDuration**(int index, float duration)	Sets the ***Duration*** for an image with the specified *index* in the Image Table. Note that the ***Fade Time*** is a part of the ***Duration***.
float **ImageListGetImageDuration**(int index)	Returns the ***Duration*** for an image with the specified *index* in the Image Table.
void **ImageListSetImageFadeTime**(int index, float fadetime)	Sets the ***Fade Time*** for an image with the specified *index* in the Image Table. Note that the ***Fade Time*** is a part of the ***Duration***.
float **ImageListGetImageFadeTime**(int index)	Returns the ***Fade Time*** for an image with the specified *index* in the Image Table.

Rotation Mode Constants

Constant	Description
int **ROTATION_CCW_0**	Does not activate Rotation. Is the same as **ROTATION_CW_0**.
int **ROTATION_CCW_90**	Rotates by ***90 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_270**.
int **ROTATION_CCW_180**	Rotates by ***180 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_180**.
int **ROTATION_CCW_270**	Rotates by ***270 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_90**.
int **ROTATION_CW_0**	Does not activate Rotation. Is the same as **ROTATION_CCW_0**.
int **ROTATION_CW_90**	Rotates by ***90 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_270**.
int **ROTATION_CW_180**	Rotates by ***180 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_180**.
int **ROTATION_CW_270**	Rotates by ***270 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_90**.

5.1.12 Using Shapes

Functions Provided For Setting Shapes

The following table provides an overview of all functions to modify shapes:

Function	Description
void **SetRenderingMode**(int mode)	Sets the ***Rendering Mode***. Please use a constant as described below for *mode*.
int **GetRenderingMode**()	Returns the currently set ***Rendering Mode***.
void **SetShapeType**(int value)	Sets the ***Shape Type***. Please use a constant as described below for *value*.
int **GetShapeType**()	Returns the currently set ***Shape Type***.
void **SetShapeAlignment**(int value)	Sets the ***Shape Alignment***, which is the orientation of the shapes related to the matrix. Please use a constant as described below for *value*.
int **GetShapeAlignment**()	Returns the currently set ***Shape Alignment***.
void **SetShapeRotation**(int value)	Sets the ***Rotation Type***. Please use a constant as described below for *value*.
int **GetShapeRotation**()	Returns the currently set ***Rotation Type***.

void **SetShapeOrigin**(int type)	Sets the ***Origin Type***. Please use a constant as described below for *type*.
int **GetShapeOrigin**()	Returns the currently set ***Origin Type***.

Remarks

Not all effects which use shapes offer Rendering Mode. Therefore, the functions SetRenderingMode and GetRenderingMode are only available if the effect offers Rendering Mode.

Not all effects which use shapes offer Shape Alignment. Therefore, the functions SetShapeAlignment and GetShapeAlignment are only available if the effect offers Shape Alignment.

Not all effects which use shapes offer Shape Rotation. Therefore, the functions SetShapeRotation and GetShapeRotation are only available if the effect offers Shape Rotation.

Not all effects which use shapes offer Origin. Therefore, the functions SetShapeOrigin and GetShapeOrigin are only available if the effect offers Origin.

Also, the effects mostly offer just a selection of all the constants, which are listed below. The function SetRenderingMode will only set Rendering Modes which are offered by the effect. The function SetShapeType will only set Shape Types which are offered by the effect. The function SetShapeAlignment will only set Shape Alignments which are offered by the effect.

Rendering Mode Constants

Constant	Description
int **RENDERING_MODE_SIMPLE**	Sets the Rendering Mode to ***Simple***.
int **RENDERING_MODE_EXTENDED**	Sets the Rendering Mode to ***Extended***.
int **RENDERING_MODE_BLOBBY**	Sets the Rendering Mode to ***Blobby***.

Shape Type Constants

Constant	Description
int **SHAPE_TYPE_LINE**	Sets the Shape Type to ***Line***.
int **SHAPE_TYPE_CURVE**	Sets the Shape Type to ***Curve***.
int **SHAPE_TYPE_FILLED**	Sets the Shape Type to ***Filled***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED**	Sets the Shape Type to ***Rectangle Outlined***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Rectangle Outlined Implode***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Rectangle Outlined Explode***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Rectangle Outlined Pulse***.
int **SHAPE_TYPE_RECTANGLE_FILLED**	Sets the Shape Type to ***Rectangle Filled***.
int **SHAPE_TYPE_RECTANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Rectangle Filled Implode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Rectangle Filled Explode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_PULSE**	Sets the Shape Type to ***Rectangle Filled Pulse***.
int **SHAPE_TYPE_SQUARE_OUTLINED**	Sets the Shape Type to ***Square Outlined*** .
int **SHAPE_TYPE_SQUARE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Square Outlined Implode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Square Outlined Explode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_PULSE**	Sets the Shape Type to ***Square Outlined Pulse***.
int **SHAPE_TYPE_SQUARE_FILLED**	Sets the Shape Type to ***Square Filled***.
int **SHAPE_TYPE_SQUARE_FILLED_IMPLODE**	Sets the Shape Type to ***Square Filled Implode***.
int **SHAPE_TYPE_SQUARE_FILLED_EXPLODE**	Sets the Shape Type to ***Square Filled Explode***.
int **SHAPE_TYPE_SQUARE_FILLED_PULSE**	Sets the Shape Type to ***Square Filled Pulse***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED**	Sets the Shape Type to ***Ellipse Outlined***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Ellipse Outlined Implode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Ellipse Outlined Explode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_PULSE**	Sets the Shape Type to ***Ellipse Outlined Pulse***.
int **SHAPE_TYPE_ELLIPSE_FILLED**	Sets the Shape Type to ***Ellipse Filled***.

Constant	Description
int **SHAPE_TYPE_ELLIPSE_FILLED_IMPLODE**	Sets the Shape Type to ***Ellipse Filled Implode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_EXPLODE**	Sets the Shape Type to ***Ellipse Filled Explode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_PULSE**	Sets the Shape Type to ***Ellipse Filled Pulse***.
int **SHAPE_TYPE_CIRCLE_OUTLINED**	Sets the Shape Type to ***Circle Outlined***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Circle Outlined Implode***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Circle Outlined Explode***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_PULSE**	Sets the Shape Type to ***Circle Outlined Pulse***.
int **SHAPE_TYPE_CIRCLE_FILLED**	Sets the Shape Type to ***Circle Filled***.
int **SHAPE_TYPE_CIRCLE_FILLED_IMPLODE**	Sets the Shape Type to ***Circle Filled Implode***.
int **SHAPE_TYPE_CIRCLE_FILLED_EXPLODE**	Sets the Shape Type to ***Circle Filled Explode***.
int **SHAPE_TYPE_CIRCLE_FILLED_PULSE**	Sets the Shape Type to ***Circle Filled Pulse*** .
int **SHAPE_TYPE_DIAMOND_OUTLINED**	Sets the Shape Type to ***Diamond Outlined***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_IMPLODE**	Sets the Shape Type to ***Diamond Outlined Implode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_EXPLODE**	Sets the Shape Type to ***Diamond Outlined Explode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_PULSE**	Sets the Shape Type to ***Diamond Outlined Pulse***.
int **SHAPE_TYPE_DIAMOND_FILLED**	Sets the Shape Type to ***Diamond Filled***.
int **SHAPE_TYPE_DIAMOND_FILLED_IMPLODE**	Sets the Shape Type to ***Diamond Filled Implode***.
int **SHAPE_TYPE_DIAMOND_FILLED_EXPLODE**	Sets the Shape Type to ***Diamond Filled Explode***.
int **SHAPE_TYPE_DIAMOND_FILLED_PULSE**	Sets the Shape Type to ***Diamond Filled Pulse***.
int **SHAPE_TYPE_PENTAGON_OUTLINED**	Sets the Shape Type to ***Pentagon Outlined***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_IMPLODE**	Sets the Shape Type to ***Pentagon Outlined Implode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_EXPLODE**	Sets the Shape Type to ***Pentagon Outlined Explode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_PULSE**	Sets the Shape Type to ***Pentagon Outlined Pulse***.

Constant	Description
int **SHAPE_TYPE_PENTAGON_FILLED**	Sets the Shape Type to ***Pentagon Filled***.
int **SHAPE_TYPE_PENTAGON_FILLED_IMPLODE**	Sets the Shape Type to ***Pentagon Filled Implode***.
int **SHAPE_TYPE_PENTAGON_FILLED_EXPLODE**	Sets the Shape Type to ***Pentagon Filled Explode***.
int **SHAPE_TYPE_PENTAGON_FILLED_PULSE**	Sets the Shape Type to ***Pentagon Filled Pulse***.
int **SHAPE_TYPE_HEXAGON_OUTLINED**	Sets the Shape Type to ***Hexagon Outlined***.
int **SHAPE_TYPE_HEXAGON_OUTLINED_IMPLODE**	Sets the Shape Type to ***Hexagon Outlined Implode***.
int **SHAPE_TYPE_HEXAGON_OUTLINED_EXPLODE**	Sets the Shape Type to ***Hexagon Outlined Explode***.
int **SHAPE_TYPE_HEXAGON_OUTLINED_PULSE**	Sets the Shape Type to ***Hexagon Outlined Pulse***.
int **SHAPE_TYPE_HEXAGON_FILLED**	Sets the Shape Type to ***Hexagon Filled***.
int **SHAPE_TYPE_HEXAGON_FILLED_IMPLODE**	Sets the Shape Type to ***Hexagon Filled Implode***.
int **SHAPE_TYPE_HEXAGON_FILLED_EXPLODE**	Sets the Shape Type to ***Hexagon Filled Explode***.
int **SHAPE_TYPE_HEXAGON_FILLED_PULSE**	Sets the Shape Type to ***Hexagon Filled Pulse***.
int **SHAPE_TYPE_HEART_OUTLINED**	Sets the Shape Type to ***Heart Outlined***.
int **SHAPE_TYPE_HEART_OUTLINED_IMPLODE**	Sets the Shape Type to ***Heart Outlined Implode***.
int **SHAPE_TYPE_HEART_OUTLINED_EXPLODE**	Sets the Shape Type to ***Heart Outlined Explode***.
int **SHAPE_TYPE_HEART_OUTLINED_PULSE**	Sets the Shape Type to ***Heart Outlined Pulse***.
int **SHAPE_TYPE_HEART_FILLED**	Sets the Shape Type to ***Heart Filled***.
int **SHAPE_TYPE_HEART_FILLED_IMPLODE**	Sets the Shape Type to ***Heart Filled Implode***.
int **SHAPE_TYPE_HEART_FILLED_EXPLODE**	Sets the Shape Type to ***Heart Filled Explode***.
int **SHAPE_TYPE_HEART_FILLED_PULSE**	Sets the Shape Type to ***Heart Filled Pulse***.
int **SHAPE_TYPE_CROSS_OUTLINED**	Sets the Shape Type to ***Cross Outlined***.
int **SHAPE_TYPE_CROSS_OUTLINED_IMPLODE**	Sets the Shape Type to ***Cross Outlined Implode***.
int **SHAPE_TYPE_CROSS_OUTLINED_EXPLODE**	Sets the Shape Type to ***Cross Outlined Explode***.
int **SHAPE_TYPE_CROSS_OUTLINED_PULSE**	Sets the Shape Type to ***Cross Outlined Pulse***.

Constant	Description
int **SHAPE_TYPE_CROSS_FILLED**	Sets the Shape Type to ***Cross Filled***.
int **SHAPE_TYPE_CROSS_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Filled Implode***.
int **SHAPE_TYPE_CROSS_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Filled Explode***.
int **SHAPE_TYPE_CROSS_FILLED_PULSE**	Sets the Shape Type to ***Cross Filled Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED**	Sets the Shape Type to ***Cross Straight Outlined***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_IMPLODE**	Sets the Shape Type to ***Cross Straight Outlined Implode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_EXPLODE**	Sets the Shape Type to ***Cross Straight Outlined Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_PULSE**	Sets the Shape Type to ***Cross Straight Outlined Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED**	Sets the Shape Type to ***Cross Straight Filled***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Straight Filled Implode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Straight Filled Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_PULSE**	Sets the Shape Type to ***Cross Straight Filled Pulse***.
int **SHAPE_TYPE_STAR_OUTLINED**	Sets the Shape Type to ***Star Outlined***.
int **SHAPE_TYPE_STAR_OUTLINED_IMPLODE**	Sets the Shape Type to ***Star Outlined Implode***.
int **SHAPE_TYPE_STAR_OUTLINED_EXPLODE**	Sets the Shape Type to ***Star Outlined Explode***.
int **SHAPE_TYPE_STAR_OUTLINED_PULSE**	Sets the Shape Type to ***Star Outlined Pulse***.
int **SHAPE_TYPE_STAR_FILLED**	Sets the Shape Type to ***Star Filled***.
int **SHAPE_TYPE_STAR_FILLED_IMPLODE**	Sets the Shape Type to ***Star Filled Implode***.
int **SHAPE_TYPE_STAR_FILLED_EXPLODE**	Sets the Shape Type to ***Star Filled Explode***.
int **SHAPE_TYPE_STAR_FILLED_PULSE**	Sets the Shape Type to ***Star Filled Pulse***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED**	Sets the Shape Type to ***Triangle Outlined***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Triangle Outlined Implode***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Triangle Outlined Explode***.

Constant	Description
int **SHAPE_TYPE_TRIANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Triangle Outlined Pulse***.
int **SHAPE_TYPE_TRIANGLE_FILLED**	Sets the Shape Type to ***Triangle Filled***.
int **SHAPE_TYPE_TRIANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Triangle Filled Implode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Triangle Filled Explode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_PULSE**	Sets the Shape Type to ***Triangle Filled Pulse***.
int **SHAPE_TYPE_BOX_OUTLINED**	Sets the Shape Type to ***3D Box Outlined*** .
int **SHAPE_TYPE_BOX_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Box Outlined Implode***.
int **SHAPE_TYPE_BOX_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Box Outlined Explode***.
int **SHAPE_TYPE_BOX_OUTLINED_PULSE**	Sets the Shape Type to ***3D Box Outlined Pulse***.
int **SHAPE_TYPE_BOX_UNFILLED**	Sets the Shape Type to ***3D Box Unfilled***.
int **SHAPE_TYPE_BOX_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Box Unfilled Implode***.
int **SHAPE_TYPE_BOX_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Box Unfilled Explode***.
int **SHAPE_TYPE_BOX_UNFILLED_PULSE**	Sets the Shape Type to ***3D Box Unfilled Pulse***.
int **SHAPE_TYPE_BOX_FILLED**	Sets the Shape Type to ***3D Box Filled***.
int **SHAPE_TYPE_BOX_FILLED_IMPLODE**	Sets the Shape Type to ***3D Box Filled Implode***.
int **SHAPE_TYPE_BOX_FILLED_EXPLODE**	Sets the Shape Type to ***3D Box Filled Explode***.
int **SHAPE_TYPE_BOX_FILLED_PULSE**	Sets the Shape Type to ***3D Box Filled Pulse***.
int **SHAPE_TYPE_CUBE_OUTLINED**	Sets the Shape Type to ***3D Cube Outlined***.
int **SHAPE_TYPE_CUBE_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Cube Outlined Implode***.
int **SHAPE_TYPE_CUBE_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Cube Outlined Explode***.
int **SHAPE_TYPE_CUBE_OUTLINED_PULSE**	Sets the Shape Type to ***3D Cube Outlined Pulse***.
int **SHAPE_TYPE_CUBE_UNFILLED**	Sets the Shape Type to ***3D Cube Unfilled*** .
int **SHAPE_TYPE_CUBE_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Unfilled Implode***.

Constant	Description
int **SHAPE_TYPE_CUBE_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Unfilled Explode***.
int **SHAPE_TYPE_CUBE_UNFILLED_PULSE**	Sets the Shape Type to ***3D Cube Unfilled Pulse***.
int **SHAPE_TYPE_CUBE_FILLED**	Sets the Shape Type to ***3D Cube Filled***.
int **SHAPE_TYPE_CUBE_FILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Filled Implode***.
int **SHAPE_TYPE_CUBE_FILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Filled Explode***.
int **SHAPE_TYPE_CUBE_FILLED_PULSE**	Sets the Shape Type to ***3D Cube Filled Pulse***.
int **SHAPE_TYPE_SPHERE_UNFILLED**	Sets the Shape Type to ***3D Sphere Unfilled***.
int **SHAPE_TYPE_SPHERE_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Sphere Unfilled Implode***.
int **SHAPE_TYPE_SPHERE_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Sphere Unfilled Explode***.
int **SHAPE_TYPE_SPHERE_UNFILLED_PULSE**	Sets the Shape Type to ***3D Sphere Unfilled Pulse***.
int **SHAPE_TYPE_SPHERE_FILLED**	Sets the Shape Type to ***3D Sphere Filled***.
int **SHAPE_TYPE_SPHERE_FILLED_IMPLODE**	Sets the Shape Type to ***3D Sphere Filled Implode***.
int **SHAPE_TYPE_SPHERE_FILLED_EXPLODE**	Sets the Shape Type to ***3D Sphere Filled Explode***.
int **SHAPE_TYPE_SPHERE_FILLED_PULSE**	Sets the Shape Type to ***3D Sphere Filled Pulse***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED**	Sets the Shape Type to ***3D Ellipsoid Unfilled***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Implode***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Explode***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_PULSE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Pulse***.
int **SHAPE_TYPE_ELLIPSOID_FILLED**	Sets the Shape Type to ***3D Ellipsoid Filled***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_IMPLODE**	Sets the Shape Type to ***3D Ellipsoid Filled Implode***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_EXPLODE**	Sets the Shape Type to ***3D Ellipsoid Filled Explode***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_PULSE**	Sets the Shape Type to ***3D Ellipsoid Filled Pulse***.
int **SHAPE_TYPE_OCTAHEDRON_UNFILLED**	Sets the Shape Type to ***3D Octahedron Unfilled***.

Constant	Description
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Implode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Explode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_PULSE	Sets the Shape Type to ***3D Octahedron Unfilled Pulse***.
int SHAPE_TYPE_OCTAHEDRON_FILLED	Sets the Shape Type to ***3D Octahedron Filled***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Filled Implode***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Filled Explode***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_PULSE	Sets the Shape Type to ***3D Octahedron Filled Pulse***.
int SHAPE_TYPE_3D_HEART_UNFILLED	Sets the Shape Type to ***3D Heart Unfilled***.
int SHAPE_TYPE_3D_HEART_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Heart Unfilled Implode***.
int SHAPE_TYPE_3D_HEART_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Heart Unfilled Explode***.
int SHAPE_TYPE_3D_HEART_UNFILLED_PULSE	Sets the Shape Type to ***3D Heart Unfilled Pulse***.
int SHAPE_TYPE_3D_HEART_FILLED	Sets the Shape Type to ***3D Heart Filled***.
int SHAPE_TYPE_3D_HEART_FILLED_IMPLODE	Sets the Shape Type to ***3D Heart Filled Implode***.
int SHAPE_TYPE_3D_HEART_FILLED_EXPLODE	Sets the Shape Type to ***3D Heart Filled Explode***.
int SHAPE_TYPE_3D_HEART_FILLED_PULSE	Sets the Shape Type to ***3D Heart Filled Pulse***.
int SHAPE_TYPE_3D_STAR_UNFILLED	Sets the Shape Type to ***3D Star Unfilled***.
int SHAPE_TYPE_3D_STAR_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Star Unfilled Implode***.
int SHAPE_TYPE_3D_STAR_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Star Unfilled Explode***.
int SHAPE_TYPE_3D_STAR_UNFILLED_PULSE	Sets the Shape Type to ***3D Star Unfilled Pulse***.
int SHAPE_TYPE_3D_STAR_FILLED	Sets the Shape Type to ***3D Star Filled***.
int SHAPE_TYPE_3D_STAR_FILLED_IMPLODE	Sets the Shape Type to ***3D Star Filled Implode***.
int SHAPE_TYPE_3D_STAR_FILLED_EXPLODE	Sets the Shape Type to ***3D Star Filled Explode***.
int SHAPE_TYPE_3D_STAR_FILLED_PULSE	Sets the Shape Type to ***3D Star Filled Pulse***.

Constant	Description
int SHAPE_TYPE_3D_CROSS_UNFILLED	Sets the Shape Type to ***3D Cross Unfilled***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Cross Unfilled Implode***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Cross Unfilled Explode***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_PULSE	Sets the Shape Type to ***3D Cross Unfilled Pulse***.
int SHAPE_TYPE_3D_CROSS_FILLED	Sets the Shape Type to ***3D Cross Filled***.
int SHAPE_TYPE_3D_CROSS_FILLED_IMPLODE	Sets the Shape Type to ***3D Cross Filled Implode***.
int SHAPE_TYPE_3D_CROSS_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Filled Explode***.
int SHAPE_TYPE_3D_CROSS_FILLED_PULSE	Sets the Shape Type to ***3D Cross Filled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED	Sets the Shape Type to ***3D Cross Straight Unfilled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Unfilled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED	Sets the Shape Type to ***3D Cross Straight Filled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Filled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Filled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Filled Pulse***.
int SHAPE_TYPE_PYRAMID_UNFILLED	Sets the Shape Type to ***3D Pyramid Unfilled***.
int SHAPE_TYPE_PYRAMID_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Implode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Explode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_PULSE	Sets the Shape Type to ***3D Pyramid Unfilled Pulse***.
int SHAPE_TYPE_PYRAMID_FILLED	Sets the Shape Type to ***3D Pyramid Filled***.
int SHAPE_TYPE_PYRAMID_FILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Filled Implode***.
int SHAPE_TYPE_PYRAMID_FILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Filled Explode***.

Constant	Description
int **SHAPE_TYPE_PYRAMID_FILLED_PULSE**	Sets the Shape Type to ***3D Pyramid Filled Pulse***.
int **SHAPE_TYPE_TEXT**	Sets the Shape Type to ***Text***.
int **SHAPE_TYPE_WAVE_LINEAR**	Sets the Shape Type to ***Wave Linear***.
int **SHAPE_TYPE_WAVE_RADAR**	Sets the Shape Type to ***Wave Radar***.
int **SHAPE_TYPE_WAVE_HELIX**	Sets the Shape Type to ***Wave Helix***.
int **SHAPE_TYPE_WAVE_CIRCLE**	Sets the Shape Type to ***Wave Circle***.
int **SHAPE_TYPE_WAVE_SQUARE**	Sets the Shape Type to ***Wave Square***.
int **SHAPE_TYPE_WAVE_DIAMOND**	Sets the Shape Type to ***Wave Diamond***.
int **SHAPE_TYPE_WAVE_SPHERE**	Sets the Shape Type to ***Wave Sphere***.
int **SHAPE_TYPE_WAVE_CUBE**	Sets the Shape Type to ***Wave Cube***.
int **SHAPE_TYPE_WAVE_OCTAHEDRON**	Sets the Shape Type to ***Wave Octahedron***.
int **SHAPE_TYPE_RANDOM**	Selects an available Shape Type for each shape randomly. Sets the Shape Type to ***Random***.
int **SHAPE_TYPE_RANDOM_STATIC**	Sets the Shape Type to ***Random Static***.
int **SHAPE_TYPE_RANDOM_IMPLODE**	Sets the Shape Type to ***Random Implode***.
int **SHAPE_TYPE_RANDOM_EXPLODE**	Sets the Shape Type to ***Random Explode***.
int **SHAPE_TYPE_RANDOM_PULSE**	Sets the Shape Type to ***Random Pulse***.
int **SHAPE_TYPE_RANDOM_OUTLINED**	Sets the Shape Type to ***Random Outlined***.
int **SHAPE_TYPE_RANDOM_OUTLINED_STATIC**	Sets the Shape Type to ***Random Outlined Static***.
int **SHAPE_TYPE_RANDOM_OUTLINED_IMPLODE**	Sets the Shape Type to ***Random Outlined Implode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_EXPLODE**	Sets the Shape Type to ***Random Outlined Explode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_PULSE**	Sets the Shape Type to ***Random Outlined Pulse***.
int **SHAPE_TYPE_RANDOM_UNFILLED**	Sets the Shape Type to ***Random Unfilled*** .
int **SHAPE_TYPE_RANDOM_UNFILLED_STATIC**	Sets the Shape Type to ***Random Unfilled Static***.
int **SHAPE_TYPE_RANDOM_UNFILLED_IMPLODE**	Sets the Shape Type to ***Random Unfilled Implode***.
int **SHAPE_TYPE_RANDOM_UNFILLED_EXPLODE**	Sets the Shape Type to ***Random Unfilled Explode***.

Constant	Description
int SHAPE_TYPE_RANDOM_UNFILLED_PULSE	Sets the Shape Type to ***Random Unfilled Pulse***.
int SHAPE_TYPE_RANDOM_FILLED	Sets the Shape Type to ***Random Filled***.
int SHAPE_TYPE_RANDOM_FILLED_STATIC	Sets the Shape Type to ***Random Filled Static***.
int SHAPE_TYPE_RANDOM_FILLED_IMPLODE	Sets the Shape Type to ***Random Filled Implode***.
int SHAPE_TYPE_RANDOM_FILLED_EXPLODE	Sets the Shape Type to ***Random Filled Explode***.
int SHAPE_TYPE_RANDOM_FILLED_PULSE	Sets the Shape Type to ***Random Filled Pulse***.
int SHAPE_TYPE_2D_RANDOM	Sets the Shape Type to ***2D Random***.
int SHAPE_TYPE_2D_RANDOM_STATIC	Sets the Shape Type to ***2D Random Static***.
int SHAPE_TYPE_2D_RANDOM_IMPLODE	Sets the Shape Type to ***2D Random Implode***.
int SHAPE_TYPE_2D_RANDOM_EXPLODE	Sets the Shape Type to ***2D Random Explode***.
int SHAPE_TYPE_2D_RANDOM_PULSE	Sets the Shape Type to ***2D Random Pulse***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED	Sets the Shape Type to ***2D Random Outlined***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_STATIC	Sets the Shape Type to ***2D Random Outlined Static***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_IMPLODE	Sets the Shape Type to ***2D Random Outlined Implode***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_EXPLODE	Sets the Shape Type to ***2D Random Outlined Explode***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_PULSE	Sets the Shape Type to ***2D Random Outlined Pulse***.
int SHAPE_TYPE_2D_RANDOM_FILLED	Sets the Shape Type to ***2D Random Filled***.
int SHAPE_TYPE_2D_RANDOM_FILLED_STATIC	Sets the Shape Type to ***2D Random Filled Static***.
int SHAPE_TYPE_2D_RANDOM_FILLED_IMPLODE	Sets the Shape Type to ***2D Random Filled Implode***.
int SHAPE_TYPE_2D_RANDOM_FILLED_EXPLODE	Sets the Shape Type to ***2D Random Filled Explode***.
int SHAPE_TYPE_2D_RANDOM_FILLED_PULSE	Sets the Shape Type to ***2D Random Filled Pulse***.
int SHAPE_TYPE_3D_RANDOM	Sets the Shape Type to ***3D Random***.
int SHAPE_TYPE_3D_RANDOM_STATIC	Sets the Shape Type to ***3D Random Static***.

Constant	Description
int **SHAPE_TYPE_3D_RANDOM_IMPLODE**	Sets the Shape Type to ***3D Random Implode***.
int **SHAPE_TYPE_3D_RANDOM_EXPLODE**	Sets the Shape Type to ***3D Random Explode***.
int **SHAPE_TYPE_3D_RANDOM_PULSE**	Sets the Shape Type to ***3D Random Pulse***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED**	Sets the Shape Type to ***3D Random Unfilled***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_STATIC**	Sets the Shape Type to ***3D Random Unfilled Static***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Random Unfilled Implode***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Random Unfilled Explode***.
int **SHAPE_TYPE_3D_RANDOM_UNFILLED_PULSE**	Sets the Shape Type to ***3D Random Unfilled Pulse***.
int **SHAPE_TYPE_3D_RANDOM_FILLED**	Sets the Shape Type to ***3D Random Filled***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_STATIC**	Sets the Shape Type to ***3D Random Filled Static***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_IMPLODE**	Sets the Shape Type to ***3D Random Filled Implode***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_EXPLODE**	Sets the Shape Type to ***3D Random Filled Explode***.
int **SHAPE_TYPE_3D_RANDOM_FILLED_PULSE**	Sets the Shape Type to ***3D Random Filled Pulse***.

Shape Alignment Constants

Constant	Description
int **LOOKAT_FRONT**	Sets the Shape Alignment to ***Front***.
int **LOOKAT_BACK**	Sets the Shape Alignment to ***Back***.
int **LOOKAT_LEFT**	Sets the Shape Alignment to ***Left***.
int **LOOKAT_RIGHT**	Sets the Shape Alignment to ***Right***.
int **LOOKAT_TOP**	Sets the Shape Alignment to ***Top***.
int **LOOKAT_BOTTOM**	Sets the Shape Alignment to ***Bottom***.
int **LOOKAT_RANDOM**	Selects an available Shape Alignment for each shape randomly. Sets the Shape Alignment to ***Random***.

Rotation Mode Constants

Constant	Description
int **ROTATION_CCW_0**	Does not activate Rotation. Is the same as **ROTATION_CW_0**.
int **ROTATION_CCW_90**	Rotates by ***90 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_270**.
int **ROTATION_CCW_180**	Rotates by ***180 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_180**.
int **ROTATION_CCW_270**	Rotates by ***270 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_90**.
int **ROTATION_CW_0**	Does not activate Rotation. Is the same as **ROTATION_CCW_0**.
int **ROTATION_CW_90**	Rotates by ***90 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_270**.
int **ROTATION_CW_180**	Rotates by ***180 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_180**.
int **ROTATION_CW_270**	Rotates by ***270 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_90**.
int **ROTATION_RANDOM**	Selects an available Rotation for each shape randomly. Sets the Rotation Mode to ***Random***.

Origin Type Constants

Constant	Description
int **ORIGIN_CENTER**	Sets the Origin Type to ***Center***.
int **ORIGIN_GEOMETRIC_CENTER**	Sets the Origin Type to ***Geometric Center***.
int **ORIGIN_FRONT**	Sets the Origin Type to ***Front***.
int **ORIGIN_BACK**	Sets the Origin Type to ***Back***.
int **ORIGIN_LEFT**	Sets the Origin Type to ***Left***.
int **ORIGIN_RIGHT**	Sets the Origin Type to ***Right***.
int **ORIGIN_TOP**	Sets the Origin Type to ***Top***.
int **ORIGIN_BOTTOM**	Sets the Origin Type to ***Bottom***.
int **ORIGIN_TOP_LEFT**	Sets the Origin Type to ***Top Left***.
int **ORIGIN_TOP_RIGHT**	Sets the Origin Type to ***Top Right***.
int **ORIGIN_BOTTOM_LEFT**	Sets the Origin Type to ***Bottom Left***.
int **ORIGIN_BOTTOM_RIGHT**	Sets the Origin Type to ***Bottom Right***.
int **ORIGIN_FRONT_LEFT**	Sets the Origin Type to ***Front Left***.
int **ORIGIN_FRONT_RIGHT**	Sets the Origin Type to ***Front Right***.
int **ORIGIN_BACK_LEFT**	Sets the Origin Type to ***Back Left***.
int **ORIGIN_BACK_RIGHT**	Sets the Origin Type to ***Back Right***.
int **ORIGIN_FRONT_TOP**	Sets the Origin Type to ***Front Top***.
int **ORIGIN_FRONT_BOTTOM**	Sets the Origin Type to ***Front Bottom***.
int **ORIGIN_BACK_TOP**	Sets the Origin Type to ***Back Top***.
int **ORIGIN_BACK_BOTTOM**	Sets the Origin Type to ***Back Bottom***.
int **ORIGIN_FRONT_TOP_LEFT**	Sets the Origin Type to ***Front Top Left***.
int **ORIGIN_FRONT_TOP_RIGHT**	Sets the Origin Type to ***Front Top Right***.
int **ORIGIN_FRONT_BOTTOM_LEFT**	Sets the Origin Type to ***Front Bottom Left***.
int **ORIGIN_FRONT_BOTTOM_RIGHT**	Sets the Origin Type to ***Front Bottom Right***.
int **ORIGIN_BACK_TOP_LEFT**	Sets the Origin Type to ***Back Top Left***.
int **ORIGIN_BACK_TOP_RIGHT**	Sets the Origin Type to ***Back Top Right***.
int **ORIGIN_BACK_BOTTOM_LEFT**	Sets the Origin Type to ***Back Bottom Left***.
int **ORIGIN_BACK_BOTTOM_RIGHT**	Sets the Origin Type to ***Back Bottom Right***.

Example

This example changes the shape type each second in a predefined order. It works with the »SCE Shapes effect, for example.

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

int g_startTime;
int g_run;

void InitEffect()
{
 time t = GetTime();
 g_startTime = t.hour * 3600 + t.min * 60 + t.sec;

 g_run = 0;
}

void PreRenderEffect()
{
 time t = GetTime();
 int t2 = t.hour * 3600 + t.min * 60 + t.sec;

 if(t2 - g_startTime >= 1)
 {
        g_startTime = t2;
        switch(g_run % 6)
        {
        case 0:       SetShapeType(SHAPE_TYPE_CIRCLE_OUTLINED_EXPLODE);      break;
        case 1:       SetShapeType(SHAPE_TYPE_CIRCLE_OUTLINED);       break;
        case 2:       SetShapeType(SHAPE_TYPE_CIRCLE_OUTLINED_IMPLODE);      break;
        case 3:       SetShapeType(SHAPE_TYPE_SQUARE_OUTLINED_EXPLODE);      break;
        case 4:       SetShapeType(SHAPE_TYPE_SQUARE_OUTLINED);       break;
        case 5:       SetShapeType(SHAPE_TYPE_SQUARE_OUTLINED_IMPLODE);      break;
        default:      break;
        }
        g_run++;
 }
}

void PostRenderEffect()
{

}

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

5.1.13 Using Shape Table

Functions Provided For Using The Shape Table

The following table provides an overview of all functions the effect can use to modify the shapes in a Shape Table:

Function	Description
void **SetRenderingMode**(int mode)	Sets the ***Rendering Mode***. See below for details.
int **GetRenderingMode**()	Returns the current ***Rendering Mode***.
int **ShapeTableGetShapeCount**()	Returns the current number of shapes in the Shape Table.
void **ShapeTableMoveShapeUp**(int index)	Moves a shape with its specified *index* up a place in the Shape Table.
void **ShapeTableMoveShapeDown**(int index)	Moves a shape with its specified *index* down a place in the Shape Table.
void **ShapeTableSwapShapes**(int index1, int index2)	Swaps shapes with their specified *indices* in the Shape Table.
void **ShapeTableAddShape**(int index, int type, int alignment, float innerglow, float border, float outerglow, int origin, float minsize)	Adds another shape to the Shape Table at the specified *index* position. If the index is 0, the new shape is added to the first position. If the index is equal to or greater than the current number of shapes, the new shape is added at the end.
void **ShapeTableRemoveShape**(int index)	Removes the shape at the specified *index*. If the given index is out of range, nothing happens.
void **ShapeTableSetShapeType**(int index, int type)	Sets the shape ***Type*** for the shape with the specified *index* in the Shape Table. See below for details.
int **ShapeTableGetShapeType**(int index)	Returns the shape ***Type*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetShapeAlignment**(int index, int alignment)	Sets the shape ***Alignment*** for the shape with the specified *index* in the Shape Table. See below for details.
int **ShapeTableGetShapeAlignment**(int index)	Returns the shape ***Alignment*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetShapeOrigin**(int index, int origin)	Sets the ***Origin*** for the shape with the specified *index* in the Shape Table. See below for details.
int **ShapeTableGetShapeOrigin**(int index)	Returns the ***Origin*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetShapeInnerGlow**(int index, float value)	Sets the ***Inner Glow*** for the shape with the specified *index* in the Shape Table.
float **ShapeTableGetShapeInnerGlow**(int index)	Returns the ***Inner Glow*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetShapeBorder**(int index, float value)	Sets the ***Border*** for the shape with the specified *index* in the Shape Table.
float **ShapeTableGetShapeBorder**(int index)	Returns the ***Border*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetShapeOuterGlow**(int index, float value)	Sets the ***Outer Glow*** for the shape with the specified *index* in the Shape Table.

float **ShapeTableGetShapeOuterGlow**(int index)	Returns the ***Outer Glow*** for the shape with the specified *index* in the Shape Table.
void **ShapeTableSetMode**(int mode)	Sets the ***Shape Table Mode***. See **below** for details.
int **ShapeTableGetMode**()	Returns the current ***Shape Table Mode***.
int **ShapeTableSetShapeMinSize**(int index, float minsize)	Sets the Minimum Size (***Size MIN***) for the shape with the specified *index* in the Shape Table.
int **ShapeTableGetShapeMinSize**(int index)	Returns the current ***Size MIN*** for the shape with the specified *index* in the Shape Table.

Remarks

Not all effects which use shapes offer Rendering Mode. Therefore, the functions SetRenderingMode and GetRenderingMode are only available if the effect offers Rendering Mode.

Not all effects which use Shape Table offer Shape Table Mode. Therefore, the functions ShapeTableSetMode and ShapeTableGetMode are only available if the effect offers Shape Table Mode.

Also, the effects mostly offer just a selection of all the constants, which are listed below. The function SetRenderingMode will only set Rendering Modes which are offered by the effect. The function SetShapeType will only set Shape Types which are offered by the effect. The function SetShapeAlignment will only set Shape Alignments which are offered by the effect.

Rendering Mode Constants

Constant	Description
int **RENDERING_MODE_SIMPLE**	Sets the Rendering Mode to ***Simple***.
int **RENDERING_MODE_EXTENDED**	Sets the Rendering Mode to ***Extended***.
int **RENDERING_MODE_BLOBBY**	Sets the Rendering Mode to ***Blobby***.

Shape Type Constants

Constant	Description
int **SHAPE_TYPE_LINE**	Sets the Shape Type to ***Line***.
int **SHAPE_TYPE_CURVE**	Sets the Shape Type to ***Curve***.
int **SHAPE_TYPE_FILLED**	Sets the Shape Type to ***Filled***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED**	Sets the Shape Type to ***Rectangle Outlined***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Rectangle Outlined Implode***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Rectangle Outlined Explode***.
int **SHAPE_TYPE_RECTANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Rectangle Outlined Pulse***.
int **SHAPE_TYPE_RECTANGLE_FILLED**	Sets the Shape Type to ***Rectangle Filled***.
int **SHAPE_TYPE_RECTANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Rectangle Filled Implode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Rectangle Filled Explode***.
int **SHAPE_TYPE_RECTANGLE_FILLED_PULSE**	Sets the Shape Type to ***Rectangle Filled Pulse***.
int **SHAPE_TYPE_SQUARE_OUTLINED**	Sets the Shape Type to ***Square Outlined*** .
int **SHAPE_TYPE_SQUARE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Square Outlined Implode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Square Outlined Explode***.
int **SHAPE_TYPE_SQUARE_OUTLINED_PULSE**	Sets the Shape Type to ***Square Outlined Pulse***.
int **SHAPE_TYPE_SQUARE_FILLED**	Sets the Shape Type to ***Square Filled***.
int **SHAPE_TYPE_SQUARE_FILLED_IMPLODE**	Sets the Shape Type to ***Square Filled Implode***.
int **SHAPE_TYPE_SQUARE_FILLED_EXPLODE**	Sets the Shape Type to ***Square Filled Explode***.
int **SHAPE_TYPE_SQUARE_FILLED_PULSE**	Sets the Shape Type to ***Square Filled Pulse***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED**	Sets the Shape Type to ***Ellipse Outlined***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Ellipse Outlined Implode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Ellipse Outlined Explode***.
int **SHAPE_TYPE_ELLIPSE_OUTLINED_PULSE**	Sets the Shape Type to ***Ellipse Outlined Pulse***.
int **SHAPE_TYPE_ELLIPSE_FILLED**	Sets the Shape Type to ***Ellipse Filled***.

Constant	Description
int **SHAPE_TYPE_ELLIPSE_FILLED_IMPLODE**	Sets the Shape Type to ***Ellipse Filled Implode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_EXPLODE**	Sets the Shape Type to ***Ellipse Filled Explode***.
int **SHAPE_TYPE_ELLIPSE_FILLED_PULSE**	Sets the Shape Type to ***Ellipse Filled Pulse***.
int **SHAPE_TYPE_CIRCLE_OUTLINED**	Sets the Shape Type to ***Circle Outlined***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Circle Outlined Implode***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Circle Outlined Explode***.
int **SHAPE_TYPE_CIRCLE_OUTLINED_PULSE**	Sets the Shape Type to ***Circle Outlined Pulse***.
int **SHAPE_TYPE_CIRCLE_FILLED**	Sets the Shape Type to ***Circle Filled***.
int **SHAPE_TYPE_CIRCLE_FILLED_IMPLODE**	Sets the Shape Type to ***Circle Filled Implode***.
int **SHAPE_TYPE_CIRCLE_FILLED_EXPLODE**	Sets the Shape Type to ***Circle Filled Explode***.
int **SHAPE_TYPE_CIRCLE_FILLED_PULSE**	Sets the Shape Type to ***Circle Filled Pulse*** .
int **SHAPE_TYPE_DIAMOND_OUTLINED**	Sets the Shape Type to ***Diamond Outlined***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_IMPLODE**	Sets the Shape Type to ***Diamond Outlined Implode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_EXPLODE**	Sets the Shape Type to ***Diamond Outlined Explode***.
int **SHAPE_TYPE_DIAMOND_OUTLINED_PULSE**	Sets the Shape Type to ***Diamond Outlined Pulse***.
int **SHAPE_TYPE_DIAMOND_FILLED**	Sets the Shape Type to ***Diamond Filled***.
int **SHAPE_TYPE_DIAMOND_FILLED_IMPLODE**	Sets the Shape Type to ***Diamond Filled Implode***.
int **SHAPE_TYPE_DIAMOND_FILLED_EXPLODE**	Sets the Shape Type to ***Diamond Filled Explode***.
int **SHAPE_TYPE_DIAMOND_FILLED_PULSE**	Sets the Shape Type to ***Diamond Filled Pulse***.
int **SHAPE_TYPE_PENTAGON_OUTLINED**	Sets the Shape Type to ***Pentagon Outlined***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_IMPLODE**	Sets the Shape Type to ***Pentagon Outlined Implode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_EXPLODE**	Sets the Shape Type to ***Pentagon Outlined Explode***.
int **SHAPE_TYPE_PENTAGON_OUTLINED_PULSE**	Sets the Shape Type to ***Pentagon Outlined Pulse***.

Constant	Description
int SHAPE_TYPE_PENTAGON_FILLED	Sets the Shape Type to ***Pentagon Filled***.
int SHAPE_TYPE_PENTAGON_FILLED_IMPLODE	Sets the Shape Type to ***Pentagon Filled Implode***.
int SHAPE_TYPE_PENTAGON_FILLED_EXPLODE	Sets the Shape Type to ***Pentagon Filled Explode***.
int SHAPE_TYPE_PENTAGON_FILLED_PULSE	Sets the Shape Type to ***Pentagon Filled Pulse***.
int SHAPE_TYPE_HEXAGON_OUTLINED	Sets the Shape Type to ***Hexagon Outlined***.
int SHAPE_TYPE_HEXAGON_OUTLINED_IMPLODE	Sets the Shape Type to ***Hexagon Outlined Implode***.
int SHAPE_TYPE_HEXAGON_OUTLINED_EXPLODE	Sets the Shape Type to ***Hexagon Outlined Explode***.
int SHAPE_TYPE_HEXAGON_OUTLINED_PULSE	Sets the Shape Type to ***Hexagon Outlined Pulse***.
int SHAPE_TYPE_HEXAGON_FILLED	Sets the Shape Type to ***Hexagon Filled***.
int SHAPE_TYPE_HEXAGON_FILLED_IMPLODE	Sets the Shape Type to ***Hexagon Filled Implode***.
int SHAPE_TYPE_HEXAGON_FILLED_EXPLODE	Sets the Shape Type to ***Hexagon Filled Explode***.
int SHAPE_TYPE_HEXAGON_FILLED_PULSE	Sets the Shape Type to ***Hexagon Filled Pulse***.
int SHAPE_TYPE_HEART_OUTLINED	Sets the Shape Type to ***Heart Outlined***.
int SHAPE_TYPE_HEART_OUTLINED_IMPLODE	Sets the Shape Type to ***Heart Outlined Implode***.
int SHAPE_TYPE_HEART_OUTLINED_EXPLODE	Sets the Shape Type to ***Heart Outlined Explode***.
int SHAPE_TYPE_HEART_OUTLINED_PULSE	Sets the Shape Type to ***Heart Outlined Pulse***.
int SHAPE_TYPE_HEART_FILLED	Sets the Shape Type to ***Heart Filled***.
int SHAPE_TYPE_HEART_FILLED_IMPLODE	Sets the Shape Type to ***Heart Filled Implode***.
int SHAPE_TYPE_HEART_FILLED_EXPLODE	Sets the Shape Type to ***Heart Filled Explode***.
int SHAPE_TYPE_HEART_FILLED_PULSE	Sets the Shape Type to ***Heart Filled Pulse***.
int SHAPE_TYPE_CROSS_OUTLINED	Sets the Shape Type to ***Cross Outlined***.
int SHAPE_TYPE_CROSS_OUTLINED_IMPLODE	Sets the Shape Type to ***Cross Outlined Implode***.
int SHAPE_TYPE_CROSS_OUTLINED_EXPLODE	Sets the Shape Type to ***Cross Outlined Explode***.
int SHAPE_TYPE_CROSS_OUTLINED_PULSE	Sets the Shape Type to ***Cross Outlined Pulse***.

Constant	Description
int **SHAPE_TYPE_CROSS_FILLED**	Sets the Shape Type to ***Cross Filled***.
int **SHAPE_TYPE_CROSS_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Filled Implode***.
int **SHAPE_TYPE_CROSS_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Filled Explode***.
int **SHAPE_TYPE_CROSS_FILLED_PULSE**	Sets the Shape Type to ***Cross Filled Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED**	Sets the Shape Type to ***Cross Straight Outlined***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_IMPLODE**	Sets the Shape Type to ***Cross Straight Outlined Implode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_EXPLODE**	Sets the Shape Type to ***Cross Straight Outlined Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_OUTLINED_PULSE**	Sets the Shape Type to ***Cross Straight Outlined Pulse***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED**	Sets the Shape Type to ***Cross Straight Filled***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_IMPLODE**	Sets the Shape Type to ***Cross Straight Filled Implode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_EXPLODE**	Sets the Shape Type to ***Cross Straight Filled Explode***.
int **SHAPE_TYPE_CROSS_STRAIGHT_FILLED_PULSE**	Sets the Shape Type to ***Cross Straight Filled Pulse***.
int **SHAPE_TYPE_STAR_OUTLINED**	Sets the Shape Type to ***Star Outlined***.
int **SHAPE_TYPE_STAR_OUTLINED_IMPLODE**	Sets the Shape Type to ***Star Outlined Implode***.
int **SHAPE_TYPE_STAR_OUTLINED_EXPLODE**	Sets the Shape Type to ***Star Outlined Explode***.
int **SHAPE_TYPE_STAR_OUTLINED_PULSE**	Sets the Shape Type to ***Star Outlined Pulse***.
int **SHAPE_TYPE_STAR_FILLED**	Sets the Shape Type to ***Star Filled***.
int **SHAPE_TYPE_STAR_FILLED_IMPLODE**	Sets the Shape Type to ***Star Filled Implode***.
int **SHAPE_TYPE_STAR_FILLED_EXPLODE**	Sets the Shape Type to ***Star Filled Explode***.
int **SHAPE_TYPE_STAR_FILLED_PULSE**	Sets the Shape Type to ***Star Filled Pulse***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED**	Sets the Shape Type to ***Triangle Outlined***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_IMPLODE**	Sets the Shape Type to ***Triangle Outlined Implode***.
int **SHAPE_TYPE_TRIANGLE_OUTLINED_EXPLODE**	Sets the Shape Type to ***Triangle Outlined Explode***.

Constant	Description
int **SHAPE_TYPE_TRIANGLE_OUTLINED_PULSE**	Sets the Shape Type to ***Triangle Outlined Pulse***.
int **SHAPE_TYPE_TRIANGLE_FILLED**	Sets the Shape Type to ***Triangle Filled***.
int **SHAPE_TYPE_TRIANGLE_FILLED_IMPLODE**	Sets the Shape Type to ***Triangle Filled Implode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_EXPLODE**	Sets the Shape Type to ***Triangle Filled Explode***.
int **SHAPE_TYPE_TRIANGLE_FILLED_PULSE**	Sets the Shape Type to ***Triangle Filled Pulse***.
int **SHAPE_TYPE_BOX_OUTLINED**	Sets the Shape Type to ***3D Box Outlined*** .
int **SHAPE_TYPE_BOX_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Box Outlined Implode***.
int **SHAPE_TYPE_BOX_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Box Outlined Explode***.
int **SHAPE_TYPE_BOX_OUTLINED_PULSE**	Sets the Shape Type to ***3D Box Outlined Pulse***.
int **SHAPE_TYPE_BOX_UNFILLED**	Sets the Shape Type to ***3D Box Unfilled***.
int **SHAPE_TYPE_BOX_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Box Unfilled Implode***.
int **SHAPE_TYPE_BOX_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Box Unfilled Explode***.
int **SHAPE_TYPE_BOX_UNFILLED_PULSE**	Sets the Shape Type to ***3D Box Unfilled Pulse***.
int **SHAPE_TYPE_BOX_FILLED**	Sets the Shape Type to ***3D Box Filled***.
int **SHAPE_TYPE_BOX_FILLED_IMPLODE**	Sets the Shape Type to ***3D Box Filled Implode***.
int **SHAPE_TYPE_BOX_FILLED_EXPLODE**	Sets the Shape Type to ***3D Box Filled Explode***.
int **SHAPE_TYPE_BOX_FILLED_PULSE**	Sets the Shape Type to ***3D Box Filled Pulse***.
int **SHAPE_TYPE_CUBE_OUTLINED**	Sets the Shape Type to ***3D Cube Outlined***.
int **SHAPE_TYPE_CUBE_OUTLINED_IMPLODE**	Sets the Shape Type to ***3D Cube Outlined Implode***.
int **SHAPE_TYPE_CUBE_OUTLINED_EXPLODE**	Sets the Shape Type to ***3D Cube Outlined Explode***.
int **SHAPE_TYPE_CUBE_OUTLINED_PULSE**	Sets the Shape Type to ***3D Cube Outlined Pulse***.
int **SHAPE_TYPE_CUBE_UNFILLED**	Sets the Shape Type to ***3D Cube Unfilled*** .
int **SHAPE_TYPE_CUBE_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Unfilled Implode***.

Constant	Description
int **SHAPE_TYPE_CUBE_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Unfilled Explode***.
int **SHAPE_TYPE_CUBE_UNFILLED_PULSE**	Sets the Shape Type to ***3D Cube Unfilled Pulse***.
int **SHAPE_TYPE_CUBE_FILLED**	Sets the Shape Type to ***3D Cube Filled***.
int **SHAPE_TYPE_CUBE_FILLED_IMPLODE**	Sets the Shape Type to ***3D Cube Filled Implode***.
int **SHAPE_TYPE_CUBE_FILLED_EXPLODE**	Sets the Shape Type to ***3D Cube Filled Explode***.
int **SHAPE_TYPE_CUBE_FILLED_PULSE**	Sets the Shape Type to ***3D Cube Filled Pulse***.
int **SHAPE_TYPE_SPHERE_UNFILLED**	Sets the Shape Type to ***3D Sphere Unfilled***.
int **SHAPE_TYPE_SPHERE_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Sphere Unfilled Implode***.
int **SHAPE_TYPE_SPHERE_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Sphere Unfilled Explode***.
int **SHAPE_TYPE_SPHERE_UNFILLED_PULSE**	Sets the Shape Type to ***3D Sphere Unfilled Pulse***.
int **SHAPE_TYPE_SPHERE_FILLED**	Sets the Shape Type to ***3D Sphere Filled***.
int **SHAPE_TYPE_SPHERE_FILLED_IMPLODE**	Sets the Shape Type to ***3D Sphere Filled Implode***.
int **SHAPE_TYPE_SPHERE_FILLED_EXPLODE**	Sets the Shape Type to ***3D Sphere Filled Explode***.
int **SHAPE_TYPE_SPHERE_FILLED_PULSE**	Sets the Shape Type to ***3D Sphere Filled Pulse***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED**	Sets the Shape Type to ***3D Ellipsoid Unfilled***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_IMPLODE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Implode***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_EXPLODE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Explode***.
int **SHAPE_TYPE_ELLIPSOID_UNFILLED_PULSE**	Sets the Shape Type to ***3D Ellipsoid Unfilled Pulse***.
int **SHAPE_TYPE_ELLIPSOID_FILLED**	Sets the Shape Type to ***3D Ellipsoid Filled***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_IMPLODE**	Sets the Shape Type to ***3D Ellipsoid Filled Implode***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_EXPLODE**	Sets the Shape Type to ***3D Ellipsoid Filled Explode***.
int **SHAPE_TYPE_ELLIPSOID_FILLED_PULSE**	Sets the Shape Type to ***3D Ellipsoid Filled Pulse***.
int **SHAPE_TYPE_OCTAHEDRON_UNFILLED**	Sets the Shape Type to ***3D Octahedron Unfilled***.

Constant	Description
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Implode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Unfilled Explode***.
int SHAPE_TYPE_OCTAHEDRON_UNFILLED_PULSE	Sets the Shape Type to ***3D Octahedron Unfilled Pulse***.
int SHAPE_TYPE_OCTAHEDRON_FILLED	Sets the Shape Type to ***3D Octahedron Filled***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_IMPLODE	Sets the Shape Type to ***3D Octahedron Filled Implode***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_EXPLODE	Sets the Shape Type to ***3D Octahedron Filled Explode***.
int SHAPE_TYPE_OCTAHEDRON_FILLED_PULSE	Sets the Shape Type to ***3D Octahedron Filled Pulse***.
int SHAPE_TYPE_3D_HEART_UNFILLED	Sets the Shape Type to ***3D Heart Unfilled***.
int SHAPE_TYPE_3D_HEART_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Heart Unfilled Implode***.
int SHAPE_TYPE_3D_HEART_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Heart Unfilled Explode***.
int SHAPE_TYPE_3D_HEART_UNFILLED_PULSE	Sets the Shape Type to ***3D Heart Unfilled Pulse***.
int SHAPE_TYPE_3D_HEART_FILLED	Sets the Shape Type to ***3D Heart Filled***.
int SHAPE_TYPE_3D_HEART_FILLED_IMPLODE	Sets the Shape Type to ***3D Heart Filled Implode***.
int SHAPE_TYPE_3D_HEART_FILLED_EXPLODE	Sets the Shape Type to ***3D Heart Filled Explode***.
int SHAPE_TYPE_3D_HEART_FILLED_PULSE	Sets the Shape Type to ***3D Heart Filled Pulse***.
int SHAPE_TYPE_3D_STAR_UNFILLED	Sets the Shape Type to ***3D Star Unfilled***.
int SHAPE_TYPE_3D_STAR_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Star Unfilled Implode***.
int SHAPE_TYPE_3D_STAR_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Star Unfilled Explode***.
int SHAPE_TYPE_3D_STAR_UNFILLED_PULSE	Sets the Shape Type to ***3D Star Unfilled Pulse***.
int SHAPE_TYPE_3D_STAR_FILLED	Sets the Shape Type to ***3D Star Filled***.
int SHAPE_TYPE_3D_STAR_FILLED_IMPLODE	Sets the Shape Type to ***3D Star Filled Implode***.
int SHAPE_TYPE_3D_STAR_FILLED_EXPLODE	Sets the Shape Type to ***3D Star Filled Explode***.
int SHAPE_TYPE_3D_STAR_FILLED_PULSE	Sets the Shape Type to ***3D Star Filled Pulse***.

Constant	Description
int SHAPE_TYPE_3D_CROSS_UNFILLED	Sets the Shape Type to ***3D Cross Unfilled***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Cross Unfilled Implode***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Cross Unfilled Explode***.
int SHAPE_TYPE_3D_CROSS_UNFILLED_PULSE	Sets the Shape Type to ***3D Cross Unfilled Pulse***.
int SHAPE_TYPE_3D_CROSS_FILLED	Sets the Shape Type to ***3D Cross Filled***.
int SHAPE_TYPE_3D_CROSS_FILLED_IMPLODE	Sets the Shape Type to ***3D Cross Filled Implode***.
int SHAPE_TYPE_3D_CROSS_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Filled Explode***.
int SHAPE_TYPE_3D_CROSS_FILLED_PULSE	Sets the Shape Type to ***3D Cross Filled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED	Sets the Shape Type to ***3D Cross Straight Unfilled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Unfilled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_UNFILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Unfilled Pulse***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED	Sets the Shape Type to ***3D Cross Straight Filled***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_IMPLODE	Sets the Shape Type to ***3D Cross Straight Filled Implode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_EXPLODE	Sets the Shape Type to ***3D Cross Straight Filled Explode***.
int SHAPE_TYPE_3D_CROSS_STRAIGHT_FILLED_PULSE	Sets the Shape Type to ***3D Cross Straight Filled Pulse***.
int SHAPE_TYPE_PYRAMID_UNFILLED	Sets the Shape Type to ***3D Pyramid Unfilled***.
int SHAPE_TYPE_PYRAMID_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Implode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Unfilled Explode***.
int SHAPE_TYPE_PYRAMID_UNFILLED_PULSE	Sets the Shape Type to ***3D Pyramid Unfilled Pulse***.
int SHAPE_TYPE_PYRAMID_FILLED	Sets the Shape Type to ***3D Pyramid Filled***.
int SHAPE_TYPE_PYRAMID_FILLED_IMPLODE	Sets the Shape Type to ***3D Pyramid Filled Implode***.
int SHAPE_TYPE_PYRAMID_FILLED_EXPLODE	Sets the Shape Type to ***3D Pyramid Filled Explode***.

Constant	Description
int **SHAPE_TYPE_PYRAMID_FILLED_PULSE**	Sets the Shape Type to ***3D Pyramid Filled Pulse***.
int **SHAPE_TYPE_TEXT**	Sets the Shape Type to ***Text***.
int **SHAPE_TYPE_WAVE_LINEAR**	Sets the Shape Type to ***Wave Linear***.
int **SHAPE_TYPE_WAVE_RADAR**	Sets the Shape Type to ***Wave Radar***.
int **SHAPE_TYPE_WAVE_HELIX**	Sets the Shape Type to ***Wave Helix***.
int **SHAPE_TYPE_WAVE_CIRCLE**	Sets the Shape Type to ***Wave Circle***.
int **SHAPE_TYPE_WAVE_SQUARE**	Sets the Shape Type to ***Wave Square***.
int **SHAPE_TYPE_WAVE_DIAMOND**	Sets the Shape Type to ***Wave Diamond***.
int **SHAPE_TYPE_WAVE_SPHERE**	Sets the Shape Type to ***Wave Sphere***.
int **SHAPE_TYPE_WAVE_CUBE**	Sets the Shape Type to ***Wave Cube***.
int **SHAPE_TYPE_WAVE_OCTAHEDRON**	Sets the Shape Type to ***Wave Octahedron***.
int **SHAPE_TYPE_RANDOM**	Selects an available Shape Type for each shape randomly. Sets the Shape Type to ***Random***.
int **SHAPE_TYPE_RANDOM_STATIC**	Sets the Shape Type to ***Random Static***.
int **SHAPE_TYPE_RANDOM_IMPLODE**	Sets the Shape Type to ***Random Implode***.
int **SHAPE_TYPE_RANDOM_EXPLODE**	Sets the Shape Type to ***Random Explode***.
int **SHAPE_TYPE_RANDOM_PULSE**	Sets the Shape Type to ***Random Pulse***.
int **SHAPE_TYPE_RANDOM_OUTLINED**	Sets the Shape Type to ***Random Outlined***.
int **SHAPE_TYPE_RANDOM_OUTLINED_STATIC**	Sets the Shape Type to ***Random Outlined Static***.
int **SHAPE_TYPE_RANDOM_OUTLINED_IMPLODE**	Sets the Shape Type to ***Random Outlined Implode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_EXPLODE**	Sets the Shape Type to ***Random Outlined Explode***.
int **SHAPE_TYPE_RANDOM_OUTLINED_PULSE**	Sets the Shape Type to ***Random Outlined Pulse***.
int **SHAPE_TYPE_RANDOM_UNFILLED**	Sets the Shape Type to ***Random Unfilled*** .
int **SHAPE_TYPE_RANDOM_UNFILLED_STATIC**	Sets the Shape Type to ***Random Unfilled Static***.
int **SHAPE_TYPE_RANDOM_UNFILLED_IMPLODE**	Sets the Shape Type to ***Random Unfilled Implode***.
int **SHAPE_TYPE_RANDOM_UNFILLED_EXPLODE**	Sets the Shape Type to ***Random Unfilled Explode***.

Constant	Description
int SHAPE_TYPE_RANDOM_UNFILLED_PULSE	Sets the Shape Type to ***Random Unfilled Pulse***.
int SHAPE_TYPE_RANDOM_FILLED	Sets the Shape Type to ***Random Filled***.
int SHAPE_TYPE_RANDOM_FILLED_STATIC	Sets the Shape Type to ***Random Filled Static***.
int SHAPE_TYPE_RANDOM_FILLED_IMPLODE	Sets the Shape Type to ***Random Filled Implode***.
int SHAPE_TYPE_RANDOM_FILLED_EXPLODE	Sets the Shape Type to ***Random Filled Explode***.
int SHAPE_TYPE_RANDOM_FILLED_PULSE	Sets the Shape Type to ***Random Filled Pulse***.
int SHAPE_TYPE_2D_RANDOM	Sets the Shape Type to ***2D Random***.
int SHAPE_TYPE_2D_RANDOM_STATIC	Sets the Shape Type to ***2D Random Static***.
int SHAPE_TYPE_2D_RANDOM_IMPLODE	Sets the Shape Type to ***2D Random Implode***.
int SHAPE_TYPE_2D_RANDOM_EXPLODE	Sets the Shape Type to ***2D Random Explode***.
int SHAPE_TYPE_2D_RANDOM_PULSE	Sets the Shape Type to ***2D Random Pulse***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED	Sets the Shape Type to ***2D Random Outlined***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_STATIC	Sets the Shape Type to ***2D Random Outlined Static***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_IMPLODE	Sets the Shape Type to ***2D Random Outlined Implode***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_EXPLODE	Sets the Shape Type to ***2D Random Outlined Explode***.
int SHAPE_TYPE_2D_RANDOM_OUTLINED_PULSE	Sets the Shape Type to ***2D Random Outlined Pulse***.
int SHAPE_TYPE_2D_RANDOM_FILLED	Sets the Shape Type to ***2D Random Filled***.
int SHAPE_TYPE_2D_RANDOM_FILLED_STATIC	Sets the Shape Type to ***2D Random Filled Static***.
int SHAPE_TYPE_2D_RANDOM_FILLED_IMPLODE	Sets the Shape Type to ***2D Random Filled Implode***.
int SHAPE_TYPE_2D_RANDOM_FILLED_EXPLODE	Sets the Shape Type to ***2D Random Filled Explode***.
int SHAPE_TYPE_2D_RANDOM_FILLED_PULSE	Sets the Shape Type to ***2D Random Filled Pulse***.
int SHAPE_TYPE_3D_RANDOM	Sets the Shape Type to ***3D Random***.
int SHAPE_TYPE_3D_RANDOM_STATIC	Sets the Shape Type to ***3D Random Static***.

Constant	Description
int SHAPE_TYPE_3D_RANDOM_IMPLODE	Sets the Shape Type to ***3D Random Implode***.
int SHAPE_TYPE_3D_RANDOM_EXPLODE	Sets the Shape Type to ***3D Random Explode***.
int SHAPE_TYPE_3D_RANDOM_PULSE	Sets the Shape Type to ***3D Random Pulse***.
int SHAPE_TYPE_3D_RANDOM_UNFILLED	Sets the Shape Type to ***3D Random Unfilled***.
int SHAPE_TYPE_3D_RANDOM_UNFILLED_STATIC	Sets the Shape Type to ***3D Random Unfilled Static***.
int SHAPE_TYPE_3D_RANDOM_UNFILLED_IMPLODE	Sets the Shape Type to ***3D Random Unfilled Implode***.
int SHAPE_TYPE_3D_RANDOM_UNFILLED_EXPLODE	Sets the Shape Type to ***3D Random Unfilled Explode***.
int SHAPE_TYPE_3D_RANDOM_UNFILLED_PULSE	Sets the Shape Type to ***3D Random Unfilled Pulse***.
int SHAPE_TYPE_3D_RANDOM_FILLED	Sets the Shape Type to ***3D Random Filled***.
int SHAPE_TYPE_3D_RANDOM_FILLED_STATIC	Sets the Shape Type to ***3D Random Filled Static***.
int SHAPE_TYPE_3D_RANDOM_FILLED_IMPLODE	Sets the Shape Type to ***3D Random Filled Implode***.
int SHAPE_TYPE_3D_RANDOM_FILLED_EXPLODE	Sets the Shape Type to ***3D Random Filled Explode***.
int SHAPE_TYPE_3D_RANDOM_FILLED_PULSE	Sets the Shape Type to ***3D Random Filled Pulse***.

Shape Alignment Constants

Constant	Description
int LOOKAT_FRONT	Sets the Shape Alignment to ***Front***.
int LOOKAT_BACK	Sets the Shape Alignment to ***Back***.
int LOOKAT_LEFT	Sets the Shape Alignment to ***Left***.
int LOOKAT_RIGHT	Sets the Shape Alignment to ***Right***.
int LOOKAT_TOP	Sets the Shape Alignment to ***Top***.
int LOOKAT_BOTTOM	Sets the Shape Alignment to ***Bottom***.
int LOOKAT_RANDOM	Selects an available Shape Alignment for each shape randomly. Sets the Shape Alignment to ***Random***.

Origin Type Constants

Constant	Description
int **ORIGIN_CENTER**	Sets the Origin Type to ***Center***.
int **ORIGIN_GEOMETRIC_CENTER**	Sets the Origin Type to ***Geometric Center***.
int **ORIGIN_FRONT**	Sets the Origin Type to ***Front***.
int **ORIGIN_BACK**	Sets the Origin Type to ***Back***.
int **ORIGIN_LEFT**	Sets the Origin Type to ***Left***.
int **ORIGIN_RIGHT**	Sets the Origin Type to ***Right***.
int **ORIGIN_TOP**	Sets the Origin Type to ***Top***.
int **ORIGIN_BOTTOM**	Sets the Origin Type to ***Bottom***.
int **ORIGIN_TOP_LEFT**	Sets the Origin Type to ***Top Left***.
int **ORIGIN_TOP_RIGHT**	Sets the Origin Type to ***Top Right***.
int **ORIGIN_BOTTOM_LEFT**	Sets the Origin Type to ***Bottom Left***.
int **ORIGIN_BOTTOM_RIGHT**	Sets the Origin Type to ***Bottom Right***.
int **ORIGIN_FRONT_LEFT**	Sets the Origin Type to ***Front Left***.
int **ORIGIN_FRONT_RIGHT**	Sets the Origin Type to ***Front Right***.
int **ORIGIN_BACK_LEFT**	Sets the Origin Type to ***Back Left***.
int **ORIGIN_BACK_RIGHT**	Sets the Origin Type to ***Back Right***.
int **ORIGIN_FRONT_TOP**	Sets the Origin Type to ***Front Top***.
int **ORIGIN_FRONT_BOTTOM**	Sets the Origin Type to ***Front Bottom***.
int **ORIGIN_BACK_TOP**	Sets the Origin Type to ***Back Top***.
int **ORIGIN_BACK_BOTTOM**	Sets the Origin Type to ***Back Bottom***.
int **ORIGIN_FRONT_TOP_LEFT**	Sets the Origin Type to ***Front Top Left***.
int **ORIGIN_FRONT_TOP_RIGHT**	Sets the Origin Type to ***Front Top Right***.
int **ORIGIN_FRONT_BOTTOM_LEFT**	Sets the Origin Type to ***Front Bottom Left***.
int **ORIGIN_FRONT_BOTTOM_RIGHT**	Sets the Origin Type to ***Front Bottom Right***.
int **ORIGIN_BACK_TOP_LEFT**	Sets the Origin Type to ***Back Top Left***.
int **ORIGIN_BACK_TOP_RIGHT**	Sets the Origin Type to ***Back Top Right***.
int **ORIGIN_BACK_BOTTOM_LEFT**	Sets the Origin Type to ***Back Bottom Left***.
int **ORIGIN_BACK_BOTTOM_RIGHT**	Sets the Origin Type to ***Back Bottom Right***.

Shape Table Mode Constants

Constant	Description
int SHAPETABLE_MODE_LOOP	The shapes of the list will be used one after another in a loop.
int SHAPETABLE_MODE_SHUFFLE	The shapes of the list will be used without a specific order.

Example

This example adds three shapes to the shape table each time it is executed. The shapes have a random shape type (circle, square or triangle) and a different border. The example works with the »SCE Rotating Shapes effect, for example.

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

void InitEffect()
{
    int count = ShapeTableGetShapeCount();
    int type = SHAPE_TYPE_CIRCLE_OUTLINED;

    switch (random(0, 2))
    {
    case 0:
        WriteText("Adding circles ...");
        type = SHAPE_TYPE_CIRCLE_OUTLINED;
        break;
    case 1:
        WriteText("Adding squares ...");
        type = SHAPE_TYPE_SQUARE_OUTLINED;
        break;
    case 2:
        WriteText("Adding triangles ...");
        type = SHAPE_TYPE_TRIANGLE_OUTLINED;
        break;
    }

    ShapeTableAddShape(count++, type, LOOKAT_FRONT, 20.0, 20.0, 20.0);
    ShapeTableAddShape(count++, type, LOOKAT_FRONT, 20.0, 40.0, 20.0);
    ShapeTableAddShape(count++, type, LOOKAT_FRONT, 20.0, 60.0, 20.0);
}

void PreRenderEffect()
{

}

void PostRenderEffect()
```

```
{

}

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

5.1.14 Using Shape Rotation

Functions Provided For Rotating Shapes

The following table provides an overview of all functions to rotate shapes:

Function	Description
void **SetRotationX**(float value)	Sets the ***Rotation X*** *value*. Valid values for *value* range from *-180.00* to *180.00*.
float **GetRotationX**()	Returns the currently set value for ***Rotation X***.
void **SetRotationY**(float value)	Sets the ***Rotation Y*** *value*. Valid values for *value* range from *-180.00* to *180.00*.
float **GetRotationY**()	Returns the currently set value for ***Rotation Y***.
void **SetRotationZ**(float value)	Sets the ***Rotation Z*** *value*. Valid values for *value* range from *-180.00* to *180.00*.
float **GetRotationZ**()	Returns the currently set value for ***Rotation Z***.
void **SetRotation**(float x, float y, float z)	Sets the ***Rotation X, Y, Z*** values. Valid values for *x, y, z* range from *-180.00* to *180.00*.
void **SetRotationOrigin**(int mode)	Sets the ***Rotation Origin***. Please use a constant **as described below** for *mode*.
int **GetRotationOrigin**()	Returns the currently set ***Rotation Origin***.
void **SetRotationAnimationX**(int value)	Sets the ***Permanent Rotation Mode*** for ***Rotation X***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetRotationAnimationX**()	Returns if the ***Permanent Rotation Mode*** for ***Rotation X*** is activated or deactivated.
void **ToggleRotationAnimationX**()	Toggles the ***Permanent Rotation Mode*** for ***Rotation X***.
void **SetRotationAnimationY**(int value)	Sets the ***Permanent Rotation Mode*** for ***Rotation Y***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetRotationAnimationY**()	Toggles the ***Permanent Rotation Mode*** for ***Rotation Y***.
void **ToggleRotationAnimationY**()	Returns if the ***Permanent Rotation Mode*** for ***Rotation Y*** is activated or deactivated.
void **SetRotationAnimationZ**(int value)	Sets the ***Permanent Rotation Mode*** for ***Rotation Z***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetRotationAnimationZ**()	Toggles the ***Permanent Rotation Mode*** for ***Rotation Z***.

void **ToggleRotationAnimationZ**()	Returns if the ***Permanent Rotation Mode*** for ***Rotation Z*** is activated or deactivated.
void **SetRotationAnimation**(int x, int y, int z)	Sets the ***Permanent Rotation Mode*** for ***Rotation X***, ***Y***, ***Z***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.

Remarks

Not all effects which use shapes offer Rotation Animation. Therefore, the functions are only available if the effect offers Rotation Animation.

Rotation Origin Constants

Constant	Description
int ORIGIN_CENTER	Sets the Rotation Origin to ***Center***.
int ORIGIN_GEOMETRIC_CENTER	Sets the Rotation Origin to ***Geometric Center***.
int ORIGIN_FRONT	Sets the Rotation Origin to ***Front***.
int ORIGIN_BACK	Sets the Rotation Origin to ***Back***.
int ORIGIN_LEFT	Sets the Rotation Origin to ***Left***.
int ORIGIN_RIGHT	Sets the Rotation Origin to ***Right***.
int ORIGIN_TOP	Sets the Rotation Origin to ***Top***.
int ORIGIN_BOTTOM	Sets the Rotation Origin to ***Bottom***.
int ORIGIN_TOP_LEFT	Sets the Rotation Origin to ***Top Left***.
int ORIGIN_TOP_RIGHT	Sets the Rotation Origin to ***Top Right***.
int ORIGIN_BOTTOM_LEFT	Sets the Rotation Origin to ***Bottom Left***.
int ORIGIN_BOTTOM_RIGHT	Sets the Rotation Origin to ***Bottom Right***.
int ORIGIN_FRONT_LEFT	Sets the Rotation Origin to ***Front Left***.
int ORIGIN_FRONT_RIGHT	Sets the Rotation Origin to ***Front Right***.
int ORIGIN_BACK_LEFT	Sets the Rotation Origin to ***Back Left***.
int ORIGIN_BACK_RIGHT	Sets the Rotation Origin to ***Back Right***.
int ORIGIN_FRONT_TOP	Sets the Rotation Origin to ***Front Top***.
int ORIGIN_FRONT_BOTTOM	Sets the Rotation Origin to ***Front Bottom***.
int ORIGIN_BACK_TOP	Sets the Rotation Origin to ***Back Top***.
int ORIGIN_BACK_BOTTOM	Sets the Rotation Origin to ***Back Bottom***.
int ORIGIN_FRONT_TOP_LEFT	Sets the Rotation Origin to ***Front Top Left***.
int ORIGIN_FRONT_TOP_RIGHT	Sets the Rotation Origin to ***Front Top Right***.
int ORIGIN_FRONT_BOTTOM_LEFT	Sets the Rotation Origin to ***Front Bottom Left***.
int ORIGIN_FRONT_BOTTOM_RIGHT	Sets the Rotation Origin to ***Front Bottom Right***.
int ORIGIN_BACK_TOP_LEFT	Sets the Rotation Origin to ***Back Top Left***.
int ORIGIN_BACK_TOP_RIGHT	Sets the Rotation Origin to ***Back Top Right***.
int ORIGIN_BACK_BOTTOM_LEFT	Sets the Rotation Origin to ***Back Bottom Left***.
int ORIGIN_BACK_BOTTOM_RIGHT	Sets the Rotation Origin to ***Back Bottom Right***.

5.1.15 Using String Table

Functions Provided For Using The String Table

The following table provides an overview of all functions the effect can use to modify the elements in a String Table:

Function	Description
int **StringTableGetCount**(int page)	Returns the current number of elements in the String Table.
void **StringTableMoveStringUp**(int index, int page)	Moves an element with its specified *index* up a place in the String Table. Indexing starts with 0.
void **StringTableMoveStringDown**(int index, int page)	Moves an element with its specified *index* down a place in the String Table. Indexing starts with 0.
void **StringTableInvert**(int page)	Inverts the complete String Table regarding the positions of the elements in the table.
void **StringTableSwapStrings**(int index1, int index2, int page)	Swaps elements with their specified *indices* in the String Table. Indexing starts with 0.
void **StringTableSetString**(int index, string, int page)	Sets a new *string* text for the element with the specified *index*. Indexing starts with 0.
string **StringTableGetString**(int index, int page)	Returns the text as a *string* for the element with the specified *index*. Indexing starts with 0.
void **StringTableAddString**(int index, string, int page)	Adds a new element with its text as *string* to the specified *index*. Indexing starts with 0.
void **StringTableRemoveString**(int index, int page)	Removes the element with the specified *index* from the String Table. Indexing starts with 0.
void **StringTableSetLoop**(int enable)	Sets the ***Loop Mode***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **StringTableGetLoop**()	Returns 1 (**TRUE**) if ***Loop Mode*** is activated, otherwise 0 (**FALSE**).
void **StringTableToggleLoop**()	Toggles the ***Loop Mode***.
int **StringTableGetShuffle**()	Sets the ***Shuffle Mode***. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
void **StringTableSetShuffle**(int enable)	Returns 1 (**TRUE**) if ***Shuffle Mode*** is activated, otherwise 0 (**FALSE**).
void **StringTableToggleShuffle**()	Toggles the ***Shuffle Mode***.
int **StringTableGetPagesCount**()	Returns the current number of pages in the String Table.
void **StringTableAddPage**(int count)	Adds the number of new pages as defined by *count*.
void **StringTableRemovePage**(int count)	Removes the number of pages as defined by *count*.
void **StringTableMovePageToNext**(int index)	Moves the pages defined by *index* to the next position to the right. Indexing starts with 0.
void **StringTableMovePageToPrev**(int index)	Moves the pages defined by *index* to the previous position. Indexing starts with 0.
void **StringTableInvertPages**()	Inverts the order of pages.

void **StringTableSwapPages**(int index1, int index2)	Swaps the position of two pages as defined by *index1* and *index2*. Indexing starts with 0.

5.1.16 Using Text

Functions Provided For Setting Text

The following table provides an overview of all functions to modify text:

Function	Description
void **EditableTextSetText**(string value)	Sets the text to display.
string **EditableTextGetText**()	Returns the currently displayed text.
void **EditableTextSetTextRotationMode**(int value)	Sets the text ***Rotation*** mode. See below for a list of constants.
int **EditableTextGetTextRotationMode**()	Returns the current text ***Rotation*** mode. See below for a list of constants.
void **EditableTextSetTextSplittingMode**(int value)	Sets the text ***Splitting*** mode. See below for a list of constants.
int **EditableTextGetTextSplittingMode**()	Returns the current text ***Splitting*** mode. See below for a list of constants.
void **EditableTextSetRepeatText**(int value)	Use *value* 1 (**TRUE**) to activate ***Repeat Text***, which means that the text is repeated to fill empty space. Use *value* 0 (**FALSE**) to deactivate it.
int **EditableTextGetRepeatText**()	Returns 1 (**TRUE**) if ***Repeat Text*** is activated, otherwise 0 (**FALSE**).
void **EditableTextToggleRepeatText**()	Activates or deactivates ***Repeat Text***, depending on the current state.
void **EditableTextSetReverseWords**(int value)	Use *value* 1 (**TRUE**) to invert the words' order, for example "Music makes the light" becomes "light the makes Music". Use *value* 0 (**FALSE**) to use the default setting.
int **EditableTextGetReverseWords**()	Returns 1 (**TRUE**) if the words' order is inverted, otherwise 0 (**FALSE**).
void **EditableTextToggleReverseWords**()	Inverts the words' order or uses the default setting, depending on the current state.
void **EditableTextSetReverseCharacters**(int value)	Use *value* 1 (**TRUE**) to invert the characters' order, for example "Music makes the light" becomes "thgil eht sekam cisuM". Use *value* 0 (**FALSE**) to use the default setting.
int **EditableTextGetReverseCharacters**()	Returns 1 (**TRUE**) if the characters' order is inverted, otherwise 0 (**FALSE**).
void **EditableTextToggleReverseCharacters**()	Inverts the characters' order or uses the default setting, depending on the current state.

void **EditableTextSetFontFaceName**(string value)	Sets the font face name. Valid values for *value* should not exceed a length of 31 characters.
string **EditableTextGetFontFaceName**()	Returns the current font face name.
void **EditableTextSetFontHeight**(int value)	Sets the font height. Valid values for *value* can be positive or negative. However, positive and negative values are interpreted differently. Value 0 uses a default font height.
int **EditableTextGetFontHeight**()	Returns the current font height.
void **EditableTextSetFontWidth**(int value)	Sets the font width. Valid values for *value* should be positive.
int **EditableTextGetFontWidth**()	Returns the current font width.
void **EditableTextSetFontWeight**(int value)	Sets the font weight. Valid values for *value* range from *0* to *1000*.
int **EditableTextGetFontWeight**()	Returns the current font weight.
void **EditableTextSetFontItalic**(int value)	Use *value* 1 (**TRUE**) to display italic text. Use *value* 0 (**FALSE**) to use the default setting.
int **EditableTextGetFontItalic**()	Returns 1 (**TRUE**) if the displayed text is italic, otherwise 0 (**FALSE**).
void **EditableTextToggleFontItalic**()	Displays italic text or uses the default setting, depending on the current state.
void **EditableTextSetFontUnderline**(int value)	Use *value* 1 (**TRUE**) to display underlined text. Use *value* 0 (**FALSE**) to use the default setting.
int **EditableTextGetFontUnderline**()	Returns 1 (**TRUE**) if the displayed text is underlined, otherwise 0 (**FALSE**).
void **EditableTextToggleFontUnderline**()	Displays underlined text or uses the default setting, depending on the current state.
void **EditableTextSetFontStrikeOut**(int value)	Use *value* 1 (**TRUE**) to display struck out text. Use *value* 0 (**FALSE**) to use the default setting.
int **EditableTextGetFontStrikeOut**()	Returns 1 (**TRUE**) if the displayed text is struck out, otherwise 0 (**FALSE**).
void **EditableTextToggleFontStrikeOut**()	Displays struck out text or uses the default setting, depending on the current state.

Remarks

Not all effects which use text offer the text rotation mode, the text splitting mode, or the possibilities to render repeated text and to reverse the words/characters. Therefore, the appropriate functions are only available if the effect offers the features.

Rotation Mode Constants

Constant	Description
int **ROTATION_CCW_0**	Does not activate Rotation. Is the same as **ROTATION_CW_0**.
int **ROTATION_CCW_90**	Rotates by ***90 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_270**.
int **ROTATION_CCW_180**	Rotates by ***180 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_180**.
int **ROTATION_CCW_270**	Rotates by ***270 degrees*** in counter-clockwise direction. Is the same as **ROTATION_CW_90**.
int **ROTATION_CW_0**	Does not activate Rotation. Is the same as **ROTATION_CCW_0**.
int **ROTATION_CW_90**	Rotates by ***90 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_270**.
int **ROTATION_CW_180**	Rotates by ***180 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_180**.
int **ROTATION_CW_270**	Rotates by ***270 degrees*** in clockwise direction. Is the same as **ROTATION_CCW_90**.

Text Splitting Mode Constants

Constant	Description
int **TEXT_SPLITTING_NONE**	Treats the whole text as a single object.
int **TEXT_SPLITTING_WORDS**	Treats every word of the text as a separate object.
int **TEXT_SPLITTING_CHARACTERS**	Treats every character of the text as a separate object.

Example

This macro example parses the displayed text for comma-separated format tags: "italic", "bold" and font face names are recognized. Numbers are interpreted as font height. It works with the »**SCE Graph** effect, for example. Just set the shape type to text and edit it. For example, set the text to "italic, Comic Sans MS, 16" and the text is displayed automatically with this format. Of course, the same text is displayed on the matrix. That is why this is only a test scenario.

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

void InitEffect()
{
```

```
 SetShapeType(SHAPE_TYPE_TEXT);
}

void PreRenderEffect()
{
 // default values
 int fontHeight = 12;
 int fontWeight = 400;
 int isItalic = false;
 string faceName = "MS Sans Serif";
 string text = EditableTextGetText(), token;
 string tokens[];

 // parse values
 tokenize(text, ",", tokens);

 for (int i = 0; i < tokens.length; i++)
 {
        token = tokens[i];
        tolower(token);             // compare keywords case-insensitive
        strip(token);        // remove whitespace around keywords

        if (token.length == 0)
               continue;
        else if (isnum(token) == true)
               fontHeight = (int)token;
        else if (token == "bold")
               fontWeight = 800;
        else if (token == "italic")
               isItalic = true;
        else
               faceName = token;
 }

 // set values
 EditableTextSetFontHeight(fontHeight);
 EditableTextSetFontWeight(fontWeight);
 EditableTextSetFontItalic(isItalic);
 EditableTextSetFontFaceName(faceName);
}

void PostRenderEffect()
{

}

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

5.1.17 Using Directions

Functions Provided For Setting Effect Directions

The following table provides an overview of all functions to modify effect directions:

Function	Description
void **SetDirection**(int value)	Sets the ***Direction***. Please use a constant as described below for *value* .
int **GetDirection**()	Returns the current ***Direction***.
void **SetDirectionCrossMode**(int value)	Use 1 (**TRUE**) to activate the direction ***Cross Mode***. Use 0 (**FALSE**) to deactivate it.
int **GetDirectionCrossMode**()	Returns 1 (**TRUE**) if the direction ***Cross Mode*** is activated, otherwise 0 (**FALSE**).
void **ToggleDirectionCrossMode**()	Toggles the direction ***Cross Mode***.

Remarks

Not all effects which use directions offer Cross Mode. Therefore, the functions SetDirectionCrossMode, GetDirectionCrossMode, and ToggleDirectionCrossMode are only available if the effect offers Cross Mode. Also the effects mostly offer just a selection of all the directions which are listed below. The function SetDirection will only set directions which are offered by the effect.

Direction Constants

Constant	Description
int **DIRECTION_NONE**	Sets no direction. Usually this is the same as stopping the movement of an effect if this is possible with the effect.
int **DIRECTION_FRONT**	Sets the direction to ***Front***.
int **DIRECTION_BACK**	Sets the direction to ***Back***.
int **DIRECTION_LEFT**	Sets the direction to ***Left***.
int **DIRECTION_RIGHT**	Sets the direction to ***Right***.
int **DIRECTION_TOP**	Sets the direction to ***Top***.

Constant	Description
int DIRECTION_BOTTOM	Sets the direction to ***Bottom***.
int DIRECTION_TOP_LEFT	Sets the direction to ***Top Left***.
int DIRECTION_TOP_RIGHT	Sets the direction to ***Top Right***.
int DIRECTION_BOTTOM_LEFT	Sets the direction to ***Bottom Left***.
int DIRECTION_BOTTOM_RIGHT	Sets the direction to ***Bottom Right***.
int DIRECTION_FRONT_LEFT	Sets the direction to ***Front Left***.
int DIRECTION_FRONT_RIGHT	Sets the direction to ***Front Right***.
int DIRECTION_BACK_LEFT	Sets the direction to ***Back Left***.
int DIRECTION_BACK_RIGHT	Sets the direction to ***Back Right***.
int DIRECTION_FRONT_TOP	Sets the direction to ***Front Top***.
int DIRECTION_FRONT_BOTTOM	Sets the direction to ***Front Bottom***.
int DIRECTION_BACK_TOP	Sets the direction to ***Back Top***.
int DIRECTION_BACK_BOTTOM	Sets the direction to ***Back Bottom***.
int DIRECTION_FRONT_TOP_LEFT	Sets the direction to ***Front Top Left***.
int DIRECTION_FRONT_TOP_RIGHT	Sets the direction to ***Front Top Right***.
int DIRECTION_FRONT_BOTTOM_LEFT	Sets the direction to ***Front Bottom Left***.
int DIRECTION_FRONT_BOTTOM_RIGHT	Sets the direction to ***Front Bottom Right***.
int DIRECTION_BACK_TOP_LEFT	Sets the direction to ***Back Top Left***.
int DIRECTION_BACK_TOP_RIGHT	Sets the direction to ***Back Top Right***.
int DIRECTION_BACK_BOTTOM_LEFT	Sets the direction to ***Back Bottom Left***.
int DIRECTION_BACK_BOTTOM_RIGHT	Sets the direction to ***Back Bottom Right***.
int DIRECTION_X_AXIS_IMPLODE	Sets the direction to ***X-Axis Implode***.
int DIRECTION_X_AXIS_EXPLODE	Sets the direction to ***X-Axis Explode***.
int DIRECTION_Y_AXIS_IMPLODE	Sets the direction to ***Y-Axis Implode***.
int DIRECTION_Y_AXIS_EXPLODE	Sets the direction to ***Y-Axis Explode***.
int DIRECTION_Z_AXIS_IMPLODE	Sets the direction to ***Z-Axis Implode***.
int DIRECTION_Z_AXIS_EXPLODE	Sets the direction to ***Z-Axis Explode***.
int DIRECTION_RECTANGLE_IMPLODE	Sets the direction to ***Rectangle Implode***.
int DIRECTION_RECTANGLE_EXPLODE	Sets the direction to ***Rectangle Explode***.
int DIRECTION_SQUARE_IMPLODE	Sets the direction to ***Square Implode***.
int DIRECTION_SQUARE_EXPLODE	Sets the direction to ***Square Explode***.
int DIRECTION_ELLIPSE_IMPLODE	Sets the direction to ***Ellipse Implode***.
int DIRECTION_ELLIPSE_EXPLODE	Sets the direction to ***Ellipse Explode***.

Constant	Description
int DIRECTION_CIRCLE_IMPLODE	Sets the direction to ***Circle Implode***.
int DIRECTION_CIRCLE_EXPLODE	Sets the direction to ***Circle Explode***.
int DIRECTION_DIAMOND_IMPLODE	Sets the direction to ***Diamond Implode***.
int DIRECTION_DIAMOND_EXPLODE	Sets the direction to ***Diamond Explode***.
int DIRECTION_BOX_IMPLODE	Sets the direction to ***Box Implode***.
int DIRECTION_BOX_EXPLODE	Sets the direction to ***Box Explode***.
int DIRECTION_CUBE_IMPLODE	Sets the direction to ***Cube Implode***.
int DIRECTION_CUBE_EXPLODE	Sets the direction to ***Cube Explode***.
int DIRECTION_ELLIPSOID_IMPLODE	Sets the direction to ***Ellipsoid Implode***.
int DIRECTION_ELLIPSOID_EXPLODE	Sets the direction to ***Ellipsoid Explode***.
int DIRECTION_SPHERE_IMPLODE	Sets the direction to ***Sphere Implode***.
int DIRECTION_SPHERE_EXPLODE	Sets the direction to ***Sphere Explode***.
int DIRECTION_OCTAHEDRON_IMPLODE	Sets the direction to ***Octahedron Implode***.
int DIRECTION_OCTAHEDRON_EXPLODE	Sets the direction to ***Octahedron Explode***.
int DIRECTION_RADIAL	Sets the direction to ***Radial***.

Example

This macro example switches the direction each second between left and right. It works with the »SCE_Color_Scroll effect, for example.

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

int g_startTime;
int g_run;

void InitEffect()
{
  time t = GetTime();
  g_startTime = t.hour * 3600 + t.min * 60 + t.sec;

  g_run = 0;
  SetDirectionCrossMode(FALSE);      //Deactivate Cross Mode
}

void PreRenderEffect()
{
  time t = GetTime();
```

```
  int t2 = t.hour * 3600 + t.min * 60 + t.sec;

  if(t2 - g_startTime >= 1)
  {
        g_startTime = t2;
        switch(g_run % 2)
        {
        case 0:       SetDirection(DIRECTION_LEFT);      break;
        case 1:       SetDirection(DIRECTION_RIGHT);     break;
        default:      break;
        }
        g_run++;
  }
}

void PostRenderEffect()
{

}

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

5.1.18 Using Look-At Types

Functions Provided For Setting Effect Look-At Types

The following table provides an overview of all functions to modify effect look-at types:

Function	Description
void **SetLookAtType**(int value)	Sets the look-at type *value*, which is the orientation of the effect related to the matrix. See **below** for a list of constants.
int **GetLookAtType**()	Returns the current look-at type. See **below** for a list of constants.

Remarks

The effects sometimes offer just a selection of all the look-at types which are listed below. The function SetLookAtType will only set look-at types which are offered by the effect.

Look-At Type Constants

Constant	Description
int **LOOKAT_FRONT**	Sets the look-at type to ***Front***.
int **LOOKAT_BACK**	Sets the look-at type to ***Back***.
int **LOOKAT_LEFT**	Sets the look-at type to ***Left***.
int **LOOKAT_RIGHT**	Sets the look-at type to ***Right***.
int **LOOKAT_TOP**	Sets the look-at type to ***Top***.
int **LOOKAT_BOTTOM**	Sets the look-at type to ***Bottom***.

Example

This macro example randomly changes the look-at type each second. It works with the »SCE Image or »SCE Ticker / Scrolling Text effect, for example. (In the SCE Image effect, please load at least one image.)

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

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 >= 1)
 {
        g_startTime = t2;
        switch(random(0,5))
        {
        case 0:       SetLookAtType(LOOKAT_FRONT);       break;
        case 1:       SetLookAtType(LOOKAT_BACK); break;
        case 2:       SetLookAtType(LOOKAT_LEFT); break;
        case 3:       SetLookAtType(LOOKAT_RIGHT);       break;
        case 4:       SetLookAtType(LOOKAT_TOP);         break;
        case 5:       SetLookAtType(LOOKAT_BOTTOM);      break;
        default:      break;
        }
```

```
  }
}

void PostRenderEffect()
{

}

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

5.1.19 Using Position Control

Functions Provided For Setting The Position

The following table provides an overview of all functions the effect can use to modify the position:

Function	Description
void **SetPositionX**(float value)	Sets the ***Position X*** *value* in percent of the matrix size.
float **GetPositionX**()	Returns the current ***Position X*** in percent of the matrix size.
void **SetPositionY**(float value)	Sets the ***Position Y*** *value* in percent of the matrix size.
float **GetPositionY**()	Returns the current ***Position Y*** in percent of the matrix size.
void **SetPositionZ**(float value)	Sets the ***Position Z*** *value* in percent of the matrix size.
float **GetPositionZ**()	Returns the current ***Position Z*** in percent of the matrix size.
void **SetPosition**(float x, float y, float z)	Sets the position values ***X, Y, Z*** in percent of the matrix size.
void **SetPixelPositionX**(int value)	Sets the ***Position X*** *value* in pixels.
int **GetPixelPositionX**()	Returns the current ***Position X*** in pixels.
void **SetPixelPositionY**(int value)	Sets the ***Position Y*** *value* in pixels.
int **GetPixelPositionY**()	Returns the current ***Position Y*** in pixels.
void **SetPixelPositionZ**(int value)	Sets the ***Position Z*** *value* in pixels.
int **GetPixelPositionZ**(void)	Returns the current ***Position Z*** in pixels.
void **SetPixelPosition**(int x, int y, int z)	Sets the position values ***X, Y, Z*** in pixels.

5.1.20 Using Size Control

Functions Provided For Setting The Size

The following table provides an overview of all functions the effect can use to modify the size:

Function	Description
void **SetSizeX**(float value)	Sets the Size X (***Width***) *value* in percent of the matrix size.
float **GetSizeX**()	Returns the current Size X (***Width***) in percent of the matrix size.
void **SetSizeY**(float value)	Sets the Size Y (***Height***) *value* in percent of the matrix size.
float **GetSizeY**()	Returns the current Size Y (***Height***) in percent of the matrix size.
void **SetSizeZ**(float value)	Sets the Size Z (***Depth***) *value* in percent of the matrix size.
float **GetSizeZ**()	Returns the current Size Z (***Depth***) in percent of the matrix size.
void **SetSize**(float x, float y, float z)	Sets the size values *x, y, z* (***Width, Height, Depth***) in percent of the matrix size.
void **SetPixelSizeX**(int value)	Sets the Size X (***Width***) *value* in pixels.
int **GetPixelSizeX**()	Returns the current Size X (***Width***) in pixels.
void **SetPixelSizeY**(int value)	Sets the Size Y (***Height***) *value* in pixels.
int **GetPixelSizeY**()	Returns the current Size Y (***Height***) in pixels.
void **SetPixelSizeZ**(int value)	Sets the Size Z (***Depth***) *value* in pixels.
int **GetPixelSizeZ**()	Returns the current Size Z (***Depth***) in pixels.
void **SetPixelSize**(int x, int y, int z)	Sets the size values *x, y, z* (***Width, Height, Depth***) in pixels.

Remarks

Not all effects that are using the Size Control support the functions SetSize and SetPixelSize with several parameters.

5.1.21 Mapping / Tiling / Rotation

Overview

- MADRIX provides advanced mapping functionality, including mapping, tiling, and rotation.
- The following functions are available for the MAS Script Effect as well as for Macros for Effects.
- The Storage Place Macro provides its own functions. Learn more: »**Storage Place Macro: Functions**

Functions

Function	Description	MAS Script	Macros for Effects	Storage Place Macro	Global Macro
int **MapDlgIsMapped**()	Retrieves if the Layer is mapped.	+	+		
void **MapDlgGetMapVector**(float map[])	Retrieves the map settings for the Layer using relative values. The values are saved in a array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapVector**(float x, float y, float w, float h)	Maps the Layer to a certain area of the matrix using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetMapVector3D** (float map[])	Retrieves the map settings for the Layer in 3D using relative values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapVector3D** (float x, float y, float z, float w, float h, float d)	Maps the Layer to a certain area of the matrix in 3D using relative values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		

void **MapDlgGetMapPixel**(int map[])	Retrieves the map settings for the Layer using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)	+	+		
void **MapDlgSetMapPixel**(int x, int y, int w, int h)	Maps the Layer to a certain area of the matrix using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetMapPixel3D**(int map[])	Retrieves the map settings for the Layer in 3D using pixel values. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)	+	+		
void **MapDlgSetMapPixel3D**(int x, int y, int z, int w, int h, int d)	Maps the Layer to a certain area of the matrix in 3D using pixel values. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
int **MapDlgGetMapMode**()	Retrieves the currently used ***Map Mirror Mode*** of the Layer. See **below** for a list of constants.	+	+		
void **MapDlgSetMapMode**(int mode)	Sets the ***Map Mirror Mode*** for the Layer. See **below** for a list of constants.	+	+		
void **MapDlgGetTileVector**(float tile[])	Retrieves the ***Tiling*** settings for the Layer using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTileVector**(float x, float y, float w, float h)	Tiles the Layer to a certain area of the mapping using relative values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		

void **MapDlgGetTileVector3D** (float tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using relative values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTileVector3D** (float x, float y, float z, float w, float h, float d)	Tiles the Layer in a certain area of the mapping in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		
void **MapDlgGetTilePixel**(int tile [])	Retrieves the ***Tiling*** settings for the Layer using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)	+	+		
void **MapDlgSetTilePixel**(int x, int y, int w, int h)	Tiles the Layer in a certain area of the mapping using pixel values. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.	+	+		
void **MapDlgGetTilePixel3D**(int tile[])	Retrieves the ***Tiling*** settings for the Layer in 3D using pixel values. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)	+	+		
void **MapDlgSetTilePixel3D**(int x, int y, int z, int w, int h, int d)	Tiles the Layer in a certain area of the mapping using pixel values. *x, y and z* describe the coordinates. *w, h, and d* describe width, height, and depth.	+	+		

void **MapDlgGetTileOffsetVector** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetVector** (float x, float y)	Sets the ***Tiling Offset*** of the Layer using relative values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgGetTileOffsetVector3D** (float offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using relative values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetVector3D** (float x, float y, float z)	Sets the ***Tiling Offset*** of the Layer in 3D using relative values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
void **MapDlgGetTileOffsetPixel** (int offset[])	Retrieves the ***Tiling Offset*** of the Layer using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)	+	+		
void **MapDlgSetTileOffsetPixel** (int x, int y)	Sets the ***Tiling Offset*** of the Layer using pixel values. *x* describes Offset X, while *y* represents Offset Y.	+	+		
void **MapDlgGetTileOffsetPixel3D**(int offset[])	Retrieves the ***Tiling Offset*** of the Layer in 3D using pixel values. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)	+	+		
void **MapDlgSetTileOffsetPixel3D**(int x, int y, int z)	Sets the ***Tiling Offset*** of the Layer in 3D using pixel values. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.	+	+		
int **MapDlgGetTileMode**()	Retrieves the currently used ***Tile Mode*** of the Layer. See **below** for a list of Tile Mode constants.	+	+		
void **MapDlgSetTileMode**(int mode)	Sets the ***Tile Mode*** of the Layer. See **below** for a list of Tile Mode constants.	+	+		

void **MapDlgGetRotationVector** (float rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values. The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationVector** (float x, float y, float z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using relative values.	+	+		
void **MapDlgGetRotationDegree** (int rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationDegree** (int x, int y, int z)	Sets the ***Rotation*** values for axes *x, y, and z* of the Layer using degree values.	+	+		
float **MapDlgGetRotationXVector**()	Retrieves the ***Rotation*** for the X-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationXVector** (float value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using degree values.	+	+		
int **MapDlgGetRotationXDegree** ()	Retrieves the ***Rotation*** for the X-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationXDegree**(int value)	Sets the ***Rotation*** *value* for the X-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationXMode**()	Retrieves the ***Rotation*** mode for the X-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationXMode** (int mode)	Sets the ***Rotation*** mode for the X-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationYVector**()	Retrieves the ***Rotation*** for the Y-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationYVector** (float value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationYDegree** ()	Retrieves the ***Rotation*** for the Y-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationYDegree**(int value)	Sets the ***Rotation*** *value* for the Y-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationYMode**()	Retrieves the ***Rotation*** mode for the Y-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		

void **MapDlgSetRotationYMode** (int mode)	Sets the ***Rotation*** mode for the Y-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
float **MapDlgGetRotationZVector**()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
void **MapDlgSetRotationZVector** (float value)	Sets the ***Rotation*** value for the Z-axis of the Layer using relative values.	+	+		
int **MapDlgGetRotationZDegree** ()	Retrieves the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
void **MapDlgSetRotationZDegree**(int value)	Sets the ***Rotation*** value for the Z-axis of the Layer using degree values.	+	+		
int **MapDlgGetRotationZMode**()	Retrieves the ***Rotation*** mode for the Z-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgSetRotationZMode** (int mode)	Sets the ***Rotation*** mode for the Z-axis of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
void **MapDlgGetRotationMode** (int mode[])	Retrieves the ***Rotation*** mode for axes *x, y, and z* of the Layer. See **below** for a list of Rotation Mode constants. The values are saved in an array (*mode[]*). ▪ mode[0] = X-coordinate (X-Axis) ▪ mode[1] = Y-coordinate (Y-Axis) ▪ mode[2] = Z-coordinate (Z-Axis)	+	+		
void **MapDlgSetRotationMode** (int x, int y, int z)	Sets the ***Rotation*** M***ode*** for axes *x, y, and z* of the Layer. See **below** for a list of Rotation Mode constants.	+	+		
int **MapDlgGetAntiAliasing**()	Retrieves the ***Anti-Aliasing*** mode of the Layer. See **below** for a list of Anti-Aliasing Mode constants.	+	+		
void **MapDlgSetAntiAliasing**(int aaLevel)	Sets the ***Anti-Aliasing*** mode of the Layer. See **below** for a list of Anti-Aliasing Mode constants.	+	+		

Map Mirror Mode Constants

Constant	Description
int **MAP_MIRROR_NONE**	Sets no Mirror 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 horizontally and vertically.
int **MAP_MIRROR_D**	Mirrors the content of the matrix in the depth.
int **MAP_MIRROR_HD**	Mirrors the content of the matrix horizontally and in the depth.

Constant	Description
int MAP_MIRROR_VD	Mirrors the content of the matrix vertically and in the depth.
int MAP_MIRROR_HVD	Mirrors the content of the matrix horizontally, vertically, and in the depth.

Map Tile Mode Constants

Constant	Description
int MAP_TILE_NONE	Sets no Tile Mode.
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 horizontally and vertically.
int MAP_TILE_MIRROR_D	Mirrors tiles in the depth.
int MAP_TILE_MIRROR_HD	Mirrors tiles horizontally and in the depth.
int MAP_TILE_MIRROR_VD	Mirrors tiles vertically and in the depth.
int MAP_TILE_MIRROR_HVD	Mirrors tiles horizontally, vertically, and in the depth.

Map Rotation Mode Constants

Constant	Description
int MAP_ROTATION_FIXED	The rotation value is fixed and not animated.
int MAP_ROTATION_LOOP	The rotation value is used for a looped animation.

Map Anti-Aliasing Mode Constants

Constant	Description
int MAP_AA_NONE	Sets no anti-aliasing.
int MAP_AA_2X	Sets simple anti-aliasing (requires some performance).

Constant	Description
int **MAP_AA_4X**	Sets complex anti-aliasing (requires a lot performance).

5.2 Static Color Effects (SCE)

5.2.1 SCE Bounce

Functions Provided By SCE Bounce

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetCount**(int value)	Sets the object ***Count*** *value*, which is the number of the displayed shapes. Valid values for *value* range from *1* to *100*.
int **GetCount**()	Returns the current object ***Count***.
void **SetPoints**(int value)	Sets the ***Points*** count *value*, which is the number of points per object. This only applies to the line and the curve shape. Valid values for *value* range from *1* to *20*.
int **GetPoints**()	Returns the current ***Points*** count.

void **SetWidth**(float value)	Sets the ***Width*** *value*, which is the width of the shapes, in percent of the matrix size (depending on the shape alignment). The width *value* means the minimum shape width if the shape width distribution is not uniform. Valid values for *value* range from *0.01* to *100*.
float **GetWidth**()	Returns the current ***Width*** of the shapes in percent. The return value means the minimum shape width if the shape width distribution is not uniform.
void **SetPixelWidth**(int value)	Sets the ***Width*** *value*, which is the width of the shapes, in pixels. The width *value* means the minimum shape width if the shape width distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the shape alignment).
int **GetPixelWidth**()	Returns the current ***Width*** of the shapes in pixels. The return value means the minimum shape width if the shape width distribution is not uniform.
void **SetWidthMax**(float value)	Sets the ***Maximum Width*** *value*, which is the maximum width of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum width *value* only applies if the shape width distribution is not uniform. Valid values for *value* range from *0.01* to *100* .
float **GetWidthMax**()	Returns the current ***Maximum Width*** of the shapes in percent.
void **SetPixelWidthMax**(int value)	Sets the ***Maximum Width*** *value*, which is the maximum width of the shapes, in pixels. The maximum width *value* only applies if the shape width distribution is not uniform. Valid values for *value* range from 1 to the matrix size (depending on the shape alignment).
int **GetPixelWidthMax**()	Returns the current ***Maximum Width*** of the shapes in pixels.
void **SetWidthDistribution**(int value)	Sets the ***Width Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape width is applied to the shapes. See »**here** for a list of constants.
int **GetWidthDistribution**()	Returns the current ***Width Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomWidth**()	Randomizes the widths of the shapes. Same as **SetWidthDistribution** (**DIST_RND**).
void **SetHeight**(float value)	Sets the ***Height*** *value*, which is the height of the shapes, in percent of the matrix size (depending on the shape alignment). The height *value* means the minimum shape height if the shape height distribution is not uniform. Valid values for *value* range from *0.01* to *100*.
float **GetHeight**()	Returns the current ***Height*** of the shapes in percent. The return value means the minimum shape height if the shape height distribution is not uniform.
void **SetPixelHeight**(int value)	Sets the ***Height*** *value*, which is the height of the shapes, in pixels. The height *value* means the minimum shape height if the shape height distribution is not uniform. Valid values for *value* range from 1 to the matrix size (depending on the shape alignment).
int **GetPixelHeight**()	Returns the current ***Height*** of the shapes in pixels. The return value means the minimum shape height if the shape height distribution is not uniform.
void **SetHeightMax**(float value)	Sets the ***Maximum Height*** *value*, which is the maximum height of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum height *value* only applies if the shape height distribution is not uniform. Valid values for *value* range from *0.01* to *100* .
float **GetHeightMax**()	Returns the current ***Maximum Height*** of the shapes in percent.

void **SetPixelHeightMax**(int value)	Sets the ***Maximum Height*** *value*, which is the maximum height of the shapes, in pixels. The maximum height *value* only applies if the shape height distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the shape alignment).
int **GetPixelHeightMax**()	Returns the current ***Maximum Height*** of the shapes in pixels.
void **SetHeightDistribution**(int value)	Sets the ***Height Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape height is applied to the shapes. See »**here** for a list of constants.
int **GetHeightDistribution**()	Returns the current ***Height Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomHeight**()	Randomizes the heights of the shapes. Same as **SetHeightDistribution**(**DIST_RND**).
void **SetDepth**(float value)	Sets the ***Depth*** *value*, which is the depth of the shapes, in percent of the matrix size (depending on the shape alignment). The depth *value* means the minimum shape depth if the shape depth distribution is not uniform. Valid values for *value* range from *0.01* to *100*.
float **GetDepth**()	Returns the current ***Depth*** of the shapes in percent. The return value means the minimum shape depth if the shape depth distribution is not uniform.
void **SetPixelDepth**(int value)	Sets the ***Depth*** *value*, which is the depth of the shapes, in pixels. The depth *value* means the minimum shape depth if the shape depth distribution is not uniform. Valid values for *value* range from 1 to the matrix size (depending on the shape alignment).
int **GetPixelDepth**()	Returns the current ***Depth*** of the shapes in pixels. The return value means the minimum shape depth if the shape depth distribution is not uniform.
void **SetDepthMax**(float value)	Sets the ***Maximum Depth*** *value*, which is the maximum depth of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum depth *value* only applies if the shape depth distribution is not uniform. Valid values for *value* range from *0.01* to *100* .
float **GetDepthMax**()	Returns the current ***Maximum Depth*** of the shapes in percent.
void **SetPixelDepthMax**(int value)	Sets the ***Maximum Depth*** *value*, which is the maximum depth of the shapes, in pixels. The maximum depth *value* only applies if the shape depth distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the shape alignment).
int **GetPixelDepthMax**()	Returns the current ***Maximum Depth*** of the shapes in pixels.
void **SetDepthDistribution**(int value)	Sets the ***Depth Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape depth is applied to the shapes. See »**here** for a list of constants.
int **GetDepthDistribution**()	Returns the current ***Depth Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomDepth**()	Randomizes the depths of the shapes. Same as **SetDepthDistribution**(**DIST_RND**).
void **SetFadeOut**(int value)	Sets the ***Fade Out*** *value*, which defines how long the progressively drawn shapes stay visible. Valid values for *value* range from *0* to *1000*.
int **GetFadeOut**()	Returns the current ***Fade Out***.
void **SetCollision**(int value)	Use *value* 1 (**TRUE**) to activate ***Collision***, which means that colliding shapes bounce of each other. This applies to all shapes except the line and the curve shape. Use *value* 0 (**FALSE**) to deactivate collision.

int **GetCollision**()	Returns 1 (**TRUE**) if ***Collision*** is activated, otherwise 0 (**FALSE**).
void **ToggleCollision**()	Activates or deactivates ***Collision***, depending on the current state.

5.2.2 SCE Capture

Functions Provided By SCE Capture

This effect uses the following functions:

- This Effect uses Directions. Learn more »**Using Direction**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**
- This Effect uses the Color Controls. Learn more »**Using Color Controls**
- This Effect uses the Position Control. Learn more »**Using Position Control**

Function	Description
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.
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 *angle* are *0*, *90*, *180*, and *270*.
int **GetRotation**()	Returns the currently set ***Rotation***.
void **SetTiling**(int enable)	Disables ***Tile Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Tile Mode*** will be enabled via 1 (**TRUE**).
int **GetTiling**()	Returns 1 (**TRUE**) if ***Tile Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleTiling**()	Enables ***Tile Mode*** or uses the default setting, depending on the current state.
void **SetSeamless**(int enable)	Disables ***Seamless*** if *enable* is 0 (**FALSE**), otherwise it will be enabled.
int **GetSeamless**()	Returns 1 (**TRUE**) if ***Seamless*** is enabled, otherwise 0 (**FALSE**).
void **ToggleSeamless**()	Enables ***Seamless*** or uses the default setting, depending on the current state.
float **GetImageWidth**()	Returns the image width of the current image in percent of the matrix width.

float **GetImageHeight**()	Returns the image height of the current image in percent of the matrix height.
float **GetImageDepth**()	Returns the image depth of the current image in percent of the matrix depth.
int **GetImagePixelWidth**()	Returns the image width of the current image in pixels.
int **GetImagePixelHeight**()	Returns the image height of the current image in pixels.
int **GetImagePixelDepth**()	Returns the image depth of the current imagein pixel.
float **GetCurrentPositionX**()	Returns the X-position of the current image in percent of the matrix width.
float **GetCurrentPositionY**()	Returns the Y-position of the current image in percent of the matrix height.
float **GetCurrentPositionZ**()	Returns the Z-position of the current image in percent of the matrix depth.
int **GetCurrentPixelPositionX**()	Returns the X-position of the current image in pixels.
int **GetCurrentPixelPositionY**()	Returns the Y-position of the current image in pixels.
int **GetCurrentPixelPositionZ**()	Returns the Z-position of the current image in pixels.
int **GetLoaded**()	Returns 1 (**TRUE**) if a capture device is selected and started, otherwise 0 (**FALSE**).
void **SetStretchMode**(int mode)	Sets the ***Stretch*** mode. See below for details.
int **GetStretchMode**()	Returns the ***Stretch*** mode. See below for details.
void **SetGrayscale**(int enable)	Disables ***Grayscale*** if *enable* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**).
int **GetGrayscale**()	Returns 1 (**TRUE**) if ***Stretch*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleGrayscale**()	Enables ***Stretch*** mode or uses the default setting, depending on the current state.
void **SetRgbw**(int enable)	Disables the ***RGB-To-RGBW*** mode if *enable* is 0 (**FALSE**), otherwise it can be enabled using 1 (**TRUE**).
int **GetRgbw**()	Returns 1 (**TRUE**) if ***RGB-To-RGBW*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleRgbw**()	Enables ***RGB-To-RGBW*** mode or uses the default setting, depending on the current state.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See below for details.
int **GetFilteringMode**()	Returns the ***Filtering*** mode. See below for details.
SetAlphaBlending(bool enable)	Disables ***Alpha*** if *enable* is 0 (**FALSE**), otherwise it can be enabled using 1 (**TRUE**).
bool **GetAlphaBlending**()	Returns 1 (**TRUE**) if ***Alpha*** is enabled, otherwise 0 (**FALSE**).

Stretch Mode Constants

Value	Description
int `STRETCH_MODE_NONE`	Sets the stretch mode to ***None*** (no stretching).
int `STRETCH_MODE_MATRIX`	Sets the stretch mode to ***Matrix*** (stretches to the current aspect ratio of the matrix).
int `STRETCH_MODE_ORIGINAL`	Sets the stretch mode to ***Original*** (stretches to the original aspect ratio of the source).
int `STRETCH_MODE_4_TO_3`	Sets the stretch mode to ***4:3***.
int `STRETCH_MODE_16_TO_9`	Sets the stretch mode to ***16:9***.

Filtering Mode Constants

Value	Description
int `FILTERING_MODE_NEAREST_NEIGHBOR`	Sets the Filtering Mode to ***None*** (no filtering).
int `FILTERING_MODE_LINEAR`	Sets the Filtering Mode to ***Linear*** (may require additional performance).

5.2.3 SCE Clouds

Functions Provided By SCE Clouds

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Gradient Dialog. Learn more »Using Color Gradient Dialog
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetFrequency**(float value)	Sets the ***Frequency*** for the effect.
float **GetFrequency**()	Returns the current ***Frequency*** value.
void **SetStretchX**(float value)	Sets the ***Stretch*** factor regarding X.
float **GetStretchX**()	Returns the current ***Stretch*** factor for X.
void **SetStretchY**(float value)	Sets the ***Stretch*** factor regarding Y.
float **GetStretchY**()	Returns the current ***Stretch*** factor for Y.

void **SetStretchZ**(float value)	Sets the ***Stretch*** factor regarding Z.
float **GetStretchZ**()	Returns the current ***Stretch*** factor for Z.
void **SetStretch**(float value, float value, float value)	Sets the ***Stretch*** value for X, Y, and Z.
void **SetContrast**(int value)	Sets the ***Contrast*** value for the effect.
int **GetContrast**()	Returns the current ***Contrast*** value.
void **SetDetail**(int value)	Sets the ***Detail*** value for the effect.
int **GetDetail**()	Returns the current ***Detail*** value.
void **SetDistinction**(int value)	Sets the ***Distinction*** value for the effect.
int **GetDistinction**()	Returns the current ***Distinction*** value.
void **SetDisplayMode**(int mode)	Sets the ***Display Mode***. See below for a list of constants.
int **GetDisplayMode**()	Returns the current ***Display Mode***. See below for a list of constants.
void **Generate**()	Creates the effect anew.

Mode Constants

This effect offers various modes. The function SetDisplayMode can be used to change them. The following values can be used as parameter:

Constant	Description
int `DISPLAY_MODE_LOW_CONTRAST`	Sets the display mode ***Low Contrast***.
int `DISPLAY_MODE_MEDIUM_CONTRAST`	Sets the display mode ***Medium Contrast***.
int `DISPLAY_MODE_HIGH_CONTRAST`	Sets the display mode ***High Contrast***.
int `DISPLAY_MODE_OUTLINE`	Sets the display mode ***Outline***.
int `DISPLAY_MODE_CLUSTER`	Sets the display mode ***Cluster***.

5.2.4 SCE Color

Functions Provided By SCE Color

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls

5.2.5 SCE Color Change

Functions Provided By SCE Color Change

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetChangeTime**(float value)	Sets the fade ***Time*** for the effect in seconds.
float **GetChangeTime**()	Returns the fade ***Time*** of the effect in seconds.

5.2.6 SCE Color Scroll

Functions Provided By SCE Color Scroll

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the the BPM Control. Learn more »Using BPM Control

Function	Description

void **SetStepWidth**(float value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using percentage values.
float **GetStepWidth**()	Returns the ***Step*** in percentage.
void **SetPixelStepWidth**(int value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using pixel values.
int **GetPixelStepWidth**()	Returns the ***Step*** in pixels.
void **SetColorWidth**(float value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using percentage values.
float **GetColorWidth**()	Returns the ***Color Width***, the width of a stripe, in percentage.
void **SetPixelColorWidth**(int value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using pixel values.
int **GetPixelColorWidth**()	Returns the ***Color Width***, the width of a stripe, in pixels.
void **SetCrossWidth**(float value)	Sets the ***Cross Width*** *value* using percentage values.
float **GetCrossWidth**()	Returns the ***Cross Width*** in percentage.
void **SetPixelCrossWidth**(int value)	Sets the ***Cross Width*** *value* using pixel values.
int **GetPixelCrossWidth**()	Returns the ***Cross Width*** in pixels.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.
void **SetCrossAxes**(int value)	Sets the ***Cross Axis*** *value*. See below for a list of constants.
int **GetCrossAxes**()	Returns the ***Cross Axis***. See below for a list of constants.

Cross Axis Constants

Constant	Description
int `AXIS_ONE`	Sets the direction cross mode to ***Axis 1***.
int `AXIS_TWO`	Sets the direction cross mode to ***Axis 2***.
int `AXES_ONE_AND_TWO`	Sets the direction cross mode to ***Axes 1 And 2***.

5.2.7 SCE Counter

Functions Provided By SCE Counter

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Position Control. Learn more »Using Position Control
- This Effect uses the String Table. Learn more »Using String Table

Function	Description
void **SetPitch**(float value)	Sets the ***Pitch*** value in percent.
float **GetPitch**()	Returns the current ***Pitch*** value.
void **SetScaleX**(float value)	Sets the ***Scale*** factor regarding X.
float **GetScaleX**()	Returns the current ***Scale*** factor for X.
void **SetScaleY**(float value)	Sets the ***Scale*** factor regarding Y.
float **GetScaleY**()	Returns the current ***Scale*** factor for Y.
void **SetScaleZ**(float value)	Sets the ***Scale*** factor regarding Z.
float **GetScaleZ**()	Returns the current ***Scale*** factor for Z.
void **SetScale**(float value, float value, float value)	Sets the ***Scale*** value for X, Y, and Z.
void **SetFadeIn**(int value)	Sets the first *int* value of the ***Fade*** slider.
int **GetFadeIn**()	Returns the current first value of the ***Fade*** slider.
void **SetFadeOut**(int value)	Sets the second *int* value of the ***Fade*** slider.
int **GetFadeOut**()	Returns the current second value of the ***Fade*** slider.
void **SetInterpolationIn**(int value)	Sets the first *int* value of the ***Interpolation*** slider.
int **GetInterpolationIn**()	Returns the current first value of the ***Interpolation*** slider.
void **SetInterpolationOut**(int value)	Sets the second *int* value of the ***Interpolation*** slider.
int **GetInterpolationOut**()	Returns the current second value of the ***Interpolation*** slider.
void **SetAutoStart**(int value)	Sets the ***Autostart*** option. Use 1 (**TRUE**) to activate Autostart. Use 0 (**FALSE**) to deactivate it.
int **GetAutoStart**()	Returns 1 (**TRUE**) if ***Autostart*** is activated, otherwise 0 (**FALSE**).
void **ToggleAutoStart**()	Toggles ***Autostart***.

void **Start**()	Forces a start.
void **SetEndMode**(int mode)	Sets the *mode* for the last element. See **below** for a list of constants.
int **GetEndMode**()	Returns the current mode for the last element.
void **SetTextAlignmentHorizontal**(int alignment)	Sets the horizontal text *alignment* of elements. See **below** for a list of constants.
int **GetTextAlignmentHorizontal**()	Returns the current horizontal text alignment of elements.
void **SetTextAlignmentVertical**(int alignment)	Sets the vertical text *alignment* of elements. See **below** for a list of constants.
int **GetTextAlignmentVertical**()	Returns the current vertical text alignment of elements.
void **SetInterpolationType1**(int type)	Sets the ***Interpolation Type 1***. See **below** for a list of constants.
int **GetInterpolationType1**()	Returns the current ***Interpolation Type 1***.
void **SetInterpolationType2**(int type)	Sets the ***Interpolation Type 2***. See **below** for a list of constants.
int **GetInterpolationType2**()	Returns the current ***Interpolation Type 2***.
void **SetRotation**(int angle)	Sets the ***Rotation*** globally for all elements of the String Table. It is possible to rotate the elements by multiples of 90°. This function is equal to the rotation button provided by the GUI of the effect. Valid values for *angle* are 0, 90, 180, and 270.
int **GetRotation**()	Returns the current ***Rotation*** value.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See **below** for details.
mode **GetFilteringMode**()	Returns the ***Filtering*** mode.
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.

Last Element Mode Constants

Constant	Description
int **ENDMODE_MOVE**	Sets the Last Element Mode to ***Move***.
int **ENDMODE_STAY**	Sets the Last Element Mode to ***Stay***.
int **ENDMODE_CLEAR**	Sets the Last Element Mode to ***Clear***.

Horizontal Text Alignment Constants

Constant	Description
int TEXT_ALIGNMENT_HORIZONTAL_LEFT	Sets the Horizontal Text Alignment to ***Left***.
int TEXT_ALIGNMENT_HORIZONTAL_CENTER	Sets the Horizontal Text Alignment to ***Center***.
int TEXT_ALIGNMENT_HORIZONTAL_RIGHT	Sets the Horizontal Text Alignment to ***Right***.

Vertical Text Alignment Constants

Constant	Description
int TEXT_ALIGNMENT_VERTICAL_TOP	Sets the Vertical Text Alignment to ***Top***.
int TEXT_ALIGNMENT_VERTICAL_CENTER	Sets the Vertical Text Alignment to ***Center***.
int TEXT_ALIGNMENT_VERTICAL_BOTTOM	Sets the Vertical Text Alignment to ***Bottom***.

Filtering Mode Constants

Value	Description
int FILTERING_MODE_NEAREST_NEIGHBOR	Sets the Filtering Mode to ***None*** (no filtering).
int FILTERING_MODE_LINEAR	Sets the Filtering Mode to ***Linear*** (may require additional performance).

Interpolation Type Constants

Constant	Description
int INTERPOLATION_TYPE_LINEAR	Sets the Interpolation Type to ***Linear***.

int **INTERPOLATION_TYPE_EASE_BOUNCE_IN**	Sets the Interpolation Type to ***Ease In Bounce***.
int **INTERPOLATION_TYPE_EASE_BOUNCE_OUT**	Sets the Interpolation Type to ***Ease Out Bounce***.
int **INTERPOLATION_TYPE_EASE_BOUNCE_INOUT**	Sets the Interpolation Type to ***Ease In Out Bounce***.
int **INTERPOLATION_TYPE_EASE_CIRC_IN**	Sets the Interpolation Type to ***Ease In Circular***.
int **INTERPOLATION_TYPE_EASE_CIRC_OUT**	Sets the Interpolation Type to ***Ease Out Circular***.
int **INTERPOLATION_TYPE_EASE_CIRC_INOUT**	Sets the Interpolation Type to ***Ease In Out Circular***.
int **INTERPOLATION_TYPE_EASE_CUBIC_IN**	Sets the Interpolation Type to ***Ease In Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_OUT**	Sets the Interpolation Type to ***Ease Out Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_INOUT**	Sets the Interpolation Type to ***Ease In Out Cubic***.
int **INTERPOLATION_TYPE_EASE_SINE_IN**	Sets the Interpolation Type to ***Ease In Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_OUT**	Sets the Interpolation Type to ***Ease Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_INOUT**	Sets the Interpolation Type to ***Ease In Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_EXPO_IN**	Sets the Interpolation Type to ***Ease In Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_OUT**	Sets the Interpolation Type to ***Ease Out Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_INOUT**	Sets the Interpolation Type to ***Ease In Out Exponential*** .

5.2.8 SCE Credits

Functions Provided By SCE Credits

This effect uses the following functions:

- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Position Control. Learn more »Using Position Control
- This Effect uses the String Table. Learn more »Using String Table

Function	Description
void **SetColorSelect**(int mode)	Sets the *color mode*. See below for a list of constants.
int **GetColorSelect**()	Returns the current *color mode*.
void **SetAutoStart**(int value)	Sets the ***Autostart*** option. Use 1 (**TRUE**) to activate Autostart. Use 0 (**FALSE**) to deactivate it.
int **GetAutoStart**()	Returns 1 (**TRUE**) if ***Autostart*** is activated, otherwise 0 (**FALSE**).
void **ToggleAutoStart**()	Toggles ***Autostart***.
void **Start**()	Forces a start.
void **SetEndMode**(int mode)	Sets the *mode* for the last shown text. See below for a list of constants.
int **GetEndMode**()	Returns the current mode for the last shown text.
void **SetTextAlignmentHorizontal**(int alignment)	Sets the *horizontal text alignment* of elements. See below for a list of constants.
int **GetTextAlignmentHorizontal**()	Returns the current horizontal text alignment of elements.
void **SetTextAlignmentVertical**(int alignment)	Sets the *vertical text alignment* of elements. See below for a list of constants.
int **GetTextAlignmentVertical**()	Returns the current vertical text alignment of elements.
void **SetShuffle**(float value)	Sets the ***Shuffle*** parameter.
float **GetShuffle**()	Returns the currently set ***Shuffle*** value.
void **SetShuffleCount**(int value)	Sets the Shuffle ***Count***.
int **GetShuffleCount**()	Returns the currently set ***Count*** value.
void **SetLineBreak**(float value)	Sets the ***Line Break*** parameter.
float **GetLineBreak**()	Returns the currently set ***Line Break*** value.
void **SetPageBreak**(float value)	Sets the ***Page Break*** parameter.
float **GetPageBreak**()	Returns the currently set ***Page Break*** value.

void **SetPageFade**(float value)	Sets the ***Page Fade*** parameter.
float **GetPageFade**()	Returns the currently set ***Page Fade*** value.
void **SetRotation**(int angle)	Sets the ***Rotation*** globally for all elements of the String Table. It is possible to rotate the elements by multiples of 90°. This function is equal to the rotation button provided by the GUI of the effect. Valid values for *angle* are 0, 90, 180, and 270.
int **GetRotation**()	Returns the current ***Rotation*** value.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See below for a list of constants.
mode **GetFilteringMode**()	Returns the ***Filtering*** mode.

Color Mode Constants

Constant	Description
int `COLORSELECT_PAGE`	Sets the color mode to ***Page***.
int `COLORSELECT_LINE`	Sets the color mode to ***Line***.

Last Element Mode Constants

Constant	Description
int `ENDMODE_STAY`	Sets the Last Element Mode to ***Stay***.
int `ENDMODE_CLEAR`	Sets the Last Element Mode to ***Clear***.

Horizontal Text Alignment Constants

Constant	Description
int `TEXT_ALIGNMENT_HORIZONTAL_LEFT`	Sets the Horizontal Text Alignment to ***Left***.
int `TEXT_ALIGNMENT_HORIZONTAL_CENTER`	Sets the Horizontal Text Alignment to ***Center***.
int `TEXT_ALIGNMENT_HORIZONTAL_RIGHT`	Sets the Horizontal Text Alignment to ***Right***.

Vertical Text Alignment Constants

Constant	Description
int `TEXT_ALIGNMENT_VERTICAL_TOP`	Sets the Vertical Text Alignment to ***Top***.
int `TEXT_ALIGNMENT_VERTICAL_CENTER`	Sets the Vertical Text Alignment to ***Center***.
int `TEXT_ALIGNMENT_VERTICAL_BOTTOM`	Sets the Vertical Text Alignment to ***Bottom***.

Filtering Mode Constants

Value	Description
int `FILTERING_MODE_NEAREST_NEIGHBOR`	Sets the Filtering Mode to ***None*** (no filtering).
int `FILTERING_MODE_LINEAR`	Sets the Filtering Mode to ***Linear*** (may require additional performance).

5.2.9 SCE Drops

Functions Provided By SCE Drops

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Shape Table. Learn more »Using Shape Table
- This Effect uses Shape Rotation. Learn more »Using Shape Rotation
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description

void **SetSpeedVariance**(int value)	Sets the ***Speed Variance*** *value*, which means a random value between *0* and *500* percent. This value will multiply with the current BPM value.
int **GetSpeedVariance**()	Returns the current ***Speed Variance***.
void **SetLength**(float value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in percent of the matrix size (depending on the direction). The ***Length*** *value* means the minimum length if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLength**()	Returns the current ***Length*** in percent. The return value means the minimum length if the length distribution mode is not uniform.
void **SetPixelLength**(int value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in pixels. The ***Length*** *value* means the minimum length if the length distribution mode is not uniform.
int **GetPixelLength**()	Returns the current ***Length*** in pixels. The return value means the minimum length if the length distribution mode is not uniform.
void **SetLengthMin**(float value)	Is the same as **SetLength**.
float **GetLengthMin**()	Is the same as **GetLength**.
void **SetPixelLengthMin**(int value)	Is the same as **SetPixelLength**.
int **GetPixelLengthMin**()	Is the same as **GetPixelLength**.
void **SetLengthMax**(float value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in percent of the matrix size (depending on the direction). The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthMax**()	Returns the current ***Maximum Length*** in percent.
void **SetPixelLengthMax**(int value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in pixels. The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform.
int **GetPixelLengthMax**()	Returns the current ***Maximum Length*** in pixels.
void **SetLengthDistribution**(int value)	Sets the ***Length Distribution*** *value*, which defines how the range between the current minimum and the current maximum length is applied to the traces of the shapes. See »**here** for a list of constants.
int **GetLengthDistribution**()	Returns the current ***Length Distribution*** of the traces of the shapes. See »**here** for a list of constants.
void **SeedRandomLength**()	Randomizes the lengths of the traces of the shapes. Same as **SetLengthDistribution**(**DIST_RND**).
void **SetLengthOffset**(float value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthOffset**()	Returns the current ***Length Offset*** in percent.
void **SetPixelLengthOffset**(int value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in pixels.
int **GetPixelLengthOffset**()	Returns the current ***Length Offset*** in pixels.
void **SetRotationOffset**(float value)	Sets the ***Rotation Offset*** *value* between the shapes of the trace. Valid values for *value* range from -180.0 to 180.0.
float **GetRotationOffset**()	Returns the current ***Rotation Offset***.

void **SetPitch1**(float value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch1**()	Returns the current ***Pitch*** for axis one in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch1**()	Returns the current ***Pitch*** for axis one in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch2**()	Returns the current ***Pitch*** for axis two in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch2**()	Returns the current ***Pitch*** for axis two in pixels.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shape's positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shape's positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shape's positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shape's positions or uses the default setting, depending on the current state.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.
void **SetCount**(int value)	Sets the ***Count*** *value*, which defines how much shapes are displayed at once. Valid values for *value* range from *1* to *100*.
int **GetCount**()	Returns the current ***Count***.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See below for a list of constants.

int **GetGeneratorType**()	Returns the current ***Generator Type***. See below for a list of constants.
void **SetPeak**(float)	Sets the ***Peak*** value. Valid values range from 0.00 to 100.00.
float **GetPeak**()	Returns the currently set ***Peak*** value.

Generator Type Constants

Constant	Description
int `GENERATOR_TYPE_RANDOM`	Sets the generator type to ***Random***.
int `GENERATOR_TYPE_LINEAR`	Sets the generator type to ***Linear***.
int `GENERATOR_TYPE_PING_PONG`	Sets the generator type to ***Ping Pong***.

5.2.10 SCE Explosions

Functions Provided By SCE Explosions

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetExplosionSize**(int size)	Sets the ***Explosion Size*** of a single explosion.
int **GetExplosionSize**()	Returns the current ***Explosion Size*** of a single explosion.
void **SetShapeSize**(int size)	Sets the ***Shape Size*** of single objects.
int **GetShapeSize**()	Returns the current ***Shape Size*** of single objects.
void **SetShapeCount**(int count)	Sets the number of objects within an explosion.
int **GetShapeCount**()	Returns the currently set number of objects of an explosion.
void **SetFadeOut**(int fadeout)	Sets the size of a ***Fade Out*** tail.
int **GetFadeOut**()	Returns the currently set ***Fade Out*** value.
void **SetGravity**(float gravity)	Sets the ***Gravity*** value for the effect.

float **GetGravity**()	Returns the currently set ***Gravity***.
void **Detonate**(int explPosX, int explPosY, int explPosZ, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates an explosion. *explPosX* is the end-coordinate of the explosion in X, *explPosY* is the end-coordinate of the explosion in Y, *explPosZ* is the end-coordinate of the explosion in Z, *ParticleCtn* is the number of objects, *explSize* is the size of the explosion, *explShape* is the type of the explosion, *drawShape* is the shape of the objects, *Col* is the color of the effect, *sparkleCol* is the color of the sparkle part.
void **FireRocket**(int posX, int posY, int posZ, int explPosX, int explPosY, int explPosZ, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates a firework. *posX* is the X-coordinate, *posY* is the Y-coordinate, *posZ* is the Z-coordinate, *explPosX* is the end-coordinate of the fireworks explosion in X, *explPosY* is the end-coordinate of the fireworks explosion in Y, *explPosZ* is the end-coordinate of the fireworks explosion in Z, *ParticleCtn* is the number of objects, *explSize* is the size of the fireworks explosion, *explShape* is the type of the fireworks explosion, *drawShape* is the shape of the objects, *Col* is the color of the effect, *sparkleCol* is the color of the sparkle part.
void **SetSparkleColor**(int index, color c)	Sets the color with the specified *index* in the Sparkle Color Table to the given color value. If the index is out of range, nothing happens.
color **GetSparkleColor**(int index)	Returns the color with the specified *index* in the Sparkle Color Table. If the index is out of range, black is returned.
int **GetSparkleColorCount**()	Returns the current number of colors in the Sparkle Color Table.
void **AddSparkleColor**(int index, color c)	Adds another color to the Sparkle Color Table at the specified *index* position. If the index is 0, the new color is added to the first position. If the index is equal to or greater than the current number of colors, the new color is added at the end.
void **RemoveSparkleColor**(int index)	Removes the color at the specified *index* from the Sparkle Color Table. If the given index is out of range, nothing happens.
void **SetSparkleColorMode**(int mode)	Sets the ***Color Mode*** for the Sparkle Color Table. See **here** for details.
int **GetSparkleColorMode**()	Returns the current ***Color Mode*** for the Sparkle Color Table.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
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.
int **GetRocketCount**()	Returns the currently set number of rockets or explosions.
void **SetExplosionMode**(int mode)	Sets the ***Explosion Mode***. See **below** for details.
int **GetExplosionMode**()	Returns which ***Explosion Mode*** is set.
void **SetExplosionShape**(int shape)	Sets the ***Explosion Shape***. See **below** for details.
int **GetExplosionShape**()	Returns which ***Explosion Shape*** is set.

Explosion Mode Constants

This effect uses various explosion modes. The function SetExplosionMode can be used to change them. The following values can be used as parameter:

Constant	Description
int MODE_EXPLOSIONS	Sets the ***Explosion*** mode.
int MODE_FIREWORKS	Sets the ***Fireworks*** mode.

Explosion Shape Constants

This effect uses various explosion shapes. The function SetExplosionShape can be used to change them. The following values can be used as parameter:

Constant	Description
int EXPLOSION_SHAPE_SPHERE	Sets the sphere type of explosion.
int EXPLOSION_SHAPE_SPHERE_GLOW	Sets the glowing sphere type of explosion.
int EXPLOSION_SHAPE_SPIRAL	Sets the spiral type of explosion.
int EXPLOSION_SHAPE_RADIAL	Sets the radial type of explosion.
int EXPLOSION_SHAPE_DIAMOND	Sets the diamond type of explosion.
int EXPLOSION_SHAPE_STAR	Sets the star type of explosion.
int EXPLOSION_SHAPE_RANDOM	Sets a random type of explosion.

5.2.11 SCE Fill Drops

Functions Provided By SCE Fill Drops

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetColorMix**(int value)	Use *value* 1 (**TRUE**) to activate the ***Color Mix***. When ***Color Mix*** is activated, every shape has its own color depending on the »Color Table Mode. Use *value* 0 (**FALSE**) to use the default setting.
int **GetColorMix**()	Returns 1 (**TRUE**) if the ***Color Mix*** is enabled, otherwise 0 (**FALSE**).
void **ToggleColorMix**()	Enables or disables the ***Color Mix*** depending on the current state.
void **SetFillTime**(float value)	Sets the ***Fill Time*** *value* which defines how long is takes to fill the whole matrix with shapes.
float **GetFillTime**()	Returns the current ***Fill Time***.
void **SetDuration**(float value)	Sets the ***Duration*** *value* which defines how long the completely filled matrix is displayed.
float **GetDuration**()	Returns the current ***Duration***.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, in percent.
float **GetPitch1**()	Returns the ***Pitch 1*** *value*, in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, in pixels.

int **GetPixelPitch1**()	Returns the ***Pitch 1*** *value*, in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, in percent.
float **GetPitch2**()	Returns the ***Pitch 2*** *value*, in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, in pixels.
int **GetPixelPitch2**()	Returns the ***Pitch 2*** *value*, in pixels.

5.2.12 SCE Fill Random

Functions Provided By SCE Fill Random

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetColorMix**(int value)	Use *value* 1 (**TRUE**) to activate the ***Color Mix***. When ***Color Mix*** is activated, every shape has its own color depending on the »Color Table Mode. Use *value* 0 (**FALSE**) to use the default setting.
int **GetColorMix**()	Returns 1 (**TRUE**) if the ***Color Mix*** is enabled, otherwise 0 (**FALSE**).
void **ToggleColorMix**()	Enables or disables the ***Color Mix*** depending on the current state.
void **SetFillTime**(float value)	Sets the ***Fill Time*** *value* which defines how long is takes to fill the whole matrix with shapes.

float **GetFillTime**()	Returns the current ***Fill Time***.
void **SetDuration**(float value)	Sets the ***Duration*** *value* which defines how long the completely filled matrix is displayed.
float **GetDuration**()	Returns the current ***Duration***.
void **SetPitchX**(float value)	Sets the ***Pitch X*** *value* in percent which defines the X-aligned distance between shapes.
float **GetPitchX**()	Returns the current ***Pitch X*** in percent.
void **SetPitchY**(float value)	Sets the ***Pitch Y*** *value* in percent which defines the Y-aligned distance between shapes.
float **GetPitchY**()	Returns the current ***Pitch Y*** in percent.
void **SetPitchZ**(float value)	Sets the ***Pitch Z*** *value* in percent which defines the Z-aligned distance between shapes.
float **GetPitchZ**()	Returns the current ***Pitch Z*** in percent.
void **SetPitch**(float value, float value, float value)	Sets the ***Pitch X, Y, Z*** *values* in percent which define the distances between shapes.
void **SetPixelPitchX**(int value)	Sets the ***Pitch X*** *value* in pixels, which defines the X-aligned distance between shapes.
int **GetPixelPitchX**()	Returns the current ***Pitch X*** in pixels.
void **SetPixelPitchY**(int value)	Sets the ***Pitch Y*** *value* in pixels, which defines the Y-aligned distance between shapes.
int **GetPixelPitchY**()	Returns the current ***Pitch Y*** in pixels.
void **SetPixelPitchZ**(int value)	Sets the ***Pitch Z*** pixel in pixels, which defines the Z-aligned distance between shapes.
int **GetPixelPitchZ**()	Returns the current ***Pitch Z*** in pixels.
void **SetPixelPitch**(int value, int value, int value)	Sets the ***Pitch X, Y, Z*** *values* in pixels which define the distances between shapes.

5.2.13 SCE Fill Snake

Functions Provided By SCE Fill Snake

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »**Using Color Table**
- This Effect uses Shapes. Learn more »**Using Shapes**
- This Effect uses the Size Control. Learn more »**Using Size Control**

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.

float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetColorMix**(int value)	Use *value* 1 (**TRUE**) to activate the ***Color Mix***. When ***Color Mix*** is activated, every shape has its own color depending on the »**Color Table Mode.** Use *value* 0 (**FALSE**) to use the default setting.
int **GetColorMix**()	Returns 1 (**TRUE**) if the ***Color Mix*** is enabled, otherwise 0 (**FALSE**).
void **ToggleColorMix**()	Enables or disables the ***Color Mix*** depending on the current state.
void **SetFillTime**(float value)	Sets the ***Fill Time*** *value* which defines how long is takes to fill the whole matrix with shapes.
float **GetFillTime**()	Returns the current ***Fill Time***.
void **SetDuration**(float value)	Sets the ***Duration*** *value* which defines how long the completely filled matrix is displayed.
float **GetDuration**()	Returns the current ***Duration***.
void **SetPitchX**(float value)	Sets the ***Pitch X*** *value* in percent which defines the X-aligned distance between shapes.
float **GetPitchX**()	Returns the current ***Pitch X*** in percent.
void **SetPitchY**(float value)	Sets the ***Pitch Y*** *value* in percent which defines the Y-aligned distance between shapes.
float **GetPitchY**()	Returns the current ***Pitch Y*** in percent.
void **SetPitchZ**(float value)	Sets the ***Pitch Z*** *value* in percent which defines the Z-aligned distance between shapes.
float **GetPitchZ**()	Returns the current ***Pitch Z*** in percent.
void **SetPitch**(float value, float value, float value)	Sets the ***Pitch X, Y, Z*** *values* in percent which define the distances between shapes.
void **SetPixelPitchX**(int value)	Sets the ***Pitch X*** *value* in pixels, which defines the X-aligned distance between shapes.
int **GetPixelPitchX**()	Returns the current ***Pitch X*** in pixels.
void **SetPixelPitchY**(int value)	Sets the ***Pitch Y*** *value* in pixels, which defines the Y-aligned distance between shapes.
int **GetPixelPitchY**()	Returns the current ***Pitch Y*** in pixels.
void **SetPixelPitchZ**(int value)	Sets the ***Pitch Z*** pixel in pixels, which defines the Z-aligned distance between shapes.

int **GetPixelPitchZ**()	Returns the current ***Pitch Z*** in pixels.
void **SetPixelPitch**(int value, int value, int value)	Sets the ***Pitch X, Y, Z*** *values* in pixels which define the distances between shapes.
void **SetStartCorner**(int mode)	Sets the ***Start Corner***. See **below** for a list of constants.
int **GetStartCorner**()	Returns the ***Start Corner***. See **below** for a list of constants.
void **SetOrientation**(int mode)	Sets the ***Orientation Mode***. See **below** for a list of constants.
int **GetOrientation**()	Returns the ***Orientation Mode***. See **below** for a list of constants.
void **SetCircle**(int enable)	Use *value* 1 (**TRUE**) to activate the ***Circle Mode***. Use *value* 0 (**FALSE**) to use the default setting.
int **GetCircle**()	Returns 1 (**TRUE**) if the ***Circle Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleCircle**()	Enables or disables the ***Circle Mode*** depending on the current state.
void **SetCenter**(int enable)	Use *value* 1 (**TRUE**) to activate the ***Center Mode***. Use *value* 0 (**FALSE**) to use the default setting.
int **GetCenter**()	Returns 1 (**TRUE**) if the ***Center Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleCenter**()	Enables or disables the ***Center Mode*** depending on the current state.
void **SetMirror**(int enable)	Use *value* 1 (**TRUE**) to activate the ***Mirror Mode***. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Enables or disables the ***Mirror Mode*** depending on the current state.

Start Corner Constants

Constant	Description
int START_CORNER_FRONT_TOP_LEFT	Sets the Start Corner to ***Front Top Left***.
int START_CORNER_FRONT_TOP_RIGHT	Sets the Start Corner to ***Front Top Right***.
int START_CORNER_FRONT_BOTTOM_LEFT	Sets the Start Corner to ***Front Bottom Left***.
int START_CORNER_FRONT_BOTTOM_RIGHT	Sets the Start Corner to ***Front Bottom Left***.
int START_CORNER_BACK_TOP_LEFT	Sets the Start Corner to ***Back Top Left***.
int START_CORNER_BACK_TOP_RIGHT	Sets the Start Corner to ***Back Top Right***.
int START_CORNER_BACK_BOTTOM_LEFT	Sets the Start Corner to ***Back Bottom Left***.
int START_CORNER_BACK_BOTTOM_RIGHT	Sets the Start Corner to ***Back Bottom Right***.

Orientation Mode Constants

Constant	Description
int ORIENTATION_XYZ	Sets the Orientation Mode to ***XYZ Orientation***.
int ORIENTATION_XZY	Sets the Orientation Mode to ***XZY Orientation***.
int ORIENTATION_YXZ	Sets the Orientation Mode to ***YXZ Orientation***.
int ORIENTATION_YZX	Sets the Orientation Mode to ***YZX Orientation***.
int ORIENTATION_ZYX	Sets the Orientation Mode to ***ZYX Orientation***.
int ORIENTATION_ZXY	Sets the Orientation Mode to ***ZXY Orientation***.

5.2.14 SCE Fill Solid

Functions Provided By SCE Fill Solid

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Look-At Types. Learn more »Using Look-At Types

Function	Description
void **SetStepWidth**(float value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using percentage values.
float **GetStepWidth**()	Returns the ***Step*** in percentage.
void **SetPixelStepWidth**(int value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using pixel values.
int **GetPixelStepWidth**()	Returns the ***Step*** in pixels.
void **SetColorWidth**(float value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using percentage values.
float **GetColorWidth**()	Returns the ***Color Width***, the width of a stripe, in percentage.
void **SetPixelColorWidth**(int value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using pixel values.
int **GetPixelColorWidth**()	Returns the ***Color Width***, the width of a stripe, in pixels.
void **SetCrossWidth**(float value)	Sets the ***Cross Width*** *value* using percentage values.
float **GetCrossWidth**()	Returns the ***Cross Width*** in percentage.
void **SetPixelCrossWidth**(int value)	Sets the ***Cross Width*** *value* using pixel values.
int **GetPixelCrossWidth**()	Returns the ***Cross Width*** in pixels.
void **SetCrossAxes**(int value)	Sets the ***Cross Axis*** *value*. See below for a list of constants.
int **GetCrossAxes**()	Returns the ***Cross Axis***. See below for a list of constants.
void **SetColorMix**(int value)	Use *value* 1 (**TRUE**) to activate the ***Color Mix***. When ***Color Mix*** is activated, stripes with distinct colors depending on the »Color Table Mode are generated. Use *value* 0 (**FALSE**) to use the default setting.
int **GetColorMix**()	Returns 1 (**TRUE**) if the ***Color Mix*** is enabled, otherwise 0 (**FALSE**).
void **ToggleColorMix**()	Enables or disables the ***Color Mix*** depending on the current state.
void **SetFillTime**(float value)	Sets the ***Fill Time*** *value* which defines how long is takes to fill the whole matrix.
float **GetFillTime**()	Returns the current ***Fill Time***.
void **SetDuration**(float value)	Sets the ***Duration*** *value* which defines how long the completely filled matrix is displayed.

float **GetDuration**()	Returns the current ***Duration***.

Cross Axis Constants

Constant	Description
int `AXIS_ONE`	Sets the direction cross mode to ***Axis 1***.
int `AXIS_TWO`	Sets the direction cross mode to ***Axis 2***.
int `AXES_ONE_AND_TWO`	Sets the direction cross mode to ***Axes 1 And 2***.

5.2.15 SCE Fire

Functions Provided By SCE Fire

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Gradient. Learn more »Using Color Gradient
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetHeight**(int height)	Sets the ***Height*** of the flames.
int **GetHeight**()	Returns the current ***Height***.

5.2.16 SCE Flames

Functions Provided By SCE Flames

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetHeight**(int height)	Sets the ***Height*** of the flames.
int **GetHeight**()	Returns the current ***Height***.
void **SetWidth**(int width)	Sets the ***Width*** of the flames.
int **GetWidth**()	Returns the ***Width*** of the flames.
void **SetDepth**(int depth)	Sets the ***Depth*** of the flames.
int **GetDepth**()	Returns the ***Depth*** of the flames.
void **SetIntensity**(int intensity)	Sets the ***Intensity*** of the flames.
int **GetIntensity**()	Returns the currently used ***Intensity*** for the flames.

5.2.17 SCE Fluid

Functions Provided By SCE Fluid

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetRadius**(int radius)	Sets the ***Radius*** for every new impulse.

int **GetRadius**()	Returns the ***Radius*** for every new impulse.
void **SetVorticity**(int vorticity)	Sets the ***Vorticity*** for the effect.
int **GetVorticity**()	Returns the ***Vorticity***.
void **SetDensity**(int density)	Sets the ***Density*** for every new impulse.
float **GetDensity**()	Returns the ***Density*** for every new impulse.
void **SetVelocity**(int velocity)	Sets the ***Velocity*** for every new impulse.
int **GetVelocity**()	Returns the ***Velocity*** for every new impulse.
void **SetDuration**(int duration)	Sets the ***Duration*** for every new impulse.
int **GetDuration**()	Returns the ***Duration*** for every new impulse.
void **SetFade**(int fade)	Sets the ***Fade*** for the effect.
int **GetFade**()	Returns the ***Fade*** value.
void **SetDeceleration**(int deceleration)	Sets the ***Deceleration*** for the effect.
int **GetDeceleration**()	Returns the ***Deceleration*** value.
void **SetColorMode**(int mode)	Sets the ***Color Mode*** for the effect. See **below** for details.
int **GetColorMode**()	Returns which ***Color Mode*** is set.

Color Mode Constants

This effect uses various color modes. The function **SetColorMode** can be used to change them. The following values can be used as parameter:

Constant	Description
int `MODE_COLOR_TABLE`	Sets the ***Color Table*** mode.
int `MODE_GRADIENT`	Sets the ***Color Gradient*** mode.

5.2.18 SCE Gradient

Functions Provided By SCE Gradient

This effect uses the following functions:

- This Effect uses Directions. Learn more »**Using Directions**
- This Effect uses the Color Gradient. Learn more »**Using Color Gradient**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**

Function	Description
void **SetAngle**(int value)	Sets the ***Angle*** *value* in degrees.
float **GetAngle**()	Returns the ***Angle*** in degrees.
void **SetCrossWidth**(float value)	Sets the ***Cross Width*** *value* using percentage values.
float **GetCrossWidth**()	Returns the ***Cross Width*** in percentage.
void **SetPixelCrossWidth**(int value)	Sets the ***Cross Width*** *value* using pixel values.
int **GetPixelCrossWidth**()	Returns the ***Cross Width*** in pixels.
void **SetCrossAxes**(int value)	Sets the ***Cross Axis*** *value*. See below for a list of constants.
int **GetCrossAxes**()	Returns the ***Cross Axis***. See below for a list of constants.

Function	Description
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.

Cross Axis Constants

Constant	Description
int `AXIS_ONE`	Sets the direction cross mode to ***Axis 1***.
int `AXIS_TWO`	Sets the direction cross mode to ***Axis 2***.
int `AXES_ONE_AND_TWO`	Sets the direction cross mode to ***Axes 1 And 2***.

Example

This macro sets the direction to `DIRECTION_RADIAL` when the effect is started. Before the effect is rendered, the angle will be increased by 1 until 360 degrees is achieved. Then, it starts again with 0 degrees. As a result, the effect looks like a rotating radial gradient.

```
@scriptname="Radial Gradient - Rotated";
@author="inoage GmbH";
```

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

int degree;

void InitEffect()
{
 SetDirection(DIRECTION_RADIAL);
 degree = 0;
}

void PreRenderEffect()
{
 degree = (degree+1)%360;
 SetAngle(degree);
}

void PostRenderEffect()
{

}

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

5.2.19 SCE Graph

Functions Provided By SCE Graph

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Directions. Learn more »Using Directions
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses Text. Learn more »Using Text

Function	Description
Height	
void **SetHeight**(float value)	Sets the ***Height*** *value*, which is the height of the shapes in percent of the matrix size (depending on the look-at type). The Height *value* means the minimum shape height if the shape height distribution is not uniform. Valid values for *value* range from *0.01* to *100*.

float **GetHeight**()	Returns the current ***Height*** of the shapes in percent. The return value means the minimum shape height if the shape height distribution is not uniform.
void **SetPixelHeight**(int value)	Sets the ***Height*** *value*, which is the height of the shapes, in pixels. The Height *value* means the minimum shape height if the shape height distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the look-at type).
int **GetPixelHeight**()	Returns the current ***Height*** of the shapes in pixels. The return value means the minimum shape height if the shape height distribution is not uniform.
void **SetHeightMax**(float value)	Sets the ***Maximum Height*** *value*, which is the maximum height of the shapes, in percent of the matrix size (depending on the look-at type). The Maximum Height *value* only applies if the shape height distribution is not uniform. Valid values for *value* range from *0.01* to *100*.
float **GetHeightMax**()	Returns the current ***Maximum Height*** of the shapes in percent.
void **SetPixelHeightMax**(int value)	Sets the ***Maximum Height*** *value*, which is the maximum height of the shapes, in pixels. The maximum height *value* only applies if the shape height distribution is not uniform. Valid values for *value* range from 1 to the matrix size (depending on the look-at type).
int **GetPixelHeightMax**()	Returns the current ***Maximum Height*** of the shapes in pixels.
void **SetHeightDistribution**(int value)	Sets the ***Height Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape height is applied to the shapes. See »**here** for a list of constants.
int **GetHeightDistribution**()	Returns the current ***Height Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomHeight**()	Randomizes the heights of the shapes. Same as **SetHeightDistribution**(**DIST_RND**).
Width/Depth	
void **SetWidth**(float value, int register)	Sets the ***Width*** *value* for register *reg*, which is the width of the shapes, in percent of the matrix size (depending on the look-at type). The Width *value* means the minimum shape width if the shape width distribution is not uniform. Valid values for *value* range from *0.01* to *100* . Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The width *value* for register **REGISTER_Z1** actually controls the depth of the shapes.
float **GetWidth**(int register)	Returns the current ***Width*** of the shapes for register *reg* in percent. The return value means the minimum shape width if the shape width distribution is not uniform. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The return value for register **REGISTER_Z1** actually represents the depth of the shapes.
void **SetPixelWidth**(int value, int register)	Sets the ***Width*** *value* for register *reg*, which is the width of the shapes, in pixels. The Width *value* means the minimum shape width if the shape width distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the look-at type). Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The width *value* for register **REGISTER_Z1** actually controls the depth of the shapes.
int **GetPixelWidth**(int reg)	Returns the current ***Width*** of the shapes for register *reg* in pixels. The return value means the minimum shape width if the shape width distribution is not uniform. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The return value for register **REGISTER_Z1** actually represents the depth of the shapes.

void **SetWidthMax**(float value, int register)	Sets the ***Maximum Width*** *value* for register *reg*, which is the maximum width of the shapes, in percent of the matrix size (depending on the look-at type). The Maximum Width *value* only applies if the shape width distribution is not uniform. Valid values for *value* range from *0.01* to *100*. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The maximum width *value* for register **REGISTER_Z1** actually controls the maximum depth of the shapes.
float **GetWidthMax**(int register)	Returns the current ***Maximum Width*** of the shapes for register *reg* in percent. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The return value for register **REGISTER_Z1** actually represents the maximum depth of the shapes.
void **SetPixelWidthMax**(int value, int register)	Sets the ***Maximum Width*** *value* for register *reg*, which is the maximum width of the shapes, in pixels. The maximum width *value* only applies if the shape width distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the look-at type). Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The maximum width *value* for register **REGISTER_Z1** actually controls the maximum depth of the shapes.
int **GetPixelWidthMax**(int register)	Returns the current ***Maximum Width*** of the shapes for register *reg* in pixels. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The return value for register **REGISTER_Z1** actually represents the maximum depth of the shapes.
void **SetWidthDistribution**(int value, int register)	Sets the width ***Distribution*** *value* for register *reg*, which defines how the range between the current minimum and the current maximum shape width is applied to the shapes. See »here for a list of constants. Valid values for *reg* are **REGISTER_X1** and **REGISTER_Z1**. The width distribution *value* for register **REGISTER_Z1** actually controls the depth distribution of the shapes.
int **GetWidthDistribution**(int register)	Returns the current width ***Distribution*** of the shapes for register *reg*. See »here for a list of constants. Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**. The return value for register **REGISTER_Z1** actually represents the depth distribution of the shapes.
void **SeedRandomWidth**(int register)	Randomizes the widths of the shapes for register *reg*. Is the same as **SetWidthDistribution**(**DIST_RND**, reg). Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**.
Pitch	
void **SetPitch**(float value, int register)	Sets the ***Pitch*** *value* for register *reg*, which is the distance of the shapes, in percent of the matrix size (depending on the look-at type). The pitch *value* means the minimum pitch if the pitch distribution is not uniform. Valid values for *value* range from *0.01* to *100*. Valid values for *reg* are **REGISTER_X1**, for which the Pitch *value* controls the X-distance, and **REGISTER_Z1** for the Z-distance.
float **GetPitch**(int register)	Returns the current ***Pitch*** of the shapes for register *reg* in percent. The return value means the minimum pitch if the pitch distribution is not uniform. Valid values for *reg* are **REGISTER_X1**, for which the return value represents the X-distance, and **REGISTER_Z1** for the Z-distance.

void **SetPixelPitch**(int value, int register)	Sets the ***Pitch*** *value* for register *reg*, which is the distance of the shapes, in pixels. The pitch *value* means the minimum pitch if the pitch distribution is not uniform. Valid values for *value* range from 1 to the matrix size (depending on the look-at type). Valid values for *reg* are **REGISTER_X1**, for which the Pitch *value* controls the X-distance, and **REGISTER_Z1** for the Z-distance.
int **GetPixelPitch**(int register)	Returns the current ***Pitch*** of the shapes for register *reg* in pixels. The return value means the minimum pitch if the pitch distribution is not uniform. Valid values for *reg* are **REGISTER_X1**, for which the return value represents the X-distance, and **REGISTER_Z1** for the Z-distance.
void **SetPitchMax**(float value, int register)	Sets the ***Maximum Pitch*** *value* for register *reg*, which is the maximum distance of the shapes, in percent of the matrix size (depending on the look-at type). The maximum pitch *value* only applies if the pitch distribution is not uniform. Valid values for *value* range from *0.01* to *100* . Valid values for *reg* are **REGISTER_X1**, for which the maximum pitch *value* controls the maximum X-distance, and **REGISTER_Z1** for the maximum Z-distance.
float **GetPitchMax**(int register)	Returns the current ***Maximum Pitch*** of the shapes for register *reg* in percent. Valid values for *reg* are **REGISTER_X1**, for which the return value represents the maximum X-distance, and **REGISTER_Z1** for the maximum Z-distance.
void **SetPixelPitchMax**(int value, int register)	Sets the ***Maximum Pitch*** *value* for register *reg*, which is the maximum distance of the shapes, in pixels. The maximum pitch *value* only applies if the pitch distribution is not uniform. Valid values for *value* range from *1* to the matrix size (depending on the look-at type). Valid values for *reg* are **REGISTER_X1**, for which the maximum pitch *value* controls the maximum X-distance, and **REGISTER_Z1** for the maximum Z-distance.
int **GetPixelPitchMax**(int register)	Returns the current ***Maximum Pitch*** of the shapes for register *reg* in pixels. Valid values for *reg* are **REGISTER_X1**, for which the return value represents the maximum X-distance, and **REGISTER_Z1** for the maximum Z-distance.
void **SetPitchDistribution**(int value, int register)	Sets the ***Pitch Distribution*** *value* for register *reg*, which defines how the range between the current minimum and the current maximum pitch is applied to the shapes. See »**here** for a list of constants. Valid values for *reg* are **REGISTER_X1**, for which the pitch distribution *value* controls the X-distance distribution, and **REGISTER_Z1** for the Z-distance distribution.
int **GetPitchDistribution**(int register)	Returns the current ***Pitch Distribution*** of the shapes for register *reg*. See »**here** for a list of constants. Valid values for *reg* are **REGISTER_X1**, for which the return value represents the X-distance distribution, and **REGISTER_Z1** for the Z-distance distribution.
void **SeedRandomPitch**(int register)	Randomizes the pitches of the shapes for register *reg*. Is the same as **SetPitchDistribution**(**DIST_RND**, reg). Valid values for *reg* are **REGISTER_X1** or **REGISTER_Z1**.
Functional Parameters	
void **SetFrequency**(float value, int register)	Sets the ***Frequency*** *value* for register *reg*, which defines the number of periods of the graph function. The frequency *value* means the minimum frequency if the frequency distribution is not uniform. Valid values for *value* range from *0* to *100*. See **below** for a list of register constants.

float **GetFrequency**(int register)	Returns the current ***Frequency*** for register *reg*. The return value means the minimum frequency if the frequency distribution is not uniform. See **below** for a list of register constants.
void **SetFrequencyMax**(float value, int register)	Sets the ***Maximum Frequency*** *value* for register *reg*, which defines the maximum number of periods of the graph function. The maximum frequency *value* only applies if the frequency distribution is not uniform. Valid values for *value* range from *0* to *100*. See **below** for a list of register constants.
float **GetFrequencyMax**(int register)	Returns the current ***Maximum Frequency*** for register *reg*. See **below** for a list of register constants.
void **SetFrequencyDistribution**(int value, int register)	Sets the ***Frequency Distribution*** *value* for register *reg*, which defines how the range between the current minimum and the current maximum frequency is applied to the shapes. See »**here** for a list of constants. See **below** for a list of register constants.
int **GetFrequencyDistribution**(int register)	Returns the current ***Frequency Distribution*** for register *reg*. See » **here** for a list of constants. See **below** for a list of register constants.
void **SeedRandomFrequency**(int register)	Randomizes the frequencies for register *reg*. Same as **SetFrequencyDistribution**(**DIST_RND**, reg). See **below** for a list of register constants.
void **SetAmplitude**(float value, int register)	Sets the ***Amplitude*** *value* for register *reg*, which defines how wide the shapes oscillate, in percent of the matrix size (depending on the look-at type). The amplitude *value* means the minimum amplitude if the amplitude distribution is not uniform. Valid values for *value* range from *-1000* to *1000*. See **below** for a list of register constants.
float **GetAmplitude**(int register)	Returns the current ***Amplitude*** for register *reg* in percent. The return value means the minimum amplitude if the amplitude distribution is not uniform. See **below** for a list of register constants.
void **SetPixelAmplitude**(int value, int register)	Sets the ***Amplitude*** *value* for register *reg*, which defines how wide the shapes oscillate, in pixels. The amplitude *value* means the minimum amplitude if the amplitude distribution is not uniform. Valid values for *value* range between - and + ten times the matrix size (depending on the look-at type). See **below** for a list of register constants.
int **GetPixelAmplitude**(int register)	Returns the current ***Amplitude*** for register *reg* in pixels. The return value means the minimum amplitude if the amplitude distribution is not uniform. See **below** for a list of register constants.
void **SetAmplitudeMax**(float value, int register)	Sets the ***Maximum Amplitude*** *value* for register *reg*, which defines how wide the shapes oscillate, in percent of the matrix size (depending on the look-at type). The maximum amplitude *value* only applies if the amplitude distribution is not uniform. Valid values for *value* range from *-1000* to *1000*. See **below** for a list of register constants.
float **GetAmplitudeMax**(int register)	Returns the current ***Maximum Amplitude*** for register *reg* in percent. See **below** for a list of register constants.
void **SetPixelAmplitudeMax**(int value, int register)	Sets the ***Maximum Amplitude*** *value* for register *reg*, which defines how wide the shapes oscillate, in pixels. The maximum amplitude *value* only applies if the amplitude distribution is not uniform. Valid values for *value* range between - and + ten times the matrix size (depending on the look-at type). See **below** for a list of register constants.
int **GetPixelAmplitudeMax**(int register)	Returns the current ***Maximum Amplitude*** for register *reg* in pixels. See **below** for a list of register constants.
void **SetAmplitudeDistribution**(int value, int register)	Sets the ***Amplitude Distribution*** *value* for register *reg*, which defines how the range between the current minimum and the current maximum amplitude is applied to the shapes. See »**here** for a list of constants. See **below** for a list of register constants.

int **GetAmplitudeDistribution**(int register)	Returns the current ***Amplitude Distribution*** for register *reg*. See » here for a list of constants. See below for a list of register constants.
void **SeedRandomAmplitude**(int register)	Randomizes the amplitudes for register *reg*. Is the same as **SetAmplitudeDistribution**(**DIST_RND**, reg). See below for a list of register constants.
void **SetPhase**(float value, int register)	Sets the ***Phase*** *value* for register *reg*, which defines an offset for the graph function, in percent of the wavelength. Valid values for *value* range from *0* to *100*. See below for a list of register constants.
float **GetPhase**(int register)	Returns the current ***Phase*** for register *reg* in percent. See below for a list of register constants.
void **SetPeak**(float value, int register)	Sets the ***Peak*** *value* for register *reg*, which defines a threshold for the graph function, in percent of the wavelength. This only applies to the Triangle and the Square function type. Valid values for *value* range from 0 to 100. See below for a list of register constants.
float **GetPeak**(int register)	Returns the current ***Peak*** for register *reg* in percent. See below for a list of register constants.
void **SetFunctionType**(int value, int register)	Sets the ***Function Type*** *value* for register *reg*, which defines the base function for the graph. See »here for a list of constants. See below for a list of register constants.
int **GetFunctionType**(int register)	Returns the current ***Function Type*** for register *reg*. See »here for a list of constants. See below for a list of register constants.
Others	
void **ClearRegister**(int register)	Sets all functional parameters for register *reg* to their defaults. See below for a list of register constants.
Function	**Description**
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.

Register Constants

Constant	Description
int **REGISTER_X1**	Selects register 1 for X-Axis.
int **REGISTER_X2**	Selects register 2 for X-Axis.
int **REGISTER_X3**	Selects register 3 for X-Axis.
int **REGISTER_Z1**	Selects register 1 for Z-Axis.
int **REGISTER_Z2**	Selects register 2 for Z-Axis.
int **REGISTER_Z3**	Selects register 3 for Z-Axis.
int **REGISTER_P1**	Selects register 1 for extruded phase.
int **REGISTER_P2**	Selects register 2 for extruded phase.
int **REGISTER_P3**	Selects register 3 for extruded phase.

MADRIX 3.X To MADRIX 5.X Migration Hints

The following functions are not supported anymore. Please follow the hints to migrate your macros.

Function	Description
void **EditableTextSetTextRotationMode**(int value)	Use **SetShapeRotation**(int value) instead.
int **EditableTextGetTextRotationMode**()	Use **GetShapeRotation**() instead.

5.2.20 SCE Image

Functions Provided By SCE Image

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the Position Control. Learn more »Using Position Control
- This Effect uses the Image List. Learn more »Using Image Table

Function	Description
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.
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 *angle* are *0*, *90*, *180*, and *270*.
int **GetRotation**()	Returns the currently set ***Rotation***.
void **SetTiling**(int enable)	Disables ***Tile Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Tile Mode*** will be enabled via 1 (**TRUE**).
int **GetTiling**()	Returns 1 (**TRUE**) if ***Tile Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleTiling**()	Enables ***Tile Mode*** or uses the default setting, depending on the current state.
void **SetSeamless**(int enable)	Disables ***Seamless*** if *enable* is 0 (**FALSE**), otherwise it will be enabled.
int **GetSeamless**()	Returns 1 (**TRUE**) if ***Seamless*** is enabled, otherwise 0 (**FALSE**).
void **ToggleSeamless**()	Enables ***Seamless*** or uses the default setting, depending on the current state.
float **GetImageWidth**()	Returns the image width of the current image in percent of the matrix width.
float **GetImageHeight**()	Returns the image height of the current image in percent of the matrix height.
float **GetImageDepth**()	Returns the image depth of the current image in percent of the matrix depth.
int **GetImagePixelWidth**()	Returns the image width of the current image in pixels.

int **GetImagePixelHeight**()	Returns the image height of the current image in pixels.
int **GetImagePixelDepth**()	Returns the image depth of the current imagein pixel.
float **GetCurrentPositionX**()	Returns the X-position of the current image in percent of the matrix width.
float **GetCurrentPositionY**()	Returns the Y-position of the current image in percent of the matrix height.
float **GetCurrentPositionZ**()	Returns the Z-position of the current image in percent of the matrix depth.
int **GetCurrentPixelPositionX**()	Returns the X-position of the current image in pixels.
int **GetCurrentPixelPositionY**()	Returns the Y-position of the current image in pixels.
int **GetCurrentPixelPositionZ**()	Returns the Z-position of the current image in pixels.
void **SetPlaybackRate**(float value)	Sets the ***Playback Rate*** of the image list. E.g., a *value* of 2.0 means that the image list will be running 2x faster than the original speed.
float **GetPlaybackRate**()	Returns the current ***Playback Rate*** of the image list.
void **SetCurrentImage**(int index)	Sets the specified image, using the *index*, as the currently displayed image.
int **GetCurrentImage**()	Return the currently displayed image.
void **SetStretchMode**(int mode)	Returns the ***Stretch*** mode. See **below** for details.
int **GetStretchMode**()	Returns the ***Stretch*** mode. See **below** for details.
void **SetGrayscale**(int enable)	Disables ***Grayscale*** if *enable* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**).
int **GetGrayscale**()	Returns 1 (**TRUE**) if ***Grayscale*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleGrayscale**()	Enables ***Grayscale*** mode or uses the default setting, depending on the current state.
void **SetRgbw**(int enable)	Disables the ***RGB-To-RGBW*** mode if *enable* is 0 (**FALSE**). Otherwise it can be enabled using **TRUE**.
int **GetRgbw**()	Returns 1 (**TRUE**) if ***RGB-To-RGBW*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleRgbw**()	Enables ***RGB-To-RGBW*** mode or uses the default setting, depending on the current state.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See **below** for details.
int **GetFilteringMode**()	Returns the ***Filtering*** mode. See **below** for details.

Stretch Mode Constants

Value	Description
int **STRETCH_MODE_NONE**	Sets the stretch mode to ***None*** (no stretching).

int `STRETCH_MODE_MATRIX`	Sets the stretch mode to ***Matrix*** (stretches to the current aspect ratio of the matrix).
int `STRETCH_MODE_ORIGINAL`	Sets the stretch mode to ***Original*** (stretches to the original aspect ratio of the source).
int `STRETCH_MODE_4_TO_3`	Sets the stretch mode to ***4:3***.
int `STRETCH_MODE_16_TO_9`	Sets the stretch mode to ***16:9***.

Filtering Mode Constants

Value	Description
int `FILTERING_MODE_NEAREST_NEIGHBOR`	Sets the Filtering Mode to ***None*** (no filtering).
int `FILTERING_MODE_LINEAR`	Sets the Filtering Mode to ***Linear*** (may require additional performance).

5.2.21 SCE Level Color Simulator

Functions Provided By SCE Level Color Simulator

This effect uses the following functions:

- This Effect uses the Gradient Dialog. Learn more »**Using Gradient Dialog**
- This Effect uses Directions. Learn more »**Using Directions**

Function	Description
void **SetFadeIn**(int value)	Sets the ***Fade In*** *value*. Hint: This value needs to be higher than the Fade Out value or the same.
int **GetFadeIn**()	Returns the ***Fade In*** *value*.
void **SetFadeOut**(int value)	Sets the ***Fade Out*** value. Hint: This value needs to be lower than the Fade In value or the same.
int **GetFadeOut**()	Returns the ***Fade Out*** value.
void **SetMode**(int value)	Sets the ***Mode***. See below for a list of constants.
int **GetMode**()	Returns the current ***Generator Type***. See below for a list of constants.

Mode Constants

Constant	Description
int `MODE_MONO`	Sets the mode to ***Mono***.
int `MODE_STEREO_LINEAR`	Sets the mode to ***Stereo Linear***.
int `MODE_STEREO_RADIAL`	Sets the mode to ***Stereo Radial***.

5.2.22 SCE Metaballs

Functions Provided By SCE Metaballs

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetCount**(int value)	Sets the ***Count*** of shapes.
int **GetCount**()	Retrieves the current ***Count*** of shapes.

void **SetSize**(int value)	Sets the ***Size*** *value*, which is the size of the shapes in percent of the matrix size (depending on the look-at type). The ***Size*** *value* means the minimum shape size if the shape size distribution mode is not uniform.
int **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMin**(int value)	Is the same as **SetSize.**
int **GetSizeMin**()	Is the same as **GetSize.**
void **SetSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes in percent of the matrix size (depending on the look-at type). The ***Maximum Size*** *value* only applies if the shape size distribution mode is not uniform.
int **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetSizeScale**(float value)	Sets the ***Size Scale*** factor of the shapes. This feature is only supported when the size distribution mode is **DIST_RND**.
float **GetSizeScale**()	Retrieves the current ***Size Scale*** factor.
void **SetSizeDistribution**(int value)	Sets the ***Size Distribution*** *value* which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »here for a list of constants.
int **GetSizeDistribution**()	Returns the current ***Size Distribution*** of the shapes. See »here for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Is the same as **SetSizeDistribution**(**DIST_RND**).
void **SetOuterGlow**(int value)	Sets the value for the ***Outer Glow*** of shapes. This feature is only supported by specific shape types.
int **GetOuterGlow**()	Returns the value of the ***Outer Glow*** of shapes.
void **SetInnerGlow**(int value)	Sets the value for the ***Inner Glow*** of shapes. This feature is only supported by specific shape types.
int **GetInnerGlow**()	Returns the value of the ***Inner Glow*** of shapes.
void **SetBorder**(int value)	Sets the ***Border*** size of shapes. This feature is only supported by specific shape types.
int **GetBorder**()	Returns the ***Border*** size of shapes.
void **SetColorMixingMode**(int value)	Sets the ***Color Mixing Mode***. This feature is only supported when the color table includes more then one color or when the color table is in random mode. Furthermore the color mixing link must be disabled. See below for a list of constants.
int **GetColorMixingMode**()	Returns the current ***Color Mixing Mode***.
void **SetColorMixingSharpness**(int value)	Sets the ***Color Mixing Sharpness***. This feature is only supported when the color table includes more then one color or when the color table is in random mode. See below for a list of constants.
int **GetColorMixingSharpness**()	Returns the current ***Color Mixing Sharpness***.
void **SetColorMixingLink**(int value)	Use *value* 1 (**TRUE**) to activate the ***Color Mixing Link***. When ***Color Mixing Link*** is activated, the auto adjustment for the color mixing mode is enabled. Use *value* 0 (**FALSE**) to use the default setting.
int **GetColorMixingLink**()	Returns 1 (**TRUE**) if the ***Color Mixing Link*** is enabled, otherwise 0 (**FALSE**).

void **ToggleColorMixingLink**()	Enables or disables the ***Color Mixing Link***, depending on the current state.
SetMotionType(int value)	Sets the ***Motion Type.*** See below for a list of constants.
int **GetMotionType**()	Returns the current ***Motion Type.***
void **SetInfluence**(float value)	Sets the ***Influence*** value. This feature is only supported by the motion type **MOTION_TYPE_SWARM**.
float **GetInfluence**()	Returns the current ***Influence*** value.
void **SetSeparation**(float value)	Sets the ***Separation*** value. This feature is only supported by the motion type **MOTION_TYPE_SWARM**.
float **GetSeparation**()	Returns the current ***Separation*** value.

Color Mixing Mode Constants

Constant	Description
int **COLOR_MIXING_MODE_CIRCLE**	Sets the color mixing mode to ***Circle***.
int **COLOR_MIXING_MODE_SQUARE**	Sets the color mixing mode to ***Square***.
int **COLOR_MIXING_MODE_DIAMOND**	Sets the color mixing mode to ***Diamond***.

Color Mixing Sharpness Constants

Constant	Description
int **COLOR_MIXING_SHARPNESS_VERY_BLURRY**	Sets the color mixing sharpness to ***Very Blurry***.
int **COLOR_MIXING_SHARPNESS_BLURRY**	Sets the color mixing sharpness to ***Blurry***.
int **COLOR_MIXING_SHARPNESS_SLIGHTLY_BLURRY**	Sets the color mixing sharpness to ***Slightly Blurry***.
int **COLOR_MIXING_SHARPNESS_MEDIUM**	Sets the color mixing sharpness to ***Medium***.
int **COLOR_MIXING_SHARPNESS_SLIGHTLY_CLEAR**	Sets the color mixing sharpness to ***Slightly Clear***.

int COLOR_MIXING_SHARPNESS_CLEAR	Sets the color mixing sharpness to ***Clear***.
int COLOR_MIXING_SHARPNESS_VERY_CLEAR	Sets the color mixing sharpness to ***Very Clear***.

Motion Type Constants

Constant	Description
int MOTION_TYPE_RANDOM	Sets the motion type to ***Random***.
int MOTION_TYPE_SWARM	Sets the motion type to ***Swarm***.

5.2.23 SCE Noise

Functions Provided By SCE Noise

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.

void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetFrequency**(float value)	Sets the ***Frequency*** for the effect.
float **GetFrequency**()	Returns the current ***Frequency*** value.
void **SetPitchX**(float value)	Sets the ***Pitch X*** value in percent which defines the X-aligned distance between objects.
float **GetPitchX**()	Returns the current ***Pitch X*** in percent.
void **SetPitchY**(float value)	Sets the ***Pitch Y*** value in percent which defines the Y-aligned distance between objects.
float **GetPitchY**()	Returns the current ***Pitch Y*** in percent.
void **SetPitchZ**(float value)	Sets the ***Pitch Z*** value in percent which defines the Z-aligned distance between objects.
float **GetPitchZ**()	Returns the current ***Pitch Z*** in percent.
void **SetPitch**(float value, float value, float value)	Sets the ***Pitch X, Y, Z*** values in percent which define the distances between objects.
void **SetPixelPitchX**(int value)	Sets the ***Pitch X*** value in pixels, which defines the X-aligned distance between objects.
int **GetPixelPitchX**()	Returns the current ***Pitch X*** in pixels.
void **SetPixelPitchY**(int value)	Sets the ***Pitch Y*** value in pixels, which defines the Y-aligned distance between objects.
int **GetPixelPitchY**()	Returns the current ***Pitch Y*** in pixels.
void **SetPixelPitchZ**(int value)	Sets the ***Pitch Z*** pixel in pixels, which defines the Z-aligned distance between objects.
int **GetPixelPitchZ**()	Returns the current ***Pitch Z*** in pixels.
void **SetPixelPitch**(int value, int value, int value)	Sets the ***Pitch X, Y, Z*** values in pixels which define the distances between objects.
void **SetVariace1**(int value)	Sets the ***Minimum Variance*** for the effect.
int **GetVariance1**()	Returns the current ***Minimum Variance*** value.
void **SetVariance2**(int value)	Sets the ***Maximum Variance*** for the effect.
int **GetVariance2**()	Returns the current ***Maximum Variance*** value.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the objects' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **Generate**()	Creates the effect anew.
void **SetMinimumSize**(int value)	Sets the ***Minimum Size*** for objects.
int **GetMinimumSize**()	Returns the current ***Minimum Size*** value.

void **SetRecolor**(int value)	Sets the ***Recolor*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetRecolor**()	Returns 1 (**TRUE**) if ***Recolor*** is enabled, otherwise 0 (**FALSE**).
void **ToggleRecolor**()	Toggles ***Recolor***.
void **SetReshape**(int value)	Sets the ***Reshape*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetReshape**()	Returns 1 (**TRUE**) if ***Reshape*** is enabled, otherwise 0 (**FALSE**).
void **ToggleReshape**()	Toggles ***Reshape***.
void **SetResize**(int value)	Sets the ***Resize*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetResize**()	Returns 1 (**TRUE**) if ***Resize*** is enabled, otherwise 0 (**FALSE**).
void **ToggleResize**()	Toggles ***Resize***.
void **SetFade**(int value)	Sets the ***Fade*** option. Use **TRUE** to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetFade**()	Returns 1 (**TRUE**) if ***Fade*** is enabled, otherwise 0 (**FALSE**).
void **ToggleFade**()	Toggles ***Fade***.

5.2.24 SCE Plasma

Functions Provided By SCE Plasma

This effect uses the following functions:

- This Effect uses the Color Gradient. Learn more »**Using Color Gradient**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**

5.2.25 SCE Pulse / Stroboscope

Functions Provided By SCE Pulse / Stroboscope

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »**Using Color Table**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**

Function	Description
void **SetRatio**(float value)	Sets the ***Ratio*** *value*. A Ratio of 80% means an on-time of 0.2s and an off-time of 0.8s with 60 BPM (1Hz).
float **GetRatio**()	Returns the ***Ratio*** *value*.
void **SetFade**(int enable)	Sets the ***Fade*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetFade**()	Returns 1 (**TRUE**) if ***Fade*** is activated, otherwise 0 (**FALSE**).
void **ToggleFade**()	Toggles the ***Fade*** option.

5.2.26 SCE Rotating Shapes

Functions Provided By SCE Rotating Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »**Using Color Table**
- This Effect uses the Shape Table. Learn more »**Using Shape Table**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**
- This Effect uses the Center Control. Learn more »**Using Center Control**

Function	Description
void **SetPathAligned**(int value)	Use *value* 1 (**TRUE**) to activate the ***Path Alignment***. Use *value* 0 (**FALSE**) otherwise.
int **GetPathAligned**()	Returns 1 (**TRUE**) if the ***Path Alignment*** is activated, otherwise 0 (**FALSE**).
void **SetPathCount**(int value)	Sets the ***Path Count*** *value*, which defines how much concentric paths are displayed. Valid values for *value* range from *1* to *20*.
int **GetPathCount**()	Returns the current ***Path Count***.
void **SetMotion**(float value)	Sets the ***Motion*** *value*, which is the speed of the shape movement across the paths. Valid values for *value* range from *-100.0* to *100.0*.
float **GetMotion**()	Returns the current ***Motion***.
void **SetPathBoundMin**(float value)	Sets the ***Minimum Path Bounds*** *value*, which is the size of the inner path in percent of the matrix size (depending on the look-at type).
float **GetPathBoundMin**()	Returns the current ***Minimum Path Bounds*** in percent.
void **SetPathBoundMax**(float value)	Sets the ***Maximum Path Bounds*** *value*, which is the size of the outer path in percent of the matrix size (depending on the look-at type).

float **GetPathBoundMax**()	Returns the current ***Maximum Path Bounds*** in percent.
void **SetPathBoundDistribution**(int value)	Sets the ***Path Bounds Distribution*** *value* which defines how the range between the current minimum and the current maximum path bounds is applied to the paths. See »**here** for a list of constants.
int **GetPathBoundDistribution**()	Returns the current ***Path Bounds Distribution***. See »**here** for a list of constants.
void **SetShapeOnBounds**(int value)	Use *value* 1 (**TRUE**) to center the shapes on the inner and outer path positions. Use *value* 0 (**FALSE**) otherwise.
int **GetShapeOnBounds**()	Returns 1 (**TRUE**) if the shapes are centered on the inner and outer path positions, otherwise 0 (**FALSE**).
void **SetOffset**(float value)	Sets the ***Offset*** *value*, which is an additional rotation of the paths against each other. Valid values for *value* range from *-10000.0* to *10000.0*.
float **GetOffset**()	Returns the current ***Offset***.
void **SetOffsetAnimation**(int value)	Use *value* 1 (**TRUE**) to activate the ***Offset Animation***. Use *value* 0 (**FALSE**) otherwise.
int **GetOffsetAnimation**()	Returns 1 (**TRUE**) if the ***Offset Animation*** is activated, otherwise 0 (**FALSE**).
void **SetPathCrossWidth**(int value)	Sets the ***Path Cross Width*** *value*, which defines the oppositeness of the ***Offset*** of the paths. Valid values for *value* range from *0* to *20*.
int **GetPathCrossWidth**()	Returns the current ***Path Cross Width***.
void **SetShapeCount**(int value)	Sets the ***Shape Count*** *value*, which defines how much shapes are displayed on each path. The ***Shape Count*** *value* means the minimum shape count if the shape count distribution mode is not uniform.
int **GetShapeCount**()	Returns the current ***Shape Count***. The return value means the minimum shape count if the shape count distribution mode is not uniform.
void **SetShapeCountMin**(int value)	Is the same as **SetShapeCount**.
int **GetShapeCountMin**()	Is the same as **GetShapeCount**.
void **SetShapeCountMax**(int value)	Sets the ***Maximum Shape Count*** *value*, which defines how much shapes are displayed on each path at a maximum. The ***Maximum Shape Count*** *value* only applies if the shape count distribution mode is not uniform.
int **GetShapeCountMax**()	Returns the current ***Maximum Shape Count***.
void **SetShapeCountDistribution**(int value)	Sets the ***Shape Count Distribution*** *value* which defines how the range between the current minimum and the current maximum shape count is applied to the shapes. See »**here** for a list of constants.
int **GetShapeCountDistribution**()	Returns the current ***Shape Count Distribution***. See »**here** for a list of constants.
void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes in percent of the matrix size (depending on the look-at type). The ***Size*** *value* means the minimum shape size if the shape size distribution mode is not uniform.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMin**(float value)	Is the same as **SetSize.**
float **GetSizeMin**()	Is the same as **GetSize.**

void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes in percent of the matrix size (depending on the look-at type). The ***Maximum Size*** *value* only applies if the shape size distribution mode is not uniform.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetSizeDistribution**(int value)	Sets the ***Size Distribution*** *value* which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »here for a list of constants.
int **GetSizeDistribution**()	Returns the current ***Size Distribution*** of the shapes. See »here for a list of constants.
void **SetShapeRotation**(float value)	Sets the ***Rotation*** *value*, which is an additional rotation of the shapes. Valid values for *value* range from *-3600.0* to *3600.0*.
float **GetShapeRotation**()	Returns the current ***Rotation***.
void **SetShapeRotationAnimation**(int value)	Use *value* 1 (**TRUE**) to activate the ***Rotation Animation***. Use *value* 0 (**FALSE**) otherwise.
int **GetShapeRotationAnimation**()	Returns 1 (**TRUE**) if the ***Rotation Animation*** is activated, otherwise 0 (**FALSE**).
void **SetShapeRotationCrossWidth**(int value)	Sets the ***Rotation Cross Width*** *value*, which defines the oppositeness of the ***Rotation*** of the shapes. Valid values for *value* range from *0* to *50*.
int **GetShapeRotationCrossWidth**()	Returns the current ***Rotation Cross Width***.
void **SetSpeed**(float value)	Sets the ***Speed*** *value*, which is the animation speed of imploding, exploding and pulsing shapes. Valid values for *value* range from *0.0* to *10.0*.
float **GetSpeed**()	Returns the current ***Speed***.
void **SetDelay**(float value)	Sets the ***Delay*** *value*, which is a time offset between the shape animation across the paths. Valid values for *value* range from *-100.0* to *100.0*.
float **GetDelay**()	Returns the current ***Delay***.
void **SetDelay2**(float value)	Sets the ***Delay 2*** *value*, which is a time offset between the shape animation along the paths. Valid values for *value* range from *-100.0* to *100.0*.
float **GetDelay2**()	Returns the current ***Delay 2***.
void **SetDisplayMode**(int value)	Sets the ***Display Mode***. See below for a list of constants.
int **GetDisplayMode**()	Returns the current ***Display Mode***. See below for a list of constants.

Display Mode Constants

Constant	Description
int **DISPLAY_MODE_CLIP**	Sets the display mode to ***Clip***.
int **DISPLAY_MODE_FIT**	Sets the display mode to ***Fit***.
int **DISPLAY_MODE_STRETCH**	Sets the display mode to ***Stretch***.

5.2.27 SCE Screen Capture

Functions Provided By SCE Screen Capture

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the Position Control. Learn more »Using Position Control

Function	Description
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.
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 *angle* are *0*, *90*, *180*, and *270*.
int **GetRotation**()	Returns the currently set ***Rotation***.
void **SetTiling**(int enable)	Disables ***Tile Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Tile Mode*** will be enabled via 1 (**TRUE**).
int **GetTiling**()	Returns 1 (**TRUE**) if ***Tile Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleTiling**()	Enables ***Tile Mode*** or uses the default setting, depending on the current state.
void **SetSeamless**(int enable)	Disables ***Seamless*** if *enable* is 0 (**FALSE**), otherwise it will be enabled.
int **GetSeamless**()	Returns 1 (**TRUE**) if ***Seamless*** is enabled, otherwise 0 (**FALSE**).
void **ToggleSeamless**()	Enables ***Seamless*** or uses the default setting, depending on the current state.
float **GetImageWidth**()	Returns the image width of the current image in percent of the matrix width.
float **GetImageHeight**()	Returns the image height of the current image in percent of the matrix height.
float **GetImageDepth**()	Returns the image depth of the current image in percent of the matrix depth.
int **GetImagePixelWidth**()	Returns the image width of the current image in pixels.
int **GetImagePixelHeight**()	Returns the image height of the current image in pixels.

int **GetImagePixelDepth**()	Returns the image depth of the current imagein pixel.
float **GetCurrentPositionX**()	Returns the X-position of the current image in percent of the matrix width.
float **GetCurrentPositionY**()	Returns the Y-position of the current image in percent of the matrix height.
float **GetCurrentPositionZ**()	Returns the Z-position of the current image in percent of the matrix depth.
int **GetCurrentPixelPositionX**()	Returns the X-position of the current image in pixels.
int **GetCurrentPixelPositionY**()	Returns the Y-position of the current image in pixels.
int **GetCurrentPixelPositionZ**()	Returns the Z-position of the current image in pixels.
void **SetStretchMode**(int mode)	Sets the ***Stretch*** mode. See below for details.
int **GetStretchMode**()	Returns the ***Stretch*** mode. See below for details.
void **SetGrayscale**(int enable)	Disables ***Grayscale*** if *enable* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**).
int **GetGrayscale**()	Returns 1 (**TRUE**) if ***Stretch*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleGrayscale**()	Enables ***Stretch*** mode or uses the default setting, depending on the current state.
void **SetRgbw**(int enable)	Disables the ***RGB-To-RGBW*** mode if *enable* is 0 (**FALSE**), otherwise it can be enabled using 1 (**TRUE**).
int **GetRgbw**()	Returns 1 (**TRUE**) if ***RGB-To-RGBW*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleRgbw**()	Enables ***RGB-To-RGBW*** mode or uses the default setting, depending on the current state.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See below for details.
int **GetFilteringMode**()	Returns the ***Filtering*** mode. See below for details.
void **SetCaptureCoordinateTop**(int coordinate)	Sets the ***Top*** *coordinate* of the capture area.
int **GetCaptureCoordinateTop**()	Returns the currently set ***Top*** coordinate of the capture area.
void **SetCaptureCoordinateLeft** (int coordinate)	Sets the ***Left*** *coordinate* of the capture area.
int **GetCaptureCoordinateLeft**()	Returns the currently set ***Left*** coordinate of the capture area.
void **SetCaptureAreaWidth**(int width)	Sets the ***Width*** of the capture area.
int **GetCaptureAreaWidth**()	Returns the currently set ***Width*** of the capture area.
void **SetCaptureAreaHeight**(int height)	Sets the ***Height*** of the capture area.
int **GetCaptureAreaHeight**()	Returns the currently set ***Height*** of the capture area.
void **SetCaptureArea**(int left, int top, int width, int height)	Sets the whole capture area with its ***left*** and ***top*** coordinates as well as the required *width* and *height*.
void **SetCaptureFrameRate**(float value)	Sets the ***Capture Frame Rate*** to use for screen capturing. The default value is 50.0.
float **GetCaptureFrameRate**()	Returns the currently set ***Capture Frame Rate***.
void **SetAeroOff**()	Activates ***Aero Off***.
int **GetAeroOff**()	Returns if ***Aero Off*** is enabled (*1*) or disabled (*0*).

Stretch Mode Constants

Value	Description
int **STRETCH_MODE_NONE**	Sets the stretch mode to ***None*** (no stretching).
int **STRETCH_MODE_MATRIX**	Sets the stretch mode to ***Matrix*** (stretches to the current aspect ratio of the matrix).
int **STRETCH_MODE_ORIGINAL**	Sets the stretch mode to ***Original*** (stretches to the original aspect ratio of the source).
int **STRETCH_MODE_4_TO_3**	Sets the stretch mode to ***4:3***.
int **STRETCH_MODE_16_TO_9**	Sets the stretch mode to ***16:9***.

Filtering Mode Constants

Value	Description
int **FILTERING_MODE_NEAREST_NEIGHBOR**	Sets the Filtering Mode to ***None*** (no filtering).
int **FILTERING_MODE_LINEAR**	Sets the Filtering Mode to ***Linear*** (may require additional performance).

5.2.28 SCE Shapes

Functions Provided By SCE Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.

void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetFade**(float value)	Sets the ***Fade*** *value*, which defines how fast the shapes disappear in BPM. Valid values for *value* range from *1* to *3000*.
float **GetFade**()	Returns the current ***Fade*** in BPM.
void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes, in percent of the matrix size (depending on the shape alignment). The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetPixelSize**(int value)	Sets the ***Size*** *value*, which is the size of the shapes, in pixels. The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSize**()	Returns the current ***Size*** of the shapes in pixels. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetPixelSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in pixels. The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSizeMax**()	Returns the current ***Maximum Size*** of the shapes in pixels.
void **SetSizeDistribution**(int value)	Sets the size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »here for a list of constants.
int **GetSizeDistribution**()	Returns the current size ***Distribution*** of the shapes. See »here for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Is the same as **SetSizeDistribution**(`DIST_RND`).

void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch1**()	Returns the current ***Pitch 1*** of the shapes in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch1**()	Returns the current ***Pitch 1*** of the shapes in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch2**()	Returns the current ***Pitch 2*** of the shapes in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch2**()	Returns the current ***Pitch 2*** of the shapes in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch3**()	Returns the current ***Pitch 3*** of the shapes in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch3**()	Returns the current ***Pitch 3*** of the shapes in pixels.
void **SetPitch**(float x, float y, float z)	Sets the pitches 1, 2 and 3 of the shapes at once, in percent of the matrix size. Valid values for *x*, *y* and *z* range from *0.01* to *100*.
void **SetPixelPitch**(int x, int y, int z)	Sets the pitches 1, 2 and 3 of the shapes at once, in pixels. Valid values for *x*, *y* and *z* range from *1* to the matrix size.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetCount**(int value)	Sets the ***Count*** *value*, which defines how much shapes are created at once. Valid values for *value* range from *1* to *100*. Use BPM and ***Fade*** as well to control how much shapes are displayed.
int **GetCount**()	Returns the current ***Count***.

5.2.29 SCE Simple Shape

Functions Provided By SCE Simple Shape

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »**Using Color Controls**
- This Effect uses Shapes. Learn more »**Using Shapes**
- This Effect uses Shape Rotation. Learn more »**Using Shape Rotation**
- This Effect uses the Size Control. Learn more »**Using Size Control**
- This Effect uses the Position Control. Learn more »**Using Position Control**

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetOuterGlowInterpolationType**(int type)	Sets the ***Outer Glow Interpolation Type***. See **below** for a list of constants.
float **GetOuterGlowInterpolationType**()	Returns the current ***Outer Glow Interpolation Type*** .
void **SetInnerGlowInterpolationType**(int type)	Sets the ***Inner Glow Interpolation Type***. See **below** for a list of constants.
int **GetInnerGlowInterpolationType**()	Returns the current ***Inner Glow Interpolation Type*** .
void **SetProportion**(float value)	Sets the ***Proportion*** of objects. Valid values range from *0.01* to *100*.
float **GetProportion**()	Returns the currently set ***Proportion***.
void **SetDiagonals**(float value)	Sets the ***Diagonals*** of objects. Valid values range from *0.01* to *100*.
float **GetDiagonals**()	Returns the currently set ***Diagonals***.

Interpolation Type Constants

Constant	Description
int `INTERPOLATION_TYPE_LINEAR`	Sets the Interpolation Type to ***Linear***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_IN`	Sets the Interpolation Type to ***Ease In Bounce***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_OUT`	Sets the Interpolation Type to ***Ease Out Bounce***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_INOUT`	Sets the Interpolation Type to ***Ease In Out Bounce***.
int `INTERPOLATION_TYPE_EASE_CIRC_IN`	Sets the Interpolation Type to ***Ease In Circular***.

int **INTERPOLATION_TYPE_EASE_CIRC_OUT**	Sets the Interpolation Type to ***Ease Out Circular***.
int **INTERPOLATION_TYPE_EASE_CIRC_INOUT**	Sets the Interpolation Type to ***Ease In Out Circular***.
int **INTERPOLATION_TYPE_EASE_CUBIC_IN**	Sets the Interpolation Type to ***Ease In Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_OUT**	Sets the Interpolation Type to ***Ease Out Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_INOUT**	Sets the Interpolation Type to ***Ease In Out Cubic***.
int **INTERPOLATION_TYPE_EASE_SINE_IN**	Sets the Interpolation Type to ***Ease In Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_OUT**	Sets the Interpolation Type to ***Ease Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_INOUT**	Sets the Interpolation Type to ***Ease In Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_EXPO_IN**	Sets the Interpolation Type to ***Ease In Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_OUT**	Sets the Interpolation Type to ***Ease Out Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_INOUT**	Sets the Interpolation Type to ***Ease In Out Exponential*** .

5.2.30 SCE Split Shapes

Functions Provided By SCE Split Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »**Using Color Table**
- This Effect uses the Shape Table. Learn more »**Using Shape Table**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**
- This Effect uses the Center Control. Learn more »**Using Center Control**

Function	Description
void **SetEmissions**(int value)	Sets the ***Emissions*** *value*.
int **GetEmissions**()	Returns the currently set ***Emissions*** value.

void **SetGenerations**(int value)	Sets the ***Generations*** value.
int **GetGenerations**()	Returns the currently set ***Generations*** value.
void **SetVelocity**(float value)	Sets the ***Velocity*** value.
float **GetVelocity**()	Returns the currently set ***Velocity*** value.
void **SetVelocityMin**(float value)	Sets the ***Minimum Velocity*** value.
float **GetVelocityMin**()	Returns the currently set ***Minimum Velocity*** value.
void **SetVelocityMax**(float value)	Sets the ***Maximum Velocity*** value.
float **GetVelocityMax**()	Returns the currently set ***Maximum Velocity*** value.
void **SetVelocityDistributionMode**(int value)	Sets the Velocity ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Velocity is applied. See »here for a list of constants.
int **GetVelocityDistributionMode**()	Returns the current Velocity ***Distribution***. See »here for a list of constants.
void **SetLifetime**(float value)	Sets the ***Lifetime*** *value*.
float **GetLifetime**()	Returns the currently set ***Lifetime*** value.
void **SetLifetimeMin**(float value)	Sets the ***Minimum Lifetime*** value.
float **GetLifetimeMin**()	Returns the currently set ***Minimum Lifetime*** value.
void **SetLifetimeMax**(float value)	Sets the ***Maximum Lifetime*** value.
float **GetLifetimeMax**()	Returns the currently set ***Maximum Lifetime*** value.
void **SetLifetimeDistributionMode**(int value)	Sets the Lifetime ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Lifetime is applied. See »here for a list of constants.
int **GetLifetimeDistributionMode**()	Returns the current Lifetime ***Distribution***. See »here for a list of constants.
void **SetSpread**(float value)	Sets the ***Spread*** *value*.
float **GetSpread**()	Returns the currently set ***Spread*** value.
void **SetSpreadMin**(float value)	Sets the ***Minimum Spread*** value.
float **GetSpreadMin**()	Returns the currently set ***Minimum Spread*** value.
void **SetSpreadMax**(float value)	Sets the ***Maximum Spread*** value.
float **GetSpreadMax**()	Returns the currently set ***Maximum Spread*** value.
void **SetSpreadDistributionMode**(int value)	Sets the Spread ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Spread is applied. See »here for a list of constants.
int **GetSpreadDistributionMode**()	Returns the current Spread ***Distribution***. See »here for a list of constants.
void **SetSize**(float value)	Sets the ***Size*** *value*.
float **GetSize**()	Returns the currently set ***Size*** value.
void **SetSizeMin**(float value)	Sets the ***Minimum Size*** value.
float **GetSizeMin**()	Returns the currently set ***Minimum Size*** value.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** value.
float **GetSizeMax**()	Returns the currently set ***Maximum Size*** value.

void **SetSizeDistributionMode**(int value)	Sets the Size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Size is applied. See »here for a list of constants.
int **GetSizeDistributionMode**()	Returns the current Size ***Distribution***. See »here for a list of constants.
void **SetSwirl**(float value)	Sets the ***Swirl*** *value*.
float **GetSwirl**()	Returns the currently set ***Swirl*** value.
void **SetEmissionAngle**(float value)	Sets the ***Angle*** *value*.
float **GetEmissionAngle**()	Returns the currently set ***Angle*** value.
void **SetFadeout**(float value)	Sets the ***Fade*** *value*.
float **GetFadeout**()	Returns the currently set ***Fade*** value.
void **SetAlphaMix**(float value)	Sets the ***Alpha*** *value*.
float **GetAlphaMix**()	Returns the currently set ***Alpha*** value.
void **SetPathAligned**(int value)	Sets the ***PA*** option. Use 1 (**TRUE**) to activate Path Alignment. Use 0 (**FALSE**) to deactivate it.
int **GetPathAligned**()	Returns 1 if ***PA*** is activated, otherwise 0.
void **SetAnimationDuration**(float value)	Sets the ***Duration*** *value*.
float **GetAnimationDuration**()	Returns the currently set ***Duration*** value.
void **SetCenterMode**(int value)	Sets the ***Center*** mode. See below for a list of constants.
int **GetCenterMode**()	Returns the currently set ***Center*** mode.
void **SetColorMode**(int value)	Sets the *color mode*. See below for a list of constants.
int **GetColorMode**()	Returns the currently set *color mode*.
void **SetShapeMode**(int value)	Sets the *shape mode*. See below for a list of constants.
int **GetShapeMode**()	Returns the currently set *shape mode*.

Center Mode Constants

Constant	Description
int **CENTER_MODE_FIXED**	Sets the center mode to ***Fixed***.
int **CENTER_MODE_RANDOM**	Sets the center mode to ***Random***.

Color Mode Constants

Constant	Description
int **COLOR_MODE_INHERITING**	Sets the color mode to ***Inheriting***.
int **COLOR_MODE_PER_CYCLE**	Sets the color mode to ***Per Cycle***.
int **COLOR_MODE_PER_GENERATION**	Sets the color mode to ***Per Generation***.
int **COLOR_MODE_ROUND_ROBIN**	Sets the color mode to ***Round Robin***.

Shape Mode Constants

Constant	Description
int **SHAPE_MODE_INHERITING**	Sets the shape mode to ***Inheriting***.
int **SHAPE_MODE_PER_CYCLE**	Sets the shape mode to ***Per Cycle***.
int **SHAPE_MODE_PER_GENERATION**	Sets the shape mode to ***Per Generation***.
int **SHAPE_MODE_ROUND_ROBIN**	Sets the shape mode to ***Round Robin***.

5.2.31 SCE Starfield

Functions Provided By SCE Starfield

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetLength**(int value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is. Valid values for *value* range from *1* to *1000*.
int **GetLength**()	Returns the current ***Length***.
void **SetCount**(int value)	Sets the ***Count*** *value*, which is the number of the displayed shapes. Valid values for *value* range from *1* to *1000*.
int **GetCount**()	Returns the current ***Count***.

void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes in percent of the matrix size (depending on the look-at type). The ***Size*** *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetPixelSize**(int value)	Sets the size *value*, which is the size of the shapes, in pixels. The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the look-at type).
int **GetPixelSize**()	Returns the current ***Size*** of the shapes in pixels. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes in percent of the matrix size (depending on the look-at type). The ***Maximum Size*** *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetPixelSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in pixels. The ***Maximum Size*** *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the look-at type).
int **GetPixelSizeMax**()	Returns the current ***Maximum Size*** of the shapes in pixels.
void **SetSizeDistribution**(int value)	Sets the ***Size Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »**here** for a list of constants.
int **GetSizeDistribution**()	Returns the current ***Size Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Same as **SetSizeDistribution**(**DIST_RND**).
void **SetRotation**(int value)	Sets the ***Rotation*** *value*, which is the rotation of the flight paths, in degrees. The ***Rotation*** *value* means the minimum rotation if the rotation distribution is not uniform. Valid values for *value* range from *-3600* to *3600*.
int **GetRotation**()	Returns the current ***Rotation*** of the flight paths in degrees. The return value means the minimum rotation if the rotation distribution is not uniform.
void **SetRotationMax**(int value)	Sets the ***Maximum Rotation*** *value*, which is the maximum rotation of the flight paths, in degrees. The ***Maximum Rotation*** *value* only applies if the rotation distribution is not uniform. Valid values for *value* range from *-3600* to *3600*.
int **GetRotationMax**()	Returns the current ***Maximum Rotation*** of the flight paths in degrees.
void **SetRotationDistribution**(int value)	Sets the ***Rotation Distribution*** *value*, which defines how the range between the current minimum and the current maximum rotation is applied to the flight paths. See »**here** for a list of constants.
int **GetRotationDistribution**()	Returns the current ***Rotation Distribution*** of the flight paths. See » **here** for a list of constants.
void **SeedRandomRotation**()	Randomizes the rotations of the flight paths. Same as **SetRotationDistribution**(**DIST_RND**).

void **SetDistortion**(int value)	Sets the ***Distortion*** *value*, which defines how deep the 3D space appears. Valid values for *value* range from *1* to *100*.
int **GetDistortion**()	Returns the current ***Rotation Distribution***.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the flight paths, which means that the objects move from near to far. Use *value* 0 (**FALSE**) to use the default setting, which is from far to near.
int **GetInvert**()	Returns 1 (**TRUE**) if the objects move from near to far, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the flight paths or uses the default setting, depending on the current state.
void **SetPathAligned**(int value)	Use *value* 1 (**TRUE**) to align the rotation of the shapes to the flight paths. Use *value* 0 (**FALSE**) otherwise.
int **GetPathAligned**()	Returns 1 (**TRUE**) if the rotation of the shapes is aligned to the flight paths, otherwise 0 (**FALSE**).
Function	**Description**
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.

5.2.32 SCE Swarm

Functions Provided By SCE Swarm

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetBounce**(int enable)	Disables ***Bounce Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Bounce Mode*** will be enabled via 1 (**TRUE**).
int **GetBounce**()	Returns 1 (**TRUE**) if ***Bounce Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleBounce**()	Disables ***Bounce Mode*** or uses the default setting, depending on the current state.
void **SetCount**(int value)	Sets the ***Count*** *value*, which defines how much objects are created at once.
int **GetCount**()	Returns the current ***Count*** value.
void **SetInfluence**(float value)	Sets the ***Influence*** value.
float **GetInfluence**()	Returns the current ***Influence*** value.
void **SetSeparation**(float value)	Sets the ***Separation*** value.
float **GetSeparation**()	Returns the current ***Separation*** value.
void **SetImitation**(float value)	Sets the ***Imitation*** value.
float **GetImitation**()	Returns the current ***Imitation*** value.
void **SetLength**(float value)	Sets the ***Length*** value.
float **GetLength**()	Returns the current ***Length*** value.

5.2.33 SCE Ticker / Scrolling Text

Functions Provided By SCE Ticker / Scrolling Text

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the Position Control. Learn more »Using Position Control

Function	Description
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.
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 *angle* are *0*, *90*, *180*, and *270*.
int **GetRotation**()	Returns the currently set ***Rotation***.
void **SetTiling**(int enable)	Disables ***Tile Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Tile Mode*** will be enabled via 1 (**TRUE**).
int **GetTiling**()	Returns 1 (**TRUE**) if ***Tile Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleTiling**()	Enables ***Tile Mode*** or uses the default setting, depending on the current state.
void **SetSeamless**(int enable)	Disables ***Seamless*** if *enable* is 0 (**FALSE**), otherwise it will be enabled.
int **GetSeamless**()	Returns 1 (**TRUE**) if ***Seamless*** is enabled, otherwise 0 (**FALSE**).
void **ToggleSeamless**()	Enables ***Seamless*** or uses the default setting, depending on the current state.
float **GetImageWidth**()	Returns the image width of the current image in percent of the matrix width.
float **GetImageHeight**()	Returns the image height of the current image in percent of the matrix height.
float **GetImageDepth**()	Returns the image depth of the current image in percent of the matrix depth.
int **GetImagePixelWidth**()	Returns the image width of the current image in pixels.
int **GetImagePixelHeight**()	Returns the image height of the current image in pixels.

int **GetImagePixelDepth**()	Returns the image depth of the current imagein pixel.
float **GetCurrentPositionX**()	Returns the X-position of the current image in percent of the matrix width.
float **GetCurrentPositionY**()	Returns the Y-position of the current image in percent of the matrix height.
float **GetCurrentPositionZ**()	Returns the Z-position of the current image in percent of the matrix depth.
int **GetCurrentPixelPositionX**()	Returns the X-position of the current image in pixels.
int **GetCurrentPixelPositionY**()	Returns the Y-position of the current image in pixels.
int **GetCurrentPixelPositionZ**()	Returns the Z-position of the current image in pixels.
void **SetText**(string text)	Sets the ***Text*** the effect displays.
string **GetText**()	Retrieves the ***Text*** the effect displays.
void **SetTextSplittingMode**(int mode)	Sets the text ***Splitting*** mode. See **below** for further details.
int **GetTextSplittingMode**()	Returns the current text ***Splitting*** mode. See **below** for further details.
viod **SetReverseCharacters**(int enable)	Disables ***Reverse Characters*** if *enable* is set to 0 (**FALSE**). Otherwise, use 1 (**TRUE**).
int **GetReverseCharacters**()	Returns 1 (**TRUE**) if ***Reverse Characters*** is active, otherwise 0 (**FALSE**) is returned.
void **SetReverseWords**(int enable)	Disables ***Reverse Characters*** if *enable* is set to 0 (**FALSE**). Otherwise, use 1 (**TRUE**).
int **GetReverseWords**()	Returns 1 (**TRUE**) if ***Reverse Characters*** is active, otherwise 0 (**FALSE**) is returned.
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*** with value 1 (**TRUE**) or not with value 0 (**FALSE**).
int **GetFontUnderline**()	Returns if the currently used font is ***underlined***.
void **SetFontUnderline**(int value)	Sets ***underlining*** for the font with value 1 (**TRUE**) or not with value 0 (**FALSE**).
int **GetFontStrikeOut**()	Returns if the a ***strikeout*** font is used.
void **SetFontStrikeOut**(int value)	Sets ***strikeout*** for the font with value 1 (**TRUE**) or not with value 0 (**FALSE**).
int **GetFontWeight**()	Returns the ***Weight*** of the currently used font.
void **SetFontWeight**(int value)	Sets the ***Weight*** of the font. Valid values for *value* range from *0* to *1000*.
string **GetFontFaceName**()	Returns the ***Name*** of the font currently in use.

void **SetFontFaceName**(string name)	Sets which font to use. Use the exact font name. A maximum of 31 characters is allowed for *string*. Example: SetFontFaceName("MS Sans Serif");
void **SetStretchMode**(int mode)	Sets the ***Stretch*** mode. See below for details.
int **GetStretchMode**()	Returns the ***Stretch*** mode. See below for details.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See below for details.
int **GetFilteringMode**()	Returns the ***Filtering*** mode. See below for details.

Stretch Mode Constants

Value	Description
int `STRETCH_MODE_NONE`	Sets the stretch mode to ***None*** (no stretching).
int `STRETCH_MODE_MATRIX`	Sets the stretch mode to ***Matrix*** (stretches to the current aspect ratio of the matrix).
int `STRETCH_MODE_ORIGINAL`	Sets the stretch mode to ***Original*** (stretches to the original aspect ratio of the source).
int `STRETCH_MODE_4_TO_3`	Sets the stretch mode to ***4:3***.
int `STRETCH_MODE_16_TO_9`	Sets the stretch mode to ***16:9***.

Filtering Mode Constants

Value	Description
int `FILTERING_MODE_NEAREST_NEIGHBOR`	Sets the Filtering Mode to ***None*** (no filtering).
int `FILTERING_MODE_LINEAR`	Sets the Filtering Mode to ***Linear*** (may require additional performance).

Text Mode Constants

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 **SetTextSplittingMode** it is possible to set the mode using the following parameters:

Constant	Description
int **MODE_SENTENCE**	Sets the sentence mode (***Splitting None***).
int **MODE_WORD**	Sets the word mode (***Splitting Words***).
int **MODE_CHAR**	Sets the character mode (***Splitting Characters***).

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("Hello World!");
    SetDirection(DIRECTION_TOP);
}

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() == DIRECTION_LEFT) {
        if(GetCurrentPixelPositionX() < 0) {
            SetDirection(DIRECTION_RIGHT);
        }
    } else if(GetDirection() == DIRECTION_RIGHT) {
        if(GetCurrentPixelPositionX() > GetMatrixWidth()- GetImagePixelWidth()) {
```

```
                SetDirection(DIRECTION_LEFT);
            }
        } else {
                SetDirection(DIRECTION_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;
}
```

5.2.34 SCE Tubes

Functions Provided By SCE Tubes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetFade**(int value)	Sets the ***Fade*** value.
int **GetFade**()	Returns the ***Fade*** value.
void **SetSize1**(float value)	Sets the ***Size 1*** value
float **GetSize1**()	Returns the ***Size 1*** value.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** value
float **GetPitch1**()	Returns the ***Pitch 1*** value.

void **SetSize2**(float value)	Sets the ***Size 2*** value
float **GetSize2**()	Returns the ***Size 2*** value.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value
float **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** value.
int **GetAlphaMix**()	Returns the ***Alpha Mix*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggles the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggles the ***Mirror*** option.
void **SetTubesVertical**(int value)	Use 1 (**TRUE**) to activate ***vertical Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesVertical**()	Returns 1 (**TRUE**) if ***vertical Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesVertical**()	Toggles ***vertical Tubes***.
void **SetTubesHorizontal**(int value)	Use 1 (**TRUE**) to activate ***horizontal Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesHorizontal**()	Returns 1 (**TRUE**) if ***horizontal Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesHorizontal**()	Toggles ***horizontal Tubes***.
void **SetTubesDepth**(int value)	Use 1 (**TRUE**) to activate ***depth Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesDepth**()	Returns 1 (**TRUE**) if ***depth Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesDepth**()	Toggles ***depth Tubes***.
void **SetCount**(int value)	Sets the ***Count*** value.
int **GetCount**()	Returns the ***Count*** value.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See below for a list of constants.
int **GetGeneratorType**()	Returns the current ***Generator Type***. See below for a list of constants.

Generator Type Constants

Constant	Description
int **GENERATOR_TYPE_RANDOM**	Sets the generator type to ***Random***.
int **GENERATOR_TYPE_LINEAR**	Sets the generator type to ***Linear***.
int **GENERATOR_TYPE_PING_PONG**	Sets the generator type to ***Ping Pong***.

5.2.35 SCE Video

Functions Provided By SCE Video

This effect uses the following functions:

Note: Certain functions may not work due to restrictions and limitations of the video codec that a video is using and encoded with. Please see the MADRIX Software's Help And Manual for more information.

- This Effect uses Directions. Learn more »**Using Direction**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**
- This Effect uses the Color Controls. Learn more »**Using Color Controls**
- This Effect uses the Position Control. Learn more »**Using Position Control**

Function	Description
void **SetExtrusion**(float value)	Sets the ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
float **GetExtrusion**()	Returns the current ***Extrusion*** *value* in percent of the matrix size (depending on the current look-at type).
void **SetPixelExtrusion**(int value)	Sets the ***Extrusion*** *value* in pixels.
int **GetPixelExtrusion**()	Returns the current ***Extrusion*** *value* in pixels.
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 *angle* are *0*, *90*, *180*, and *270*.
int **GetRotation**()	Returns the currently set ***Rotation***.
void **SetTiling**(int enable)	Disables ***Tile Mode*** if *enable* is set to 0 (**FALSE**), otherwise the ***Tile Mode*** will be enabled via 1 (**TRUE**).
int **GetTiling**()	Returns 1 (**TRUE**) if ***Tile Mode*** is enabled, otherwise 0 (**FALSE**).
void **ToggleTiling**()	Enables ***Tile Mode*** or uses the default setting, depending on the current state.
void **SetSeamless**(int enable)	Disables ***Seamless*** if *enable* is 0 (**FALSE**), otherwise it will be enabled.

int **GetSeamless**()	Returns 1 (**TRUE**) if ***Seamless*** is enabled, otherwise 0 (**FALSE**).
void **ToggleSeamless**()	Enables ***Seamless*** or uses the default setting, depending on the current state.
float **GetImageWidth**()	Returns the image width of the current image in percent of the matrix width.
float **GetImageHeight**()	Returns the image height of the current image in percent of the matrix height.
float **GetImageDepth**()	Returns the image depth of the current image in percent of the matrix depth.
int **GetImagePixelWidth**()	Returns the image width of the current image in pixels.
int **GetImagePixelHeight**()	Returns the image height of the current image in pixels.
int **GetImagePixelDepth**()	Returns the image depth of the current imagein pixel.
float **GetCurrentPositionX**()	Returns the X-position of the current image in percent of the matrix width.
float **GetCurrentPositionY**()	Returns the Y-position of the current image in percent of the matrix height.
float **GetCurrentPositionZ**()	Returns the Z-position of the current image in percent of the matrix depth.
int **GetCurrentPixelPositionX**()	Returns the X-position of the current image in pixels.
int **GetCurrentPixelPositionY**()	Returns the Y-position of the current image in pixels.
int **GetCurrentPixelPositionZ**()	Returns the Z-position of the current image in pixels.
void **SetAutostart**(int enable)	Disables ***Automatic Start*** if *enable* is set to 0 (**FALSE**). Otherwise, auto start can be enabled using 1 (**TRUE**).
int **GetAutostart**()	Returns 1 (**TRUE**) if ***Automatic Start*** is enabled, otherwise 0 (**FALSE**).
void **SetLoopingMode**(int mode)	Sets the ***Looping*** mode. See below for details.
int **GetLoopingMode**()	Returns the ***Looping*** mode. See below for details.
int **GetVideoLoaded**()	Returns 1 (**TRUE**) if a video is already loaded.
int **GetVideoRunning**()	Returns 1 (**TRUE**) if video ***Playback*** is active, otherwise 0 (**FALSE**).
void **StartVideo**()	Toggles the video ***Playback***. Starts the video, when stopped and stops the video when it runs.
void **StopVideo**()	Stops video ***Playback***.
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, 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 **StartVideoBackward**()	Starts playing the video ***backwards***. Is only available with QuickTime video files.
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 **SetVideoPlaybackRate**(float rate)	Sets the ***Playback Rate*** of the video. E.g., a *value* of 2.0 means that the video will be running 2x faster than the original speed.
float **GetVideoPlaybackRate**()	Returns the current ***Playback Rate*** of the video.
void **StepForward**()	Steps the video one frame forward.
void **StepBackward**()	Steps the video one frame backward.
void **SetStretchMode**(int mode)	Returns the ***Stretch*** mode. See **below** for details.
int **GetStretchMode**()	Returns the ***Stretch*** mode. See **below** for details.
void **SetGrayscale**(int enable)	Disables ***Grayscale*** if *enable* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**).
int **GetGrayscale**()	Returns 1 (**TRUE**) if ***Grayscale*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleGrayscale**()	Enables ***Grayscale*** mode or uses the default setting, depending on the current state.
void **SetRgbw**(int enable)	Disables the ***RGB-To-RGBW*** mode if *enable* is 0 (**FALSE**), otherwise it can be enabled using 1 (**TRUE**).
int **GetRgbw**()	Returns 1 (**TRUE**) if ***RGB-To-RGBW*** mode is enabled, otherwise 0 (**FALSE**).
void **ToggleRgbw**()	Enables ***RGB-To-RGBW*** mode or uses the default setting, depending on the current state.
void **SetFilteringMode**(int mode)	Sets the ***Filtering*** mode. See **below** for details.
int **GetFilteringMode**()	Returns the ***Filtering*** mode. See **below** for details.

Stretch Mode Constants

Value	Description
int **STRETCH_MODE_NONE**	Sets the stretch mode to ***None*** (no stretching).
int **STRETCH_MODE_MATRIX**	Sets the stretch mode to ***Matrix*** (stretches to the current aspect ratio of the matrix).
int **STRETCH_MODE_ORIGINAL**	Sets the stretch mode to ***Original*** (stretches to the original aspect ratio of the source).
int **STRETCH_MODE_4_TO_3**	Sets the stretch mode to ***4:3***.
int **STRETCH_MODE_16_TO_9**	Sets the stretch mode to ***16:9***.

Filtering Mode Constants

Value	Description
int **FILTERING_MODE_NEAREST_NEIGHBOR**	Sets the Filtering Mode to ***None*** (no filtering).
int **FILTERING_MODE_LINEAR**	Sets the Filtering Mode to ***Linear*** (may require additional performance).

Looping Mode Constants

This effect uses various playback modes. The function **SetLoopingMode** can be used to change them. The following values can be used as parameter:

Constant	Description
int **PLAYBACK_MODE_ONCE**	Sets the Playback Mode to ***Once*** (no looping).
int **PLAYBACK_MODE_LOOP**	Sets the Playback Mode to ***Loop*** (forward or backward looping).
int **PLAYBACK_MODE_PING_PONG**	Sets the Playback Mode to ***Ping Pong*** (changing the playback direction when reaching the video end or start time).

Setting the Time for Video Playback

If a video is loaded that has a length of 2:45:00 for example, the following source code would set the playback position to 1:25:30.

```
time t = {1, 25, 30};
SetVideoTime(t);
```

Note: If the given time is higher than the length of the video, the time is set to the end of the video.

SetVideoTime

5.2.36 SCE Water

Functions Provided By SCE Water

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetSpeedVariance**(int value)	Sets the ***SV*** *value*.
int **GetSpeedVariance**()	Returns the currently set ***SV*** *value*.
void **SetEnergy**(float value)	Sets the ***Energy*** *value*.
float **GetEnergy**()	Returns the currently set ***Energy*** *value*.
void **SetEnergyMax**(float value)	Sets the ***Maximum Energy*** *value*. It only applies if the Energy distribution is not uniform.
float **GetEnergyMax**()	Returns the currently set ***Maximum Energy***.
void **SetEnergyDistribution**(int mode)	Sets the Energy ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Energy is applied. See »here for a list of constants.
int **GetEnergyDistribution**()	Returns the current Energy ***Distribution***. See »here for a list of constants.
void **SetLength**(float)	Sets the ***Length*** *value*.
float **GetLength**()	Returns the currently set ***Length*** *value*.
void **SetLengthMax**(float)	Sets the ***Maximum Length*** *value*. It only applies if the Length distribution is not uniform.
float **GetLengthMax**()	Returns the currently set ***Maximum Length***.
void **SetLengthDistribution**(int)	Sets the Length ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Length is applied. See »here for a list of constants.
int **GetLengthDistribution**()	Returns the current Length ***Distribution***. See »here for a list of constants.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, in percent.
float **GetPitch1**()	Returns the ***Pitch 1*** *value*, in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, in pixels.
int **GetPixelPitch1**()	Returns the ***Pitch 1*** *value*, in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, in percent.
float **GetPitch2**()	Returns the ***Pitch 2*** *value*, in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, in pixels.

int **GetPixelPitch2**()	Returns the ***Pitch 2*** *value*, in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, in percent.
float **GetPitch3**()	Returns the ***Pitch 3*** *value*, in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, in pixels.
int **GetPixelPitch3**()	Returns the ***Pitch 3*** *value*, in pixels.
void **SetPitch**(float, float, float)	Sets the ***Pitch 1***, ***Pitch 2***, and ***Pitch 3*** *values*, in percent.
void **SetPixelPitch**(int, int, int)	Sets the ***Pitch 1***, ***Pitch 2***, and ***Pitch 3*** *values*, in pixels.
void **SetInterval**(float)	Sets the ***Interval*** *value*.
float **GetInterval**()	Returns the currently set ***Interval*** *value*.
void **SetIntervalMax**(float)	Sets the ***Maximum Interval*** *value*. It only applies if the Interval distribution is not uniform.
float **GetIntervalMax**()	Returns the currently set ***Maximum Interval***.
void **SetIntervalDistribution**(int)	Sets the Interval ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Interval is applied. See »here for a list of constants.
int **GetIntervalDistribution**()	Returns the current Interval ***Distribution***. See »here for a list of constants.
void **SetReflect**(int enable)	Disables ***Reflection*** if *enable* is set to 0 (**FALSE**). Otherwise, use 1 (**TRUE**).
int **GetReflect**()	Returns 1 (**TRUE**) if ***Reflection*** is active, otherwise 0 (**FALSE**) is returned.
void **ToggleReflect**()	Enables ***Reflection*** or uses the default setting, depending on the current state.

Function	Description
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.

5.2.37 SCE Wave / Radial

Functions Provided By SCE Wave/Radial

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses Shapes. Learn more »Using Shapes
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the Center Control. Learn more »Using Center Control

void **SetLength**(int value)	Sets the ***Length*** of the wave/radial, which is the wavelength in percent of the matrix size (depending on the current direction and the current shape).
int **GetLength**()	Returns the current ***Length*** of the wave/radial.
void **SetWidth**(int value)	Sets the ***Width*** of the wave/radial, which stretches or compresses the periodic part of the wave function in percent of the wavelength.
int **GetWidth**()	Returns the current ***Width*** of the wave/radial.
void **SetCount**(int value)	Sets the ***Count*** of tails. This feature is only supported by shape type **SHAPE_TYPE_WAVE_RADAR** and **SHAPE_TYPE_WAVE_HELIX**.
int **GetCount**()	Returns the current ***Count*** of tails.
void **SetFactor**(int value)	Sets the distortion ***Factor*** of the wave.
int **GetFactor**()	Returns the current distortion ***Factor*** of the wave.
void **SetAmplitude**(int value)	Sets the ***Amplitude*** *value* of the distortion factor. This feature is only supported when the distortion factor is higher than 0.
int **GetAmplitude**()	Return the current ***Amplitude*** of the distortion factor.
void **SetPeak**(float value)	Sets the ***Peak*** value of the wave.
float **GetPeak**()	Returns the ***Peak*** value of the wave.
void **SetWaveType**(int wavetype)	Sets the ***Wave Type***. See »here for a list of constants.
int **GetWaveType**()	Returns the current ***Wave Type***. See »here for a list of constants.
void **SetPhaseMode**(int phasemode)	Use *value* **PHASE_MODE_ON** to activate ***Phase Mode***. When ***Phase Mode*** is activated the color's green and blue color values will be phase-shifted. Use *value* **PHASE_MODE_OFF** to use the default settings.
int **GetPhaseMode**()	Returns **PHASE_MODE_ON** if ***Phase Mode*** is enabled, otherwise **PHASE_MODE_OFF**.
void **TogglePhaseMode**()	Enables or disables the ***Phase Mode*** depending on the current state.

void **SetGoingInDirection**(int direction)	Use *value* **GOING_IN_DIRECTION** to activate ***Going In Direction***. Use *value* **GOING_OUT_DIRECTION** to use the default settings. This feature is supported for all shape types, except for **SHAPE_TYPE_WAVE_LINEAR.**
int **GetGoingInDirection**()	Returns **GOING_IN_DIRECTION** if ***Going In Direction*** is enabled, otherwise **GOING_OUT_DIRECTION**.
void **ToggleGoingInDirection**()	Enables or disables the ***Going In Direction*** depending on the current state.
void **SetClockwiseRotation**(int rotation)	Use *value* **CLOCKWISE_ROTATION** to activate ***Clockwise Rotation***. Use *value* **COUNTER_CLOCKWISE_ROTATION** to use the default settings. This feature is only supported by shape type **SHAPE_TYPE_WAVE_RADAR** and **SHAPE_TYPE_WAVE_HELIX**.
int **GetClockwiseRotation**()	Returns **CLOCKWISE_ROTATION** if ***Clockwise Rotation*** is enabled, otherwise **COUNTER_CLOCKWISE_ROTATION**.
void **ToggleClockwiseRotation**()	Enables or disables ***Clockwise Rotation*** depending on the current state.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.

MADRIX 3.X To MADRIX 5.X Migration Hints

The following functions are not supported anymore. Please follow the hints to migrate your macros.

Function	Description
All functions of the Position Control	Use the Center Control instead.

5.3 Sound2Light Effects (S2L)

5.3.1 S2L Drops

Functions Provided By S2L Drops

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Shape Table. Learn more »Using Shape Table
- This Effect uses Shape Rotation. Learn more »Using Shape Rotation
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetSpeedVariance**(int value)	Sets the ***Speed Variance*** *value*, which means a random value between *0* and *500* percent. This value will multiply with the current BPM value.
int **GetSpeedVariance**()	Returns the current ***Speed Variance***.
void **SetLength**(float value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in percent of the matrix size (depending on the direction). The ***Length*** *value* means the minimum length if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLength**()	Returns the current ***Length*** in percent. The return value means the minimum length if the length distribution mode is not uniform.
void **SetPixelLength**(int value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in pixels. The ***Length*** *value* means the minimum length if the length distribution mode is not uniform.
int **GetPixelLength**()	Returns the current ***Length*** in pixels. The return value means the minimum length if the length distribution mode is not uniform.
void **SetLengthMin**(float value)	Is the same as **SetLength**.
float **GetLengthMin**()	Is the same as **GetLength**.
void **SetPixelLengthMin**(int value)	Is the same as **SetPixelLength**.
int **GetPixelLengthMin**()	Is the same as **GetPixelLength**.
void **SetLengthMax**(float value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in percent of the matrix size (depending on the direction). The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthMax**()	Returns the current ***Maximum Length*** in percent.

void **SetPixelLengthMax**(int value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in pixels. The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform.
int **GetPixelLengthMax**()	Returns the current ***Maximum Length*** in pixels.
void **SetLengthDistribution**(int value)	Sets the ***Length Distribution*** *value*, which defines how the range between the current minimum and the current maximum length is applied to the traces of the shapes. See »**here** for a list of constants.
int **GetLengthDistribution**()	Returns the current ***Length Distribution*** of the traces of the shapes. See »**here** for a list of constants.
void **SeedRandomLength**()	Randomizes the lengths of the traces of the shapes. Same as **SetLengthDistribution**(**DIST_RND**).
void **SetLengthOffset**(float value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthOffset**()	Returns the current ***Length Offset*** in percent.
void **SetPixelLengthOffset**(int value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in pixels.
int **GetPixelLengthOffset**()	Returns the current ***Length Offset*** in pixels.
void **SetRotationOffset**(float value)	Sets the ***Rotation Offset*** *value* between the shapes of the trace. Valid values for *value* range from -180.0 to 180.0.
float **GetRotationOffset**()	Returns the current ***Rotation Offset***.
void **SetPitch1**(float value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch1**()	Returns the current ***Pitch*** for axis one in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch1**()	Returns the current ***Pitch*** for axis one in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch2**()	Returns the current ***Pitch*** for axis two in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch2**()	Returns the current ***Pitch*** for axis two in pixels.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shape's positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shape's positions or uses the default setting, depending on the current state.

void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shape's positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shape's positions or uses the default setting, depending on the current state.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.
void **SetAmplification**(int value)	Sets the ***Amplification*** *value*, which controls how much shapes are created by defining a factor for the incoming audio data, in percent. Valid values for *value* range from *0* to *100*.
int **GetAmplification**()	Returns the current ***Amplification*** in percent.
void **SetShift**(int value)	Sets the ***Shift*** *value*, which controls the mapping between the incoming audio data and the shapes' positions by defining an offset, in percent. Valid values for *value* range from *0* to *100*.
int **GetShift**()	Returns the current ***Shift*** in percent.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See below for a list of constants.
int **GetGeneratorType**()	Returns the current ***Generator Type***. See below for a list of constants.
void **SetSensitivity**(int value)	Sets the ***Sensitivity*** *value*, which controls how much shapes are created by defining a threshold for the incoming audio data, in percent. Valid values for *value* range from *0* to *100*.
int **GetSensitivity**()	Returns the ***Sensitivity*** value.
void **SetPeak**(float)	Sets the ***Peak*** value. Valid values range from 0.00 to 100.00.
float **GetPeak**()	Returns the currently set ***Peak*** value.

Generator Type Constants

Constant	Description
int GENERATOR_TYPE_EQ	Sets the generator type to ***EQ***.
int GENERATOR_TYPE_LEVEL_MONO	Sets the generator type to ***Level Mono***.
int GENERATOR_TYPE_LEVEL_STEREO	Sets the generator type to ***Level Stereo***.

5.3.2 S2L EQ / Spectrum

Functions Provided By S2L EQ / Spectrum

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetDrop**(int value)	Sets the ***Drop*** *value*.
float **GetDrop**()	Returns the ***Drop*** value.
void **SetFade**(int value)	Sets the ***Fade*** *value*.
int **GetFade**()	Returns the ***Fade*** value.
void **SetWidth1**(float value)	Sets the ***Width 1*** *value*.
float **GetWidth1**()	Returns the ***Width 1*** value.
void **SetPixelWidth1**(int value)	Sets the ***Width 1*** in pixels.
int **GetPixelWidth1**()	Returns the ***Width 1*** in pixels.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*.
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** in pixels.
int **GetPixelPitch1**()	Returns the ***Pitch 1*** in pixels.
void **SetWidth2**(float value)	Sets the ***Width 2*** value.
float **GetWidth2**()	Returns the ***Width 2*** value.
void **SetPixelWidth2**(int value)	Sets the ***Width 2*** in pixels.
int **GetPixelWidth2**()	Returns the ***Width 2*** in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value.

int **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** in pixels.
int **GetPixelPitch2**()	Returns the ***Pitch 2*** in pixels.
void **SetLogarithmic**(int value)	Sets the ***Logarithmic*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetLogarithmic**()	Returns 1 (**TRUE**) if the ***Logarithmic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleLogarithmic**()	Toggles the ***Logarithmic*** option.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggle the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggle the ***Mirror*** option.
void **SetColorMode**(int value)	Sets the ***Color Mode***. See below for a list of constants.
int **GetColorMode**()	Returns the current ***Color Mode***. See below for a list of constants.
void **SetDRC**(int value)	Sets the ***DRC*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetDRC**()	Returns 1 (**TRUE**) if the ***DRC*** option is activated, otherwise 0 (**FALSE**) .
void **ToggleDRC**()	Toggle the ***DRC*** option.
void **SetShift**(int value)	Sets the ***Shift*** value.
int **GetShift**()	Returns the ***Shift*** value.

Color Mode Constants

Constant	Description
int **COLOR_MODE_MONOCHROME**	Sets the color mode to ***Monochrome***.
int **COLOR_MODE_BAND**	Sets the color mode to ***Band***.
int **COLOR_MODE_MATRIX**	Sets the color mode to ***Matrix***.

5.3.3 S2L Frequency Flash

Functions provided By S2L Frequency Flash

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls

Function	Description
void **SetFadeIn**(int fade)	Sets the ***Fade In*** value. This value represents Beats Per Minute (BPM).
int **GetFadeIn**()	Returns the current ***Fade In*** value.
void **SetFadeOut**(int fade)	Sets the ***Fade Out*** value. This value represents Beats Per Minute (BPM).
int **GetFadeOut**()	Returns the current ***Fade Out*** value.
void **SetSensitivity**(int value)	Set the ***Sensitivity*** value.
int **GetSensitivity**()	Returns the ***Sensitivity*** value.
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 of the available values and the appropriate bands.
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 of the available values and the appropriate bands.
int **GetMaxBand**()	Returns the specifier of the current ***Maximum Band***. See below for further details.

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	5
1	40
2	57
3	75
4	99
5	133
6	175
7	239
8	318
9	425
10	567
11	766
12	1013
13	1352
14	1832
15	2441
16	3269
17	4315
18	5769
19	7749
20	10.299
21	13.007

Example

The source code below would select 57 Hz as minimum band and 567 Hz as maximum band. Furthermore the color red is set with a fade in value of 3000 BPM and a fade out value of 120 BPM.

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

void InitEffect()
{
 SetColor(RED);
 SetFadeIn(3000);
```

```
  SetFadeOut(120);
  SetMinBand(2);
  SetMaxBand(10);
}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

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

5.3.4 S2L Level Color

Functions Provided By S2L Level Color

This effect uses the following functions:

- This Effect uses the Gradient Dialog. Learn more »**Using Gradient Dialog**
- This Effect uses Directions. Learn more »**Using Directions**

Function	Description
void **SetFadeIn**(int value)	Sets the ***Fade In*** value.
int **GetFadeIn**()	Returns the ***Fade In*** value.
void **SetFadeOut**(int value)	Sets the ***Fade Out*** value.
int **GetFadeOut**()	Returns the ***Fade Out*** value.
void **SetMode**(int value)	Sets the ***Mode***. See below for a list of constants.
int **GetMode**()	Returns the current ***Mode***. See below for a list of constants.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.

Mode Constants

Constant	Description
int `MODE_MONO`	Sets the mode to ***Mono***.
int `MODE_STEREO_LINEAR`	Sets the mode to ***Stereo Linear***.
int `MODE_STEREO_RADIAL`	Sets the mode to ***Stereo Radial***.

5.3.5 S2L Level Color Scroll

Functions Provided By S2L Level Color Scroll

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetStepWidth**(float value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using percentage values.
float **GetStepWidth**()	Returns the ***Step*** in percentage.

void **SetPixelStepWidth**(int value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using pixel values.
int **GetPixelStepWidth**()	Returns the ***Step*** in pixels.
void **SetColorWidth**(float value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using percentage values.
float **GetColorWidth**()	Returns the ***Color Width***, the width of a stripe, in percentage.
void **SetPixelColorWidth**(int value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using pixel values.
int **GetPixelColorWidth**()	Returns the ***Color Width***, the width of a stripe, in pixels.
void **SetCrossWidth**(float value)	Sets the ***Cross Width*** *value* using percentage values.
float **GetCrossWidth**()	Returns the ***Cross Width*** in percentage.
void **SetPixelCrossWidth**(int value)	Sets the ***Cross Width*** *value* using pixel values.
int **GetPixelCrossWidth**()	Returns the ***Cross Width*** in pixels.
void **SetCrossAxes**(int value)	Sets the ***Cross Axis*** *value*. See **below** for a list of constants.
int **GetCrossAxes**()	Returns the ***Cross Axis***. See **below** for a list of constants.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetLogarithmic**(int value)	Sets the ***Logarithmic*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetLogarithmic**()	Returns 1 (**TRUE**) if the ***Logarithmic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleLogarithmic**()	Toggles the ***Logarithmic*** option.
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from *0.0* to *100.0*.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from *0.0* to *100.0*.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.

Cross Axis Constants

Constant	Description
int **AXIS_ONE**	Sets the direction cross mode to ***Axis 1***.
int **AXIS_TWO**	Sets the direction cross mode to ***Axis 2***.
int **AXES_ONE_AND_TWO**	Sets the direction cross mode to ***Axes 1 And 2***.

5.3.6 S2L Level Meter

Functions Provided By S2L Level Meter

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetDrop**(int value)	Sets the ***Drop*** *value*.
float **GetDrop**()	Returns the ***Drop*** value.
void **SetFade**(int value)	Sets the ***Fade*** *value*.
int **GetFade**()	Returns the ***Fade*** value.
void **SetWidth1**(float value)	Sets the ***Width 1*** *value*.
float **GetWidth1**()	Returns the ***Width 1*** value.
void **SetPixelWidth1**(int value)	Sets the ***Width 1*** in pixels.
int **GetPixelWidth1**()	Returns the ***Width 1*** in pixels.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*.
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** in pixels.
int **GetPixelPitch1**()	Returns the ***Pitch 1*** in pixels.
void **SetWidth2**(float value)	Sets the ***Width 2*** value.
float **GetWidth2**()	Returns the ***Width 2*** value.
void **SetPixelWidth2**(int value)	Sets the ***Width 2*** in pixels.
int **GetPixelWidth2**()	Returns the ***Width 2*** in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value.
int **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** in pixels.
int **GetPixelPitch2**()	Returns the ***Pitch 2*** in pixels.
void **SetLogarithmic**(int value)	Sets the ***Logarithmic*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetLogarithmic**()	Returns 1 (**TRUE**) if the ***Logarithmic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleLogarithmic**()	Toggles the ***Logarithmic*** option.

void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggle the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggle the ***Mirror*** option.
void **SetColorMode**(int value)	Sets the ***Color Mode***. See below for a list of constants.
int **GetColorMode**()	Returns the current ***Color Mode***. See below for a list of constants.
void **SetMono**(int value)	Sets the ***Mono*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMono**()	Returns 1 (**TRUE**) if the ***Mono*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMono**()	Toggle the ***Mono*** option.

Color Mode Constants

Constant	Description
int **COLOR_MODE_MONOCHROME**	Sets the color mode to ***Monochrome***.
int **COLOR_MODE_BAND**	Sets the color mode to ***Band***.
int **COLOR_MODE_MATRIX**	Sets the color mode to ***Matrix***.

5.3.7 S2L Level Shape

Functions Provided By S2L Level Shape

This effect uses the following functions:

- This Effect uses the Gradient Dialog. Learn more »Using Gradient Dialog
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetDrop**(int value)	Sets the ***Drop*** *value*.
float **GetDrop**()	Returns the ***Drop*** value.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetLimitMin**(int value)	Sets the ***Limit*** minimum value.
int **GetLimitMin**()	Returns the ***Limit*** minimum value.
void **SetLimitMax**(int value)	Sets the ***Limit*** maximum value.
int **GetLimitMax**()	Returns the ***Limit*** maximum value.
void **SetOuterGlow**(int value)	Sets the ***Outer Glow*** value.
int **GetOuterGlow**()	Returns the ***Outer Glow*** value.
void **SetInnerGlow**(int value)	Sets the ***Inner Glow*** value.
int **GetInnerGlow**()	Returns the ***Inner Glow*** value.
void **SetBorder**(int value)	Sets the ***Border*** value.
int **GetBorder**()	Returns the ***Border*** value.
void **SetLogarithmic**(int value)	Sets the ***Logarithmic*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetLogarithmic**()	Returns 1 (**TRUE**) if the ***Logarithmic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleLogarithmic**()	Toggles the ***Logarithmic*** option.
void **SetMono**(int value)	Sets the ***Mono*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMono**()	Returns 1 (**TRUE**) if the ***Mono*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMono**()	Toggle the ***Mono*** option.
void **SetColorMode**(int value)	Sets the ***Color Mode***. See **below** for a list of constants.
int **GetColorMode**()	Returns the current ***Color Mode***. See **below** for a list of constants.
void **SetDisplayMode**(int value)	Sets the ***Display Mode***. See **below** for a list of constants.
int **GetDisplayMode**()	Returns the current ***Display Mode***. See **below** for a list of constants.
void **SetOuterGlowInterpolationType**(int type)	Sets the ***Outer Glow Interpolation Type***. See **below** for a list of constants.
float **GetOuterGlowInterpolationType**()	Returns the current ***Outer Glow Interpolation Type*** .
void **SetInnerGlowInterpolationType**(int type)	Sets the ***Inner Glow Interpolation Type***. See **below** for a list of constants.
int **GetInnerGlowInterpolationType**()	Returns the current ***Inner Glow Interpolation Type*** .
void **SetProportion**(float value)	Sets the ***Proportion*** of objects. Valid values range from *0.01* to *100*.
float **GetProportion**()	Returns the currently set ***Proportion***.
void **SetDiagonals**(float value)	Sets the ***Diagonals*** of objects. Valid values range from *0.01* to *100*.

float **GetDiagonals**()	Returns the currently set ***Diagonals***.

Color Mode Constants

Constant	Description
int `COLOR_MODE_MONOCHROME`	Sets the color mode to ***Monochrome***.
int `COLOR_MODE_BAND`	Sets the color mode to ***Band***.
int `COLOR_MODE_MATRIX`	Sets the color mode to ***Matrix***.

Display Mode Constants

Constant	Description
int `DISPLAY_MODE_CLIP`	Sets the display mode to ***Clip***.
int `DISPLAY_MODE_FIT`	Sets the display mode to ***Fit***.
int `DISPLAY_MODE_STRETCH`	Sets the display mode to ***Stretch***.

Interpolation Type Constants

Constant	Description
int `INTERPOLATION_TYPE_LINEAR`	Sets the Interpolation Type to ***Linear***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_IN`	Sets the Interpolation Type to ***Ease In Bounce***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_OUT`	Sets the Interpolation Type to ***Ease Out Bounce***.
int `INTERPOLATION_TYPE_EASE_BOUNCE_INOUT`	Sets the Interpolation Type to ***Ease In Out Bounce***.
int `INTERPOLATION_TYPE_EASE_CIRC_IN`	Sets the Interpolation Type to ***Ease In Circular***.
int `INTERPOLATION_TYPE_EASE_CIRC_OUT`	Sets the Interpolation Type to ***Ease Out Circular***.
int `INTERPOLATION_TYPE_EASE_CIRC_INOUT`	Sets the Interpolation Type to ***Ease In Out Circular***.

int **INTERPOLATION_TYPE_EASE_CUBIC_IN**	Sets the Interpolation Type to ***Ease In Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_OUT**	Sets the Interpolation Type to ***Ease Out Cubic***.
int **INTERPOLATION_TYPE_EASE_CUBIC_INOUT**	Sets the Interpolation Type to ***Ease In Out Cubic***.
int **INTERPOLATION_TYPE_EASE_SINE_IN**	Sets the Interpolation Type to ***Ease In Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_OUT**	Sets the Interpolation Type to ***Ease Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_SINE_INOUT**	Sets the Interpolation Type to ***Ease In Out Sinusoidal***.
int **INTERPOLATION_TYPE_EASE_EXPO_IN**	Sets the Interpolation Type to ***Ease In Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_OUT**	Sets the Interpolation Type to ***Ease Out Exponential***.
int **INTERPOLATION_TYPE_EASE_EXPO_INOUT**	Sets the Interpolation Type to ***Ease In Out Exponential*** .

5.3.8 S2L Shapes

Functions Provided By S2L Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.

float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetFade**(float value)	Sets the ***Fade*** *value*, which defines how fast the shapes disappear in BPM. Valid values for *value* range from *1* to *3000*.
float **GetFade**()	Returns the current ***Fade*** in BPM.
void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes, in percent of the matrix size (depending on the shape alignment). The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetPixelSize**(int value)	Sets the ***Size*** *value*, which is the size of the shapes, in pixels. The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSize**()	Returns the current ***Size*** of the shapes in pixels. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetPixelSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in pixels. The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSizeMax**()	Returns the current ***Maximum Size*** of the shapes in pixels.
void **SetSizeDistribution**(int value)	Sets the size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »**here** for a list of constants.
int **GetSizeDistribution**()	Returns the current size ***Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Is the same as **SetSizeDistribution**(**DIST_RND**).
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch1**()	Returns the current ***Pitch 1*** of the shapes in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch1**()	Returns the current ***Pitch 1*** of the shapes in pixels.

void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch2**()	Returns the current ***Pitch 2*** of the shapes in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch2**()	Returns the current ***Pitch 2*** of the shapes in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch3**()	Returns the current ***Pitch 3*** of the shapes in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch3**()	Returns the current ***Pitch 3*** of the shapes in pixels.
void **SetPitch**(float x, float y, float z)	Sets the pitches 1, 2 and 3 of the shapes at once, in percent of the matrix size. Valid values for *x*, *y* and *z* range from *0.01* to *100*.
void **SetPixelPitch**(int x, int y, int z)	Sets the pitches 1, 2 and 3 of the shapes at once, in pixels. Valid values for *x*, *y* and *z* range from *1* to the matrix size.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetSensitivity**(int value)	Sets the ***Sensitivity*** *value*, which controls how much shapes are created by defining a threshold for the incoming audio data, in percent. Valid values for *value* range from *0* to *100*.
int **GetSensitivity**()	Returns the current ***Sensitivity*** in percent.
void **SetShift**(int value)	Sets the ***Shift*** *value*, which controls the mapping between the incoming audio data and the shapes' positions by defining an offset, in percent. Valid values for *value* range from *0* to *100*.
int **GetShift**()	Returns the current ***Shift*** in percent.
void **SetAmplification**(int value)	Sets the ***Amplification*** *value*, which controls how much shapes are created by defining a factor for the incoming audio data, in percent. Valid values for *value* range from *0* to *100*.
int **GetAmplification**()	Returns the current ***Amplification*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shapes' positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shapes' positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shapes' positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shapes' positions or uses the default setting, depending on the current state.

5.3.9 S2L Tubes

Functions Provided By S2L Tubes

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls

Function	Description
void **SetFade**(int value)	Sets the ***Fade*** value.
int **GetFade**()	Returns the ***Fade*** value.
void **SetSize1**(float value)	Sets the ***Size 1*** value
float **GetSize1**()	Returns the ***Size 1*** value.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** value
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetSize2**(float value)	Sets the ***Size 2*** value
float **GetSize2**()	Returns the ***Size 2*** value.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value
float **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** value.
int **GetAlphaMix**()	Returns the ***Alpha Mix*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggles the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggles the ***Mirror*** option.
void **SetTubesVertical**(int value)	Use 1 (**TRUE**) to activate ***vertical Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesVertical**()	Returns 1 (**TRUE**) if ***vertical Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesVertical**()	Toggles ***vertical Tubes***.

void **SetTubesHorizontal**(int value)	Use 1 (**TRUE**) to activate ***horizontal Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesHorizontal**()	Returns 1 (**TRUE**) if ***horizontal Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesHorizontal**()	Toggles ***horizontal Tubes***.
void **SetTubesDepth**(int value)	Use 1 (**TRUE**) to activate ***depth Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesDepth**()	Returns 1 (**TRUE**) if ***depth Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesDepth**()	Toggles ***depth Tubes***.
void **SetSensitivity**(int value)	Sets the ***Sensitivity*** value.
int **SetSensitivity**()	Returns the ***Sensitivity*** value.
void **SetShift**(int value)	Sets the ***Shift*** value.
int **GetShift**()	Returns the ***Shift*** value.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See below for a list of constants.
int **GetGeneratorType**()	Returns the current ***Generator Type***. See below for a list of constants.

Generator Type Constants

Constant	Description
int **GENERATOR_TYPE_EQ**	Sets the generator type to ***EQ***.
int **GENERATOR_TYPE_LEVEL_MONO**	Sets the generator type to ***Level Mono***.
int **GENERATOR_TYPE_LEVEL_STEREO**	Sets the generator type to ***Level Stereo***.

5.3.10 S2L Waveform

Functions Provided By S2L Waveform

This effect uses the following functions:

- This Effect uses the Color Gradient. Learn more »Using Color Gradient
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Directions. Learn more »Using Directions
- This Effect uses Look-At Types. Learn more »Using Look-At Types

Function	Description
void **SetTimeSlot**(float value)	Sets the ***Time Slot*** *value*, which is the length of the displayed wave data, in seconds. This function also automatically sets the ***BPM*** to the corresponding value. Valid values for *value* range from *0.01* to *9999*.
float **GetTimeSlot**()	Returns the current ***Time Slot*** in seconds.
void **SetChannelMode**(int value)	Sets the ***Channel Mode*** *value*, which defines the displayed audio channels. See below for a list of constants.
int **GetChannelMode**()	Returns the current ***Channel Mode***. See below for a list of constants.

Channel Mode Constants

Constant	Description
int `CHANNEL_MODE_STEREO`	Selects ***Stereo*** mode, which displays the left channel and the right channel at the same time.
int `CHANNEL_MODE_MONO`	Selects ***Mono*** mode, which is the average of the left channel and the right channel.
int `CHANNEL_MODE_LEFT`	Selects the ***Left*** channel only.
int `CHANNEL_MODE_RIGHT`	Selects the ***Right*** channel only.

5.3.11 S2L Wavegraph

Functions Provided By S2L Wavegraph

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Directions. Learn more »Using Directions
- This Effect uses Look-At Types. Learn more »Using Look-At Types

Function	Description
void **SetTimeSlot**(float value)	Sets the ***Time Slot*** *value*, which is the length of the displayed wave data, in seconds. This function also automatically sets the ***BPM*** to the corresponding value. Valid values for *value* range from *0.01* to *9999*.
float **GetTimeSlot**()	Returns the current ***Time Slot*** in seconds.
void **SetChannelMode**(int value)	Sets the ***Channel Mode*** *value*, which defines the displayed audio channels. See below for a list of constants.
int **GetChannelMode**()	Returns the current ***Channel Mode***. See below for a list of constants.

Channel Mode Constants

Constant	Description
int `CHANNEL_MODE_STEREO`	Selects ***Stereo*** mode, which displays the left channel and the right channel at the same time.
int `CHANNEL_MODE_MONO`	Selects ***Mono*** mode, which is the average of the left channel and the right channel.
int `CHANNEL_MODE_LEFT`	Selects the ***Left*** channel only.
int `CHANNEL_MODE_RIGHT`	Selects the ***Right*** channel only.

5.4 Music2Light Effects (M2L)

5.4.1 M2L Color Change

Functions Provided By M2L Color Change

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the M2L Color Table. Learn more »Using M2L Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetFade**(int value)	Sets the ***Fade*** value.
int **GetFade**()	Returns the ***Fade*** value.

5.4.2 M2L Color Scroll

Functions Provided By M2L Color Scroll

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the M2L Color Table. Learn more »Using M2L Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Look-At Types. Learn more »Using Look-At Types

Function	Description
void **SetStepWidth**(float value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using percentage values.
float **GetStepWidth**()	Returns the ***Step*** in percentage.
void **SetPixelStepWidth**(int value)	Sets the ***Step*** *value*, which is the number of pixels the effect should scroll per frame using pixel values.
int **GetPixelStepWidth**()	Returns the ***Step*** in pixels.

void **SetColorWidth**(float value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using percentage values.
float **GetColorWidth**()	Returns the ***Color Width***, the width of a stripe, in percentage.
void **SetPixelColorWidth**(int value)	Sets the ***Color Width*** *value*, which means the width of the single stripes using pixel values.
int **GetPixelColorWidth**()	Returns the ***Color Width***, the width of a stripe, in pixels.
void **SetCrossWidth**(float value)	Sets the ***Cross Width*** *value* using percentage values.
float **GetCrossWidth**()	Returns the ***Cross Width*** in percentage.
void **SetPixelCrossWidth**(int value)	Sets the ***Cross Width*** *value* using pixel values.
int **GetPixelCrossWidth**()	Returns the ***Cross Width*** in pixels.
void **SetCrossAxes**(int value)	Sets the ***Cross Axis*** *value*. See **below** for a list of constants.
int **GetCrossAxes**()	Returns the ***Cross Axis***. See **below** for a list of constants.
void **SetSensitivity**(int value)	Sets the ***Sensitivity*** value.
int **GetSensitivity**()	Returns the current ***Sensitivity***.
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from *0.0* to *100.0*.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from *0.0* to *100.0*.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.

Cross Axis Constants

Constant	Description
int `AXIS_ONE`	Sets the direction cross mode to ***Axis 1***.
int `AXIS_TWO`	Sets the direction cross mode to ***Axis 2***.
int `AXES_ONE_AND_TWO`	Sets the direction cross mode to ***Axes 1 And 2***.

5.4.3 M2L Drops

Functions Provided By M2L Drops

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Shape Table. Learn more »Using Shape Table
- This Effect uses Shape Rotation. Learn more »Using Shape Rotation
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetSpeedVariance**(int value)	Sets the ***Speed Variance*** *value*, which means a random value between *0* and *500* percent. This value will multiply with the current BPM value.
int **GetSpeedVariance**()	Returns the current ***Speed Variance***.
void **SetLength**(float value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in percent of the matrix size (depending on the direction). The ***Length*** *value* means the minimum length if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLength**()	Returns the current ***Length*** in percent. The return value means the minimum length if the length distribution mode is not uniform.
void **SetPixelLength**(int value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in pixels. The ***Length*** *value* means the minimum length if the length distribution mode is not uniform.
int **GetPixelLength**()	Returns the current ***Length*** in pixels. The return value means the minimum length if the length distribution mode is not uniform.
void **SetLengthMin**(float value)	Is the same as **SetLength**.
float **GetLengthMin**()	Is the same as **GetLength**.
void **SetPixelLengthMin**(int value)	Is the same as **SetPixelLength**.
int **GetPixelLengthMin**()	Is the same as **GetPixelLength**.
void **SetLengthMax**(float value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in percent of the matrix size (depending on the direction). The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthMax**()	Returns the current ***Maximum Length*** in percent.
void **SetPixelLengthMax**(int value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in pixels. The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform.
int **GetPixelLengthMax**()	Returns the current ***Maximum Length*** in pixels.

void **SetLengthDistribution**(int value)	Sets the ***Length Distribution*** *value*, which defines how the range between the current minimum and the current maximum length is applied to the traces of the shapes. See »**here** for a list of constants.
int **GetLengthDistribution**()	Returns the current ***Length Distribution*** of the traces of the shapes. See »**here** for a list of constants.
void **SeedRandomLength**()	Randomizes the lengths of the traces of the shapes. Same as **SetLengthDistribution**(**DIST_RND**).
void **SetLengthOffset**(float value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthOffset**()	Returns the current ***Length Offset*** in percent.
void **SetPixelLengthOffset**(int value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in pixels.
int **GetPixelLengthOffset**()	Returns the current ***Length Offset*** in pixels.
void **SetRotationOffset**(float value)	Sets the ***Rotation Offset*** *value* between the shapes of the trace. Valid values for *value* range from -180.0 to 180.0.
float **GetRotationOffset**()	Returns the current ***Rotation Offset***.
void **SetPitch1**(float value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch1**()	Returns the current ***Pitch*** for axis one in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch1**()	Returns the current ***Pitch*** for axis one in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch2**()	Returns the current ***Pitch*** for axis two in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch2**()	Returns the current ***Pitch*** for axis two in pixels.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shape's positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shape's positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shape's positions are mirrored, otherwise 0 (**FALSE**).

void **ToggleMirror**()	Mirrors the shape's positions or uses the default setting, depending on the current state.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See below for a list of constants.
int **GetGeneratorType**()	Returns the current ***Generator Type***. See below for a list of constants.
void **SetPeak**(float)	Sets the ***Peak*** value. Valid values range from 0.00 to 100.00.
float **GetPeak**()	Returns the currently set ***Peak*** value.
Function	**Description**
void **SetToneRangeMin**(int index)	Sets the minimum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »Valid parameters (Notes)
int **GetToneRangeMin**()	Returns the currently set minimum ***Tone Range***.
void **SetToneRangeMax**(int index)	Sets the maximum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »Valid parameters (Notes)
int **GetToneRangeMax**()	Returns the currently set maximum ***Tone Range***.

Generator Type Constants

Constant	Description
int **GENERATOR_TYPE_TONE**	Sets the generator type to ***Tone***.
int **GENERATOR_TYPE_INTERVAL**	Sets the generator type to ***Interval***.
int **GENERATOR_TYPE_BASS_TYPE**	Sets the generator type to ***Bass Type***.

5.4.4 M2L Note Flash

Functions provided By M2L Note Flash

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls

Function	Description
void **SetFadeIn**(int fade)	Sets the ***Fade In*** value. This value represents Beats Per Minute (BPM).
int **GetFadeIn**()	Returns the current ***Fade In*** value.
void **SetFadeOut**(int fade)	Sets the ***Fade Out*** value. This value represents Beats Per Minute (BPM).
int **GetFadeOut**()	Returns the current ***Fade Out*** value.
void **SetSensitivity**(int value)	Set the ***Sensitivity*** value.
int **GetSensitivity**()	Returns the ***Sensitivity*** value.
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 of the available values and the appropriate bands.
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 of the available values and the appropriate bands.
int **GetMaxBand**()	Returns the specifier of the current ***Maximum Band***. See below for further details.

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 inimum band and the maximum band. Both use an identifier value between 0 and 10, which describes one of the following frequency bands:

Value	Minimum Band	Maximum Band
0	C-1 (8.2 Hz)	B-1 (15.5 Hz)
1	C0 (16.4 Hz)	B0 (30.9 Hz)
2	C1 (32.7 Hz)	B1 (61.9 Hz)
3	C2 (65.4 Hz)	B2 (123.8 Hz)
4	C3 (130.8 Hz)	B3 (247.5 Hz)
5	C4 (261.6 Hz)	B4 (495 Hz)
6	C5 (523.2 Hz)	B5 (990 Hz)
7	C6 (1047 Hz)	B6 (1980 Hz)
8	C7 (2093 Hz)	B7 (3960 Hz)
9	C8 (4186 Hz)	B8 (7920 Hz)
10	C9 (8448 Hz)	G9 (12672 hz)

Example

The source code below would select C0 as minimum band and B1 as maximum band. Furthermore, the color white is set with a fade-in value of 3000 BPM and a fade-out value of 120 BPM.

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

void InitEffect()
{
 SetColor(WHITE);
 SetFadeIn(3000);
 SetFadeOut(120);
 SetMinBand(1);
 SetMaxBand(2);
}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

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

```
}
```

5.4.5 M2L Shapes

Functions Provided By M2L Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetFade**(float value)	Sets the ***Fade*** *value*, which defines how fast the shapes disappear in BPM. Valid values for *value* range from *1* to *3000*.
float **GetFade**()	Returns the current ***Fade*** in BPM.
void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes, in percent of the matrix size (depending on the shape alignment). The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetPixelSize**(int value)	Sets the ***Size*** *value*, which is the size of the shapes, in pixels. The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).

int **GetPixelSize**()	Returns the current ***Size*** of the shapes in pixels. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetPixelSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in pixels. The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSizeMax**()	Returns the current ***Maximum Size*** of the shapes in pixels.
void **SetSizeDistribution**(int value)	Sets the size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »**here** for a list of constants.
int **GetSizeDistribution**()	Returns the current size ***Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Is the same as **SetSizeDistribution**(**DIST_RND**).
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch1**()	Returns the current ***Pitch 1*** of the shapes in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch1**()	Returns the current ***Pitch 1*** of the shapes in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch2**()	Returns the current ***Pitch 2*** of the shapes in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch2**()	Returns the current ***Pitch 2*** of the shapes in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch3**()	Returns the current ***Pitch 3*** of the shapes in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch3**()	Returns the current ***Pitch 3*** of the shapes in pixels.
void **SetPitch**(float x, float y, float z)	Sets the pitches 1, 2 and 3 of the shapes at once, in percent of the matrix size. Valid values for *x*, *y* and *z* range from *0.01* to *100*.
void **SetPixelPitch**(int x, int y, int z)	Sets the pitches 1, 2 and 3 of the shapes at once, in pixels. Valid values for *x*, *y* and *z* range from *1* to the matrix size.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.

int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetSensitivity**(int value)	Sets the ***Sensitivity*** *value*, which controls how much shapes are created by defining a threshold for the incoming audio data, in percent. Valid values for *value* range from *0* to *100*.
int **GetSensitivity**()	Returns the current ***Sensitivity*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shapes' positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shapes' positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shapes' positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shapes' positions or uses the default setting, depending on the current state.
Function	**Description**
void **SetToneRangeMin**(int index)	Sets the minimum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMin**()	Returns the currently set minimum ***Tone Range***.
void **SetToneRangeMax**(int index)	Sets the maximum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMax**()	Returns the currently set maximum ***Tone Range***.

5.4.6 M2L Single Tone Spectrum

Functions Provided By M2L Single Tone Spectrum

This effect uses the following functions:

- This Effect uses Directions. Learn more »**Using Directions**
- This Effect uses the Gradient Dialog. Learn more »**Using Gradient Dialog**
- This Effect uses Look-At Types. Learn more »**Using Look-At Types**
- This Effect uses Shapes. Learn more »**Using Shapes**

Function	Description

void **SetDrop**(int value)	Sets the ***Drop*** *value*.
float **GetDrop**()	Returns the ***Drop*** value.
void **SetFade**(int value)	Sets the ***Fade*** *value*.
int **GetFade**()	Returns the ***Fade*** value.
void **SetWidth1**(float value)	Sets the ***Width 1*** *value*.
float **GetWidth1**()	Returns the ***Width 1*** value.
void **SetPixelWidth1**(int value)	Sets the ***Width 1*** in pixels.
int **GetPixelWidth1**()	Returns the ***Width 1*** in pixels.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*.
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** in pixels.
int **GetPixelPitch1**()	Returns the ***Pitch 1*** in pixels.
void **SetWidth2**(float value)	Sets the ***Width 2*** value.
float **GetWidth2**()	Returns the ***Width 2*** value.
void **SetPixelWidth2**(int value)	Sets the ***Width 2*** in pixels.
int **GetPixelWidth2**()	Returns the ***Width 2*** in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value.
int **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** in pixels.
int **GetPixelPitch2**()	Returns the ***Pitch 2*** in pixels.
void **SetLogarithmic**(int value)	Sets the ***Logarithmic*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetLogarithmic**()	Returns 1 (**TRUE**) if the ***Logarithmic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleLogarithmic**()	Toggles the ***Logarithmic*** option.
void **SetAmplification**(int value)	Sets the ***Amplification*** value.
int **GetAmplification**()	Returns the ***Amplification*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggle the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggle the ***Mirror*** option.
void **SetColorMode**(int value)	Sets the ***Color Mode***. See below for a list of constants.
int **GetColorMode**()	Returns the current ***Color Mode***. See below for a list of constants.

void **SetDRC**(int value)	Sets the ***DRC*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetDRC**()	Returns 1 (**TRUE**) if the ***DRC*** option is activated, otherwise 0 (**FALSE**) .
void **ToggleDRC**()	Toggle the ***DRC*** option.
void **SetShift**(int value)	Sets the ***Shift*** value.
int **GetShift**()	Returns the ***Shift*** value.
Function	**Description**
void **SetToneRangeMin**(int index)	Sets the minimum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMin**()	Returns the currently set minimum ***Tone Range***.
void **SetToneRangeMax**(int index)	Sets the maximum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMax**()	Returns the currently set maximum ***Tone Range***.

Color Mode Constants

Constant	Description
int **COLOR_MODE_MONOCHROME**	Sets the color mode to ***Monochrome***.
int **COLOR_MODE_BAND**	Sets the color mode to ***Band***.
int **COLOR_MODE_MATRIX**	Sets the color mode to ***Matrix***.

5.4.7 M2L Tubes

Functions Provided By M2L Tubes

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »**Using Color Controls**

Function	Description
void **SetFade**(int value)	Sets the ***Fade*** value.
int **GetFade**()	Returns the ***Fade*** value.
void **SetSize1**(float value)	Sets the ***Size 1*** value
float **GetSize1**()	Returns the ***Size 1*** value.

void **SetPitch1**(float value)	Sets the ***Pitch 1*** value
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetSize2**(float value)	Sets the ***Size 2*** value
float **GetSize2**()	Returns the ***Size 2*** value.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value
float **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** value.
int **GetAlphaMix**()	Returns the ***Alpha Mix*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggles the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggles the ***Mirror*** option.
void **SetTubesVertical**(int value)	Use 1 (**TRUE**) to activate ***vertical Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesVertical**()	Returns 1 (**TRUE**) if ***vertical Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesVertical**()	Toggles ***vertical Tubes***.
void **SetTubesHorizontal**(int value)	Use 1 (**TRUE**) to activate ***horizontal Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesHorizontal**()	Returns 1 (**TRUE**) if ***horizontal Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesHorizontal**()	Toggles ***horizontal Tubes***.
void **SetTubesDepth**(int value)	Use 1 (**TRUE**) to activate ***depth Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesDepth**()	Returns 1 (**TRUE**) if ***depth Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesDepth**()	Toggles ***depth Tubes***.
void **SetGeneratorType**(int value)	Sets the ***Generator Type***. See **below** for a list of constants.
int **GetGeneratorType**()	Returns the current ***Generator Type***. See **below** for a list of constants.
Function	**Description**
void **SetToneRangeMin**(int index)	Sets the minimum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMin**()	Returns the currently set minimum ***Tone Range***.

void **SetToneRangeMax**(int index)	Sets the maximum ***Tone Range*** for the effect with the help of 128 MIDI notes. Valid values for *index* range from *0* to *127*. »**Valid parameters (Notes)**
int **GetToneRangeMax**()	Returns the currently set maximum ***Tone Range***.

Generator Type Constants

Constant	Description
int **GENERATOR_TYPE_TONE**	Sets the generator type to ***Tone***.
int **GENERATOR_TYPE_INTERVAL**	Sets the generator type to ***Interval***.
int **GENERATOR_TYPE_BASS_TYPE**	Sets the generator type to ***Bass Type***.

5.5 Trigger Effects (TRI)

5.5.1 TRI Color Change

Functions Provided By TRI Color Change

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »**Using Color Controls**
- This Effect uses the Color Table. Learn more »**Using Color Table**

Function	Description
void **SetFadeIn**(int value)	Sets the ***Fade In*** value.
int **GetFadeIn**()	Returns the current ***Fade In*** value.
void **SetFadeOut**(int value)	Sets the ***Fade Out*** value.
int **GetFadeOut**()	Returns the current ***Fade Out*** value.
void **SetTriggerAutomatic**(int value)	Use 1 (**TRUE**) to activate the ***Automatic*** option. Use 0 (**FALSE**) to deactivate it.
int **GetTriggerAutomatic**()	Returns 1 (**TRUE**) if the ***Automatic*** option is activated, otherwise 0 (**FALSE**).
void **ToggleTriggerAutomatic**()	Toggles the ***Automatic*** option.
void **SetTriggerMode**(int mode)	Sets the ***Trigger Mode***. Please use a constant **as described below** for *mode*.

int **GetTriggerMode**()	Returns the current ***Trigger Mode***.

Trigger Mode Constants

Constant	Description
int `MODE_FLASH`	The triggered color will be displayed only as long as the trigger port input continues. Otherwise, the base color will be displayed.
int `MODE_LATCH`	The triggered color will be displayed even after the trigger port input has finished. The base color will be ignored.

5.5.2 TRI Drops

Functions Provided By TRI Drops

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Directions
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Shape Table. Learn more »Using Shape Table
- This Effect uses Shape Rotation. Learn more »Using Shape Rotation
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Size Control. Learn more »Using Size Control

Function	Description
void **SetSpeedVariance**(int value)	Sets the ***Speed Variance*** *value*, which means a random value between *0* and *500* percent. This value will multiply with the current BPM value.
int **GetSpeedVariance**()	Returns the current ***Speed Variance***.
void **SetLength**(float value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in percent of the matrix size (depending on the direction). The ***Length*** *value* means the minimum length if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLength**()	Returns the current ***Length*** in percent. The return value means the minimum length if the length distribution mode is not uniform.

void **SetPixelLength**(int value)	Sets the ***Length*** *value*, which defines how long the trace of the shapes is, in pixels. The ***Length*** *value* means the minimum length if the length distribution mode is not uniform.
int **GetPixelLength**()	Returns the current ***Length*** in pixels. The return value means the minimum length if the length distribution mode is not uniform.
void **SetLengthMin**(float value)	Is the same as **SetLength**.
float **GetLengthMin**()	Is the same as **GetLength**.
void **SetPixelLengthMin**(int value)	Is the same as **SetPixelLength**.
int **GetPixelLengthMin**()	Is the same as **GetPixelLength**.
void **SetLengthMax**(float value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in percent of the matrix size (depending on the direction). The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform. Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthMax**()	Returns the current ***Maximum Length*** in percent.
void **SetPixelLengthMax**(int value)	Sets the ***Maximum Length*** *value*, which defines how long the trace of the shapes is at the maximum, in pixels. The ***Maximum Length*** *value* only applies if the length distribution mode is not uniform.
int **GetPixelLengthMax**()	Returns the current ***Maximum Length*** in pixels.
void **SetLengthDistribution**(int value)	Sets the ***Length Distribution*** *value*, which defines how the range between the current minimum and the current maximum length is applied to the traces of the shapes. See »**here** for a list of constants.
int **GetLengthDistribution**()	Returns the current ***Length Distribution*** of the traces of the shapes. See »**here** for a list of constants.
void **SeedRandomLength**()	Randomizes the lengths of the traces of the shapes. Same as **SetLengthDistribution**(**DIST_RND**).
void **SetLengthOffset**(float value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetLengthOffset**()	Returns the current ***Length Offset*** in percent.
void **SetPixelLengthOffset**(int value)	Sets the ***Length Offset*** *value* between the shapes of the trace, in pixels.
int **GetPixelLengthOffset**()	Returns the current ***Length Offset*** in pixels.
void **SetRotationOffset**(float value)	Sets the ***Rotation Offset*** *value* between the shapes of the trace. Valid values for *value* range from -180.0 to 180.0.
float **GetRotationOffset**()	Returns the current ***Rotation Offset***.
void **SetPitch1**(float value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.
float **GetPitch1**()	Returns the current ***Pitch*** for axis one in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch*** *value* for axis one (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch1**()	Returns the current ***Pitch*** for axis one in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in percent of the matrix size (depending on the direction). Valid values for *value* range from *0.0* to *100.0*.

float **GetPitch2**()	Returns the current ***Pitch*** for axis two in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch*** *value* for axis two (depending on the direction), which is the distance of the shapes, in pixels.
int **GetPixelPitch2**()	Returns the current ***Pitch*** for axis two in pixels.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.
void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shape's positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shape's positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shape's positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shape's positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shape's positions or uses the default setting, depending on the current state.
Function	**Description**
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.
void **SetPeak**(float)	Sets the ***Peak*** value. Valid values range from 0.00 to 100.00.
float **GetPeak**()	Returns the currently set ***Peak*** value.

5.5.3 TRI Explosions

Functions Provided By TRI Explosions

This effect uses the following functions:

- This Effect uses Directions. Learn more »Using Direction
- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetExplosionSize**(int size)	Sets the ***Explosion Size*** of a single explosion.
int **GetExplosionSize**()	Returns the current ***Explosion Size*** of a single explosion.
void **SetShapeSize**(int size)	Sets the ***Shape Size*** of single objects.
int **GetShapeSize**()	Returns the current ***Shape Size*** of single objects.
void **SetShapeCount**(int count)	Sets the number of objects within an explosion.
int **GetShapeCount**()	Returns the currently set number of objects of an explosion.
void **SetFadeOut**(int fadeout)	Sets the size of a ***Fade Out*** tail.
int **GetFadeOut**()	Returns the currently set ***Fade Out*** value.
void **SetGravity**(float gravity)	Sets the ***Gravity*** value for the effect.
float **GetGravity**()	Returns the currently set ***Gravity***.
void **Detonate**(int explPosX, int explPosY, int explPosZ, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates an explosion. *explPosX* is the end-coordinate of the explosion in X, *explPosY* is the end-coordinate of the explosion in Y, *explPosZ* is the end-coordinate of the explosion in Z, *ParticleCtn* is the number of objects, *explSize* is the size of the explosion, *explShape* is the type of the explosion, *drawShape* is the shape of the objects, *Col* is the color of the effect, *sparkleCol* is the color of the sparkle part.
void **FireRocket**(int posX, int posY, int posZ, int explPosX, int explPosY, int explPosZ, int ParticleCtn, int explSize, int explShape, int drawShape, color Col, color sparkleCol)	Manually creates a firework. *posX* is the X-coordinate, *posY* is the Y-coordinate, *posZ* is the Z-coordinate, *explPosX* is the end-coordinate of the fireworks explosion in X, *explPosY* is the end-coordinate of the fireworks explosion in Y, *explPosZ* is the end-coordinate of the fireworks explosion in Z, *ParticleCtn* is the number of objects, *explSize* is the size of the fireworks explosion, *explShape* is the type of the fireworks explosion, *drawShape* is the shape of the objects, *Col* is the color of the effect, *sparkleCol* is the color of the sparkle part.
void **SetSparkleColor**(int index, color c)	Sets the color with the specified *index* in the Sparkle Color Table to the given color value. If the index is out of range, nothing happens.
color **GetSparkleColor**(int index)	Returns the color with the specified *index* in the Sparkle Color Table. If the index is out of range, black is returned.
int **GetSparkleColorCount**()	Returns the current number of colors in the Sparkle Color Table.

void **AddSparkleColor**(int index, color c)	Adds another color to the Sparkle Color Table at the specified *index* position. If the index is 0, the new color is added to the first position. If the index is equal to or greater than the current number of colors, the new color is added at the end.
void **RemoveSparkleColor**(int index)	Removes the color at the specified *index* from the Sparkle Color Table. If the given index is out of range, nothing happens.
void **SetSparkleColorMode**(int mode)	Sets the ***Color Mode*** for the Sparkle Color Table. See **here** for details.
int **GetSparkleColorMode**()	Returns the current ***Color Mode*** for the Sparkle Color Table.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
void **SetExplosionMode**(int mode)	Sets the ***Explosion Mode***. See **below** for details.
int **GetExplosionMode**()	Returns which ***Explosion Mode*** is set.
void **SetExplosionShape**(int shape)	Sets the ***Explosion Shape***. See **below** for details.
int **GetExplosionShape**()	Returns which ***Explosion Shape*** is set.

Explosion Mode Constants

This effect uses various explosion modes. The function **SetExplosionMode** can be used to change them. The following values can be used as parameter:

Constant	Description
int `MODE_EXPLOSIONS`	Sets the ***Explosion*** mode.
int `MODE_FIREWORKS`	Sets the ***Fireworks*** mode.

Explosion Shape Constants

This effect uses various explosion shapes. The function **SetExplosionShape** can be used to change them. The following values can be used as parameter:

Constant	Description
int **EXPLOSION_SHAPE_SPHERE**	Sets the sphere type of explosion.
int **EXPLOSION_SHAPE_SPHERE_GLOW**	Sets the glowing sphere type of explosion.
int **EXPLOSION_SHAPE_SPIRAL**	Sets the spiral type of explosion.
int **EXPLOSION_SHAPE_RADIAL**	Sets the radial type of explosion.
int **EXPLOSION_SHAPE_DIAMOND**	Sets the diamond type of explosion.
int **EXPLOSION_SHAPE_STAR**	Sets the star type of explosion.
int **EXPLOSION_SHAPE_RANDOM**	Sets a random type of explosion.

5.5.4 TRI Flash

Functions provided By TRI Flash

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls

Function	Description
void **SetFadeIn**(int fade)	Sets the ***Fade In*** value. This value represents Beats Per Minute (BPM).
int **GetFadeIn**()	Returns the current ***Fade In*** value.
void **SetFadeOut**(int fade)	Sets the ***Fade Out*** value. This value represents Beats Per Minute (BPM).
int **GetFadeOut**()	Returns the current ***Fade Out*** value.

5.5.5 TRI Fluid

Functions Provided By TRI Fluid

This effect uses the following functions:

- This Effect uses Directions. Learn more »**Using Direction**
- This Effect uses the Color Table. Learn more »**Using Color Table**
- This Effect uses the Gradient Dialog. Learn more »**Using Gradient Dialog**
- This Effect uses the BPM Control. Learn more »**Using BPM Control**

Function	Description
void **SetRadius**(int radius)	Sets the ***Radius*** for every new impulse.
int **GetRadius**()	Returns the ***Radius*** for every new impulse.
void **SetVorticity**(int vorticity)	Sets the ***Vorticity*** for the effect.
int **GetVorticity**()	Returns the ***Vorticity***.
void **SetDensity**(int density)	Sets the ***Density*** for every new impulse.
float **GetDensity**()	Returns the ***Density*** for every new impulse.
void **SetVelocity**(int velocity)	Sets the ***Velocity*** for every new impulse.
int **GetVelocity**()	Returns the ***Velocity*** for every new impulse.
void **SetDuration**(int duration)	Sets the ***Duration*** for every new impulse.
int **GetDuration**()	Returns the ***Duration*** for every new impulse.
void **SetFade**(int fade)	Sets the ***Fade*** for the effect.
int **GetFade**()	Returns the ***Fade*** value.
void **SetDeceleration**(int deceleration)	Sets the ***Deceleration*** for the effect.
int **GetDeceleration**()	Returns the ***Deceleration*** value.
void **SetColorMode**(int mode)	Sets the ***Color Mode*** for the effect. See **below** for details.
int **GetColorMode**()	Returns which ***Color Mode*** is set.

Color Mode Constants

This effect uses various color modes. The function **SetColorMode** can be used to change them. The following values can be used as parameter:

Constant	Description
int `MODE_COLOR_TABLE`	Sets the ***Color Table*** mode.
int `MODE_GRADIENT`	Sets the ***Color Gradient*** mode.

5.5.6 TRI Shapes

Functions Provided By TRI Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses Shapes. Learn more »Using Shapes

Function	Description
void **SetBorder**(float value)	Sets the ***Border*** of objects. Valid values range from *0.01* to *100*.
float **GetBorder**()	Returns the currently set ***Border***.
void **SetPixelBorder**(int value)	Sets the ***Border*** of objects in pixels based on the object size.
int **GetPixelBorder**()	Returns the currently set ***Border*** in pixels.
void **SetOuterGlow**(float value)	Sets the ***Outer Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetOuterGlow**()	Returns the currently set ***Outer Glow***.
void **SetPixelOuterGlow**(int value)	Sets the ***Outer Glow*** of objects in pixels based on the object size.
int **GetPixelOuterGlow**()	Returns the currently set ***Outer Glow*** in pixels.
void **SetInnerGlow**(float value)	Sets the ***Inner Glow*** of objects. Valid values range from *0.00* to *100*.
float **GetInnerGlow**()	Returns the currently set ***Inner Glow***.
void **SetPixelInnerGlow**(int value)	Sets the ***Inner Glow*** of objects in pixels based on the object size.
int **GetPixelInnerGlow**()	Returns the currently set ***Inner Glow*** in pixels.
Function	**Description**
void **SetFade**(float value)	Sets the ***Fade*** *value*, which defines how fast the shapes disappear in BPM. Valid values for *value* range from *1* to *3000*.
float **GetFade**()	Returns the current ***Fade*** in BPM.
void **SetSize**(float value)	Sets the ***Size*** *value*, which is the size of the shapes, in percent of the matrix size (depending on the shape alignment). The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSize**()	Returns the current ***Size*** of the shapes in percent. The return value means the minimum shape size if the shape size distribution mode is not uniform.
void **SetPixelSize**(int value)	Sets the ***Size*** *value*, which is the size of the shapes, in pixels. The size *value* means the minimum shape size if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSize**()	Returns the current ***Size*** of the shapes in pixels. The return value means the minimum shape size if the shape size distribution mode is not uniform.

void **SetSizeMax**(float value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in percent of the matrix size (depending on the shape alignment). The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *0.01* to *1000*.
float **GetSizeMax**()	Returns the current ***Maximum Size*** of the shapes in percent.
void **SetPixelSizeMax**(int value)	Sets the ***Maximum Size*** *value*, which is the maximum size of the shapes, in pixels. The maximum size *value* only applies if the shape size distribution mode is not uniform. Valid values for *value* range from *1* to ten times the matrix size (depending on the shape alignment).
int **GetPixelSizeMax**()	Returns the current ***Maximum Size*** of the shapes in pixels.
void **SetSizeDistribution**(int value)	Sets the size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum shape size is applied to the shapes. See »**here** for a list of constants.
int **GetSizeDistribution**()	Returns the current size ***Distribution*** of the shapes. See »**here** for a list of constants.
void **SeedRandomSize**()	Randomizes the sizes of the shapes. Is the same as **SetSizeDistribution**(**DIST_RND**).
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch1**()	Returns the current ***Pitch 1*** of the shapes in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, which is the x-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch1**()	Returns the current ***Pitch 1*** of the shapes in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch2**()	Returns the current ***Pitch 2*** of the shapes in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, which is the y-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch2**()	Returns the current ***Pitch 2*** of the shapes in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in percent of the matrix size. Valid values for *value* range from *0.01* to *100*.
float **GetPitch3**()	Returns the current ***Pitch 3*** of the shapes in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, which is the z-distance of the shapes, in pixels. Valid values for *value* range from *1* to the matrix size.
int **GetPixelPitch3**()	Returns the current ***Pitch 3*** of the shapes in pixels.
void **SetPitch**(float x, float y, float z)	Sets the pitches 1, 2 and 3 of the shapes at once, in percent of the matrix size. Valid values for *x*, *y* and *z* range from *0.01* to *100*.
void **SetPixelPitch**(int x, int y, int z)	Sets the pitches 1, 2 and 3 of the shapes at once, in pixels. Valid values for *x*, *y* and *z* range from *1* to the matrix size.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** *value*, which defines the variance of the shapes' alpha channel, in percent. Valid values for *value* range from *0* to *100*.
int **GetAlphaMix**()	Returns the current ***Alpha Mix*** in percent.

void **SetInvert**(int value)	Use *value* 1 (**TRUE**) to invert the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetInvert**()	Returns 1 (**TRUE**) if the shapes' positions are inverted, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Inverts the shapes' positions or uses the default setting, depending on the current state.
void **SetMirror**(int value)	Use *value* 1 (**TRUE**) to mirror the shapes' positions. Use *value* 0 (**FALSE**) to use the default setting.
int **GetMirror**()	Returns 1 (**TRUE**) if the shapes' positions are mirrored, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Mirrors the shapes' positions or uses the default setting, depending on the current state.

5.5.7 TRI Split Shapes

Functions Provided By TRI Split Shapes

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the Shape Table. Learn more »Using Shape Table
- This Effect uses Look-At Types. Learn more »Using Look-At Types
- This Effect uses the BPM Control. Learn more »Using BPM Control
- This Effect uses the Center Control. Learn more »Using Center Control

Function	Description
void **SetEmissions**(int value)	Sets the ***Emissions*** *value*.
int **GetEmissions**()	Returns the currently set ***Emissions*** value.
void **SetGenerations**(int value)	Sets the ***Generations*** value.
int **GetGenerations**()	Returns the currently set ***Generations*** value.
void **SetVelocity**(float value)	Sets the ***Velocity*** value.
float **GetVelocity**()	Returns the currently set ***Velocity*** value.
void **SetVelocityMin**(float value)	Sets the ***Minimum Velocity*** value.
float **GetVelocityMin**()	Returns the currently set ***Minimum Velocity*** value.
void **SetVelocityMax**(float value)	Sets the ***Maximum Velocity*** value.
float **GetVelocityMax**()	Returns the currently set ***Maximum Velocity*** value.

void **SetVelocityDistributionMode**(int value)	Sets the Velocity ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Velocity is applied. See »here for a list of constants.
int **GetVelocityDistributionMode**()	Returns the current Velocity ***Distribution***. See »here for a list of constants.
void **SetLifetime**(float value)	Sets the ***Lifetime*** *value*.
float **GetLifetime**()	Returns the currently set ***Lifetime*** value.
void **SetLifetimeMin**(float value)	Sets the ***Minimum Lifetime*** value.
float **GetLifetimeMin**()	Returns the currently set ***Minimum Lifetime*** value.
void **SetLifetimeMax**(float value)	Sets the ***Maximum Lifetime*** value.
float **GetLifetimeMax**()	Returns the currently set ***Maximum Lifetime*** value.
void **SetLifetimeDistributionMode**(int value)	Sets the Lifetime ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Lifetime is applied. See »here for a list of constants.
int **GetLifetimeDistributionMode**()	Returns the current Lifetime ***Distribution***. See »here for a list of constants.
void **SetSpread**(float value)	Sets the ***Spread*** *value*.
float **GetSpread**()	Returns the currently set ***Spread*** value.
void **SetSpreadMin**(float value)	Sets the ***Minimum Spread*** value.
float **GetSpreadMin**()	Returns the currently set ***Minimum Spread*** value.
void **SetSpreadMax**(float value)	Sets the ***Maximum Spread*** value.
float **GetSpreadMax**()	Returns the currently set ***Maximum Spread*** value.
void **SetSpreadDistributionMode**(int value)	Sets the Spread ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Spread is applied. See »here for a list of constants.
int **GetSpreadDistributionMode**()	Returns the current Spread ***Distribution***. See »here for a list of constants.
void **SetSize**(float value)	Sets the ***Size*** *value*.
float **GetSize**()	Returns the currently set ***Size*** value.
void **SetSizeMin**(float value)	Sets the ***Minimum Size*** value.
float **GetSizeMin**()	Returns the currently set ***Minimum Size*** value.
void **SetSizeMax**(float value)	Sets the ***Maximum Size*** value.
float **GetSizeMax**()	Returns the currently set ***Maximum Size*** value.
void **SetSizeDistributionMode**(int value)	Sets the Size ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Size is applied. See »here for a list of constants.
int **GetSizeDistributionMode**()	Returns the current Size ***Distribution***. See »here for a list of constants.
void **SetSwirl**(float value)	Sets the ***Swirl*** *value*.
float **GetSwirl**()	Returns the currently set ***Swirl*** value.
void **SetEmissionAngle**(float value)	Sets the ***Angle*** *value*.
float **GetEmissionAngle**()	Returns the currently set ***Angle*** value.

void **SetFadeout**(float value)	Sets the ***Fade*** *value*.
float **GetFadeout**()	Returns the currently set ***Fade*** value.
void **SetAlphaMix**(float value)	Sets the ***Alpha*** *value*.
float **GetAlphaMix**()	Returns the currently set ***Alpha*** value.
void **SetPathAligned**(int value)	Sets the ***PA*** option. Use 1 (**TRUE**) to activate Path Alignment. Use 0 (**FALSE**) to deactivate it.
int **GetPathAligned**()	Returns 1 (**TRUE**) if ***PA*** is activated, otherwise 0 (**FALSE**).
void **SetAnimationDuration**(float value)	Sets the ***Duration*** *value*.
float **GetAnimationDuration**()	Returns the currently set ***Duration*** value.
void **SetColorMode**(int value)	Sets the *color mode*. See **below** for a list of constants.
int **GetColorMode**()	Returns the currently set *color mode*.
void **SetShapeMode**(int value)	Sets the *shape mode*. See **below** for a list of constants.
int **GetShapeMode**()	Returns the currently set *shape mode*.

Color Mode Constants

Constant	Description
int **COLOR_MODE_INHERITING**	Sets the color mode to ***Inheriting***.
int **COLOR_MODE_PER_CYCLE**	Sets the color mode to ***Per Cycle***.
int **COLOR_MODE_PER_GENERATION**	Sets the color mode to ***Per Generation***.
int **COLOR_MODE_ROUND_ROBIN**	Sets the color mode to ***Round Robin***.

Shape Mode Constants

Constant	Description
int **SHAPE_MODE_INHERITING**	Sets the shape mode to ***Inheriting***.
int **SHAPE_MODE_PER_CYCLE**	Sets the shape mode to ***Per Cycle***.
int **SHAPE_MODE_PER_GENERATION**	Sets the shape mode to ***Per Generation***.
int **SHAPE_MODE_ROUND_ROBIN**	Sets the shape mode to ***Round Robin***.

5.5.8 TRI Tubes

Functions Provided By TRI Tubes

This effect uses the following functions:

- This Effect uses the Color Controls. Learn more »Using Color Controls
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetFade**(int value)	Sets the ***Fade*** value.
int **GetFade**()	Returns the ***Fade*** value.
void **SetSize1**(float value)	Sets the ***Size 1*** value
float **GetSize1**()	Returns the ***Size 1*** value.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** value
float **GetPitch1**()	Returns the ***Pitch 1*** value.
void **SetSize2**(float value)	Sets the ***Size 2*** value
float **GetSize2**()	Returns the ***Size 2*** value.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** value
float **GetPitch2**()	Returns the ***Pitch 2*** value.
void **SetAlphaMix**(int value)	Sets the ***Alpha Mix*** value.
int **GetAlphaMix**()	Returns the ***Alpha Mix*** value.
void **SetInvert**(int value)	Sets the ***Invert*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetInvert**()	Returns 1 (**TRUE**) if the ***Invert*** option is activated, otherwise 0 (**FALSE**).
void **ToggleInvert**()	Toggles the ***Invert*** option.
void **SetMirror**(int value)	Sets the ***Mirror*** option. Use 1 (**TRUE**) to activate it. Use 0 (**FALSE**) to deactivate it.
int **GetMirror**()	Returns 1 (**TRUE**) if the ***Mirror*** option is activated, otherwise 0 (**FALSE**).
void **ToggleMirror**()	Toggles the ***Mirror*** option.
void **SetTubesVertical**(int value)	Use 1 (**TRUE**) to activate ***vertical Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesVertical**()	Returns 1 (**TRUE**) if ***vertical Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesVertical**()	Toggles ***vertical Tubes***.
void **SetTubesHorizontal**(int value)	Use 1 (**TRUE**) to activate ***horizontal Tubes***. Use 0 (**FALSE**) to deactivate it.

int **GetTubesHorizontal**()	Returns 1 (**TRUE**) if ***horizontal Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesHorizontal**()	Toggles ***horizontal Tubes***.
void **SetTubesDepth**(int value)	Use 1 (**TRUE**) to activate ***depth Tubes***. Use 0 (**FALSE**) to deactivate it.
int **GetTubesDepth**()	Returns 1 (**TRUE**) if ***depth Tubes*** are activated, otherwise 0 (**FALSE**).
void **ToggleTubesDepth**()	Toggles ***depth Tubes***.

5.5.9 TRI Water

Functions Provided By TRI Water

This effect uses the following functions:

- This Effect uses the Color Table. Learn more »Using Color Table
- This Effect uses the BPM Control. Learn more »Using BPM Control

Function	Description
void **SetSpeedVariance**(int value)	Sets the ***SV*** *value*.
int **GetSpeedVariance**()	Returns the currently set ***SV*** *value*.
void **SetEnergy**(float value)	Sets the ***Energy*** *value*.
float **GetEnergy**()	Returns the currently set ***Energy*** *value*.
void **SetEnergyMax**(float value)	Sets the ***Maximum Energy*** *value*. It only applies if the Energy distribution is not uniform.
float **GetEnergyMax**()	Returns the currently set ***Maximum Energy***.
void **SetEnergyDistribution**(int mode)	Sets the Energy ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Energy is applied. See »here for a list of constants.
int **GetEnergyDistribution**()	Returns the current Energy ***Distribution***. See »here for a list of constants.
void **SetLength**(float)	Sets the ***Length*** *value*.
float **GetLength**()	Returns the currently set ***Length*** *value*.
void **SetLengthMax**(float)	Sets the ***Maximum Length*** *value*. It only applies if the Length distribution is not uniform.
float **GetLengthMax**()	Returns the currently set ***Maximum Length***.

void **SetLengthDistribution**(int)	Sets the Length ***Distribution*** *value*, which defines how the range between the current minimum and the current maximum Length is applied. See »here for a list of constants.
int **GetLengthDistribution**()	Returns the current Length ***Distribution***. See »here for a list of constants.
void **SetPitch1**(float value)	Sets the ***Pitch 1*** *value*, in percent.
float **GetPitch1**()	Returns the ***Pitch 1*** *value*, in percent.
void **SetPixelPitch1**(int value)	Sets the ***Pitch 1*** *value*, in pixels.
int **GetPixelPitch1**()	Returns the ***Pitch 1*** *value*, in pixels.
void **SetPitch2**(float value)	Sets the ***Pitch 2*** *value*, in percent.
float **GetPitch2**()	Returns the ***Pitch 2*** *value*, in percent.
void **SetPixelPitch2**(int value)	Sets the ***Pitch 2*** *value*, in pixels.
int **GetPixelPitch2**()	Returns the ***Pitch 2*** *value*, in pixels.
void **SetPitch3**(float value)	Sets the ***Pitch 3*** *value*, in percent.
float **GetPitch3**()	Returns the ***Pitch 3*** *value*, in percent.
void **SetPixelPitch3**(int value)	Sets the ***Pitch 3*** *value*, in pixels.
int **GetPixelPitch3**()	Returns the ***Pitch 3*** *value*, in pixels.
void **SetPitch**(float, float, float)	Sets the ***Pitch 1***, ***Pitch 2***, and ***Pitch 3*** *values*, in percent.
void **SetPixelPitch**(int, int, int)	Sets the ***Pitch 1***, ***Pitch 2***, and ***Pitch 3*** *values*, in pixels.
void **SetReflect**(int enable)	Disables ***Reflection*** if *enable* is set to 0 (**FALSE**). Otherwise, use 1 (**TRUE**).
int **GetReflect**()	Returns 1 (**TRUE**) if ***Reflection*** is active, otherwise 0 (**FALSE**) is returned.
void **ToggleReflect**()	Enables ***Reflection*** or uses the default setting, depending on the current state.

Function	Description
void **SetDisplacement**(float value)	Sets the ***Displacement*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacement**()	Returns the current ***Displacement***.
void **SetDisplacementSpeed**(float value)	Sets the ***Displacement Speed*** *value*. Valid values range from 0.0 to 100.0.
float **GetDisplacementSpeed**()	Returns the current ***Displacement Speed***.
void **SetDisplacementDistribution**(int mode)	Sets the ***Displacement Distribution***. See »here for a list of constants (not all apply).
int **GetDisplacementDistribution**()	Returns the current ***Displacement Distribution***.

MADRIX LIGHTING CONTROL

//PART F

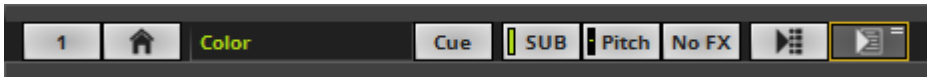
Storage Place Macro

6 Storage Place Macro

6.1 Overview (Storage Place Macro)

Introduction

- Storage Place Macros affect effects placed in Storage Places.
- That means the complete effect including all of its Layers is affected.
- 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*.



Editor - Controls the corresponding Macro Editor.

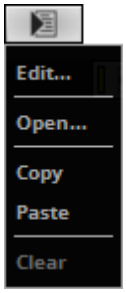
Macro - Allows you to configure and manage a Macro. At the same time this means that no Macro is running or included.

Deactivated [Macro Included] - A macro is included/inserted in the Macro Editor, but the macro is currently not running.

Left Mouse Click - Click once to activate the Macro.

Activated - A macro has been compiled and is currently running.

Left Mouse Click - Click once to deactivate the Macro.



Right Mouse Click - Opens the context menu.

Edit... - Opens the Macro Editor to write, edit, include, and compile Macros.

Open... - Allows you to load a Macro from an external file [of the file type *.mms and *.mcm]. Once loaded, the Macro will automatically be activated.

Copy - Allows you to copy the Macro to the clipboard of the computer.

Paste - Allows you to paste the Macro from the clipboard into the currently selected Macro Editor. If the copied Macro is running, the new Macro will be automatically activated as well. If the copied Macro is deactivated, the new Macro will be automatically deactivated as well.

Clear - Deletes all content of the Macro Editor and thereby deactivates and erases any Macro.

Functions Called By MADRIX

Overview

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 information for the developer of the script.

InitEffect

(Automatically included in a new macro)

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 array 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 arrays, set the speed of an effect, or whatever is necessary to (re)start the macro.

PreRenderEffect

(Automatically included in a new macro)

PreRenderEffect is called before the Storage Place is going to be rendered. Changes done here affect the current frame, but may be overwritten by the Storage Place itself.

PostRenderEffect

(Automatically included in a new macro)

This function is called after the Storage Place has been rendered. Here, the result of the Storage Place can be manipulated. You could use a filter on it, for example.

```
void InitEffect()
{

}
void PreRenderEffect()
{
}
void PostRenderEffect()
{
```

```
        color c = {random(0, 255), random(0, 255)};
        Clear(c);
    }
```

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

Note: The matrix, which the macro manipulates, is the same matrix that the Storage Place uses to calculate itself. The Storage Place may rely on the output being the input for the next frame with undefined behavior.

Also note: Mapping operations done in *PostRenderEffect* will affect the next frame, but not the current one. To control the current frame, please use *PreRenderEffect*.

MartrixSizeChanged

(Automatically included in a new macro)

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 (Storage Place Macro)

Specific Resources

- »Functions called by MADRIX
- Storage Place Macro: Available Functions
- Constants

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

- For non-specific functions, see »List of Functions (Alphabetical Order)

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 Global Macro.

- Each Storage Place can have 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** or **NOPAUSE**. »Description
void **TogglePause**()	Toggles the status of the ***Pause*** function.
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 set in quotation marks. E.g. SetDescription("Storage Place 1");
int **GetSubMaster**()	Retrieves the current value of the ***Submaster***.
void **SetSubMaster**(int value)	Sets the ***Submaster*** value. Valid values range from *0* to *255*.
void **SetSpeedPitch**(float value)	Sets the ***Pitch*** *value*, which defines an additional multiplicative factor for the speed of the Storage. Valid values for *value* range from *-10* to *10*. Setting *value* lower than 0 lets most effects run backwards.
float **GetSpeedPitch**()	Returns the current ***Pitch***.
Layer Functions (For Single Layers)	
int **GetLayerCount**()	Retrieves the currently used number of Layers.
int **LayerGetBlind**(int number)	Retrieves if the specified Layer is used in ***Blind Mode***. »Description Layer indexing (*number*) starts with 0.
void **LayerSetBlind**(int number, int value)	Sets ***Blind Mode*** for a specified Layer. »Description Layer indexing (*number*) starts with 0.
void **LayerToggleBlind**(int number)	Toggles ***Blind Mode*** for a specified Layer. »Description Layer indexing (*number*) starts with 0.
int **LayerGetSolo**(int number)	Retrieves if the specified Layer is used in ***Solo Mode***. »Description Layer indexing (*number*) starts with 0.
void **LayerSetSolo**(int number, int value)	Sets ***Solo Mode*** for a specified Layer. »Description Layer indexing (*number*) starts with 0.
void **LayerToggleSolo**(int number)	Toggles ***Solo Mode*** for a specified Layer. »Description Layer indexing (*number*) starts with 0.
int **LayerGetSubMaster**(int number)	Retrieves the value of the ***Submaster*** of a specified Layer. Layer indexing (*number*) starts with 0.
void **LayerSetSubMaster**(int number, int value)	Sets the value of the ***Submaster*** of a specified Layer. Layer indexing (*number*) starts with 0.
int **LayerGetOpacity**(int number)	Retrieves the ***Opacity*** of the specified Layer. Layer indexing (*number*) starts with 0.
void **LayerSetOpacity**(int number, int value)	Sets the ***Opacity*** *value* of the specified Layer. Layer indexing (*number*) starts with 0.
int **LayerGetMixMode**(int number)	Retrieves the currently used ***Mix Mode*** of the specified Layer. Valid values are described »here. Layer indexing (*number*) starts with 0.
void **LayerSetMixMode**(int number, int value)	Sets the ***Mix Mode*** for the specified Layer. Valid values are described » here. Layer indexing (*number*) starts with 0.
int **LayerGetLink**(int number)	Retrieves if ***Link Mode*** is enabled for the specified Layer. Layer indexing (*number*) starts with 0.

void **LayerSetLink**(int number, int value)	Enables ***Link Mode*** for the specified Layer. Layer indexing (*number*) starts with 0.
void **LayerToggleLink**(int number)	Toggles ***Link Mode*** for the specified Layer. Layer indexing (*number*) starts with 0.
int **LayerGetFilter**(int number)	Returns which ***Filter (FX)*** is applied to the specified Layer. Layer indexing (*number*) starts with 0.
void **LayerSetFilter**(int number, int value)	Applies a ***Filter (FX)*** to the specified Layer. Layer indexing (*number*) starts with 0. Valid values for *filter* are »**Filters**
int **LayerGetStep**(int number)	Retrieves if the specified Layer uses stepped rendering. »**Description** Layer indexing (*number*) starts with 0.
void **LayerSetStep**(int number, int value)	Sets the stepped rendering mode for the specified Layer. »**Description** Layer indexing (*number*) starts with 0.
void **LayerToggleStep**(int number)	Toggles the stepped rendering mode for the specified Layer. » **Description** Layer indexing (*number*) starts with 0.
int **LayerGetAsync**(int number)	Retrieves if the specified Layer uses asynchronous rendering. » **Description** Layer indexing (*number*) starts with 0.
void **LayerSetAsync**(int number, int value)	Sets the asynchronous rendering mode for the specified Layer. » **Description** Layer indexing (*number*) starts with 0.
void **LayerToggleAsync**(int number)	Toggles the asynchronous rendering mode for the specified Layer. » **Description** Layer indexing (*number*) starts with 0.
float **LayerGetFrameId**(int number)	Returns the ***Frame ID*** of the current frame for the specified Layer. » **Description** Layer indexing (*number*) starts with 0.
void **LayerSetFrameId**(int number, float value)	Sets the ***Frame ID*** of the current frame for the specified Layer. » **Description** Layer indexing (*number*) starts with 0.
float **LayerGetFrameSteps**(int number)	Returns the number of frames which are between this and the last call of the specified Layer. »**Description** Layer indexing (*number*) starts with 0.
float **LayerGetFrameCount**(int number)	Retrieves the number of frames used by the specified Layer. Layer indexing (*number*) starts with 0.
Mapping Functions (For Single Layers)	
int **LayerMapDlgIsMapped**(int number)	Retrieves if the specified Layer is mapped. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetMapVector**(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 (*number*) starts with 0. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.
void **LayerMapDlgSetMapVector3D**(int number, float x, float y, float z, float w, float h, float d)	Maps the effect of the specified Layer to a certain area of the matrix in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.
void **LayerMapDlgSetMapPixel**(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 pixel values. Layer indexing (*number*) starts with 0. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.

void **LayerMapDlgSetMapPixel3D**(int number, int x, int y, int z, int w, int h, int d);	Maps the effect of the specified Layer to a certain area of the matrix in 3D using pixel values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.
void **LayerMapDlgGetMapVector**(int number, float map[])	Retrieves the map settings for a specified Layer using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)
void **LayerMapDlgGetMapVector3D**(int number, float map[])	Retrieves the map settings for a specified Layer in 3D using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)
void **LayerMapDlgGetMapPixel**(int number, int map[])	Retrieves the map settings for a specified Layer using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = width (Size X) ▪ map[3] = height (Size Y)

void **LayerMapDlgGetMapPixel3D**(int number, int map[])	Retrieves the map settings for a specified Layer in 3D using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*map[]*). ▪ map[0] = X-coordinate (Position X) ▪ map[1] = Y-coordinate (Position Y) ▪ map[2] = Z-coordinate (Position Z) ▪ map[3] = width (Size X) ▪ map[4] = height (Size Y) ▪ map[5] = depth (Size Z)
void **LayerMapDlgSetTileVector**(int number, float x, float y, float w, float h)	Tiles the effect of the specified Layer to a certain area using relative values. Layer indexing (*number*) starts with 0. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.
void **LayerMapDlgSetTileVector3D**(int number, float x, float y, float z, float w, float h, float d)	Tiles the effect of the specified Layer in a certain area in 3D using relative values. Layer indexing (*number*) starts with 0. *x, y, and z* describe the coordinates. *w, h, and d* describe width, height, and depth.
void **LayerMapDlgSetTilePixel**(int number, int x, int y, int w, int h)	Tiles the effect of the specified Layer in a certain area using pixel values. Layer indexing (*number*) starts with 0. *x* and *y* describe the coordinates. *w* and *h* describe width and height, respectively.
void **LayerMapDlgSetTilePixel3D**(int number, int x, int y, int z, int w, int h, int d)	Tiles the effect of the specified Layer in a certain area using pixel values. Layer indexing (*number*) starts with 0. *x, y and z* describe the coordinates. *w, h, and d* describe width, height, and depth.
void **LayerMapDlgGetTileVector**(int number, float tile[])	Retrieves the ***Tiling*** settings for a specified Layer using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)

void **LayerMapDlgGetTileVector3D**(int number, float tile[])	Retrieves the ***Tiling*** settings for a specified Layer in 3D using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)
void **LayerMapDlgGetTilePixel**(int number, int tile[])	Retrieves the ***Tiling*** settings for a specified Layer using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = width (Size X) ▪ tile[3] = height (Size Y)
void **LayerMapDlgGetTilePixel3D**(int number, int tile[])	Retrieves the ***Tiling*** settings for a specified Layer in 3D using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*tile[]*). ▪ tile[0] = X-coordinate (Position X) ▪ tile[1] = Y-coordinate (Position Y) ▪ tile[2] = Z-coordinate (Position Z) ▪ tile[3] = width (Size X) ▪ tile[4] = height (Size Y) ▪ tile[5] = depth (Size Z)
void **LayerMapDlgSetMapMode**(int number, int mode)	Sets the ***Map Mirror Mode*** of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of constants.
void **LayerMapDlgSetTileMode**(int number, int mode)	Sets the ***Tile Mode*** of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of Tile Mode constants.
int **LayerMapDlgGetMapMode**(int number)	Retrieves the currently used ***Map Mirror Mode*** of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of constants.
int **LayerMapDlgGetTileMode**(int number)	Retrieves the currently used ***Tile Mode*** of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of Tile Mode constants.

void **LayerMapDlgSetTileOffsetVector**(int number, float x, float y)	Sets the ***Tiling Offset*** of the specified Layer using relative values. Layer indexing (*number*) starts with 0. *x* describes Offset X, while *y* represents Offset Y.
void **LayerMapDlgSetTileOffsetVector3D**(int number, float x, float y, float z)	Sets the ***Tiling Offset*** of the specified Layer in 3D using relative values. Layer indexing (*number*) starts with 0. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.
void **LayerMapDlgSetTileOffsetPixel**(int number, int x, int y)	Sets the ***Tiling Offset*** of the specified Layer using pixel values. Layer indexing (*number*) starts with 0. *x* describes Offset X, while *y* represents Offset Y.
void **LayerMapDlgSetTileOffsetPixel3D**(int number, int x, int y, int z)	Sets the ***Tiling Offset*** of the specified Layer in 3D using pixel values. Layer indexing (*number*) starts with 0. *x* describes Offset X, *y* describes Offset Y, while *z* represents Offset Z.
void **LayerMapDlgGetTileOffsetVector**(int number, float offset[])	Retrieves the ***Tiling Offset*** of the specified Layer using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)
void **LayerMapDlgGetTileOffsetVector3D**(int number, float offset[])	Retrieves the ***Tiling Offset*** of the specified Layer in 3D using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)
void **LayerMapDlgGetTileOffsetPixel**(int number, int offset[])	Retrieves the ***Tiling Offset*** of the specified Layer using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y)
void **LayerMapDlgGetTileOffsetPixel3D**(int number, int offset[])	Retrieves the ***Tiling Offset*** of the specified Layer in 3D using pixel values. Layer indexing (*number*) starts with 0. The values are saved in an array (*offset[]*). ▪ offset[0] = X-coordinate (Offset X) ▪ offset[1] = Y-coordinate (Offset Y) ▪ offset[2] = Z-coordinate (Offset Z)
void **LayerMapDlgSetRotationXVector**(int number, float value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationXDegree**(int number int value)	Sets the ***Rotation*** *value* for the X-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
float **LayerMapDlgGetRotationXVector**(int number)	Retrieves the ***Rotation*** for the X-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.

int **LayerMapDlgGetRotationXDegree**(int number)	Retrieves the ***Rotation*** for the X-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationXMode**(int number, int mode)	Sets the ***Rotation*** mode for the X-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
int **LayerMapDlgGetRotationXMode**(int number)	Retrieves the ***Rotation*** mode for the X-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
void **LayerMapDlgSetRotationYVector**(int number, float value)	Sets the ***Rotation*** *value* for the Y-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationYDegree**(int number int value)	Sets the ***Rotation*** *value* for the Y-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
float **LayerMapDlgGetRotationYVector**(int number)	Retrieves the ***Rotation*** for the Y-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
int **LayerMapDlgGetRotationYDegree**(int number)	Retrieves the ***Rotation*** for the Y-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationYMode**(int number, int mode)	Sets the ***Rotation*** mode for the Y-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
int **LayerMapDlgGetRotationYMode**(int number)	Retrieves the ***Rotation*** mode for the Y-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below**for a list of rotation mode constants.
void **LayerMapDlgSetRotationZVector**(int number, float value)	Sets the ***Rotation*** value for the Z-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationZDegree**(int number int value)	Sets the ***Rotation*** value for the Z-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
float **LayerMapDlgGetRotationZVector**(int number)	Retrieves the ***Rotation*** value for the Z-axis of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
int **LayerMapDlgGetRotationZDegree**(int number)	Retrieves the ***Rotation*** value for the Z-axis of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationZMode**(int number, int mode)	Sets the ***Rotation*** mode for the Z-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
int **LayerMapDlgGetRotationZMode**(int number)	Retrieves the ***Rotation*** mode for the Z-axis of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
void **LayerMapDlgSetRotationVector**(int number, float x, float y, float z)	Sets the ***Rotation*** values for axes *x, y, and z* of the specified Layer using relative values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgSetRotationDegree**(int number, int x, int y, int z)	Sets the ***Rotation*** values for axes *x, y, and z* of the specified Layer using degree values. Layer indexing (*number*) starts with 0.
void **LayerMapDlgGetRotationVector**(int number, float rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the specified Layer using relative values. Layer indexing (*number*) starts with 0. The values are saved in an array (*rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)

void **LayerMapDlgGetRotationDegree**(int number, int rot[])	Retrieves the ***Rotation*** values for axes *x, y, and z* of the specified Layer using degree values. Layer indexing (*number*) starts with 0. The values are saved in an array *(rot[]*). ▪ rot[0] = X-coordinate (X-Axis) ▪ rot[1] = Y-coordinate (Y-Axis) ▪ rot[2] = Z-coordinate (Z-Axis)
void **LayerMapDlgSetRotationMode**(int number, int x, int y, int z)	Sets the ***Rotation*** mode for axes *x, y, and z* of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of rotation mode constants.
void **LayerMapDlgGetRotationMode**(int number, int mode[])	Retrieves the ***Rotation*** mode for axes *x, y, and z* of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list tile rotation mode constants. The values are saved in an array (*mode[]*). ▪ mode[0] = X-coordinate (X-Axis) ▪ mode[1] = Y-coordinate (Y-Axis) ▪ mode[2] = Z-coordinate (Z-Axis)
void **LayerMapDlgSetAntiAliasing**(int number, int aaLevel)	Sets the ***Anti-Aliasing*** mode of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of anti-aliasing mode constants.
int **LayerMapDlgGetAntiAliasing**(int number)	Retrieves the ***Anti-Aliasing*** mode of the specified Layer. Layer indexing (*number*) starts with 0. See **below** for a list of anti-aliasing mode constants.

Remarks

All functions affecting the map and tile options, such as LayerMapDlgSetMapVector(), will be applied one frame later!

Constants

Constant	Description
int **PAUSE**	Activates a Storage Place to be paused. To be used with **SetPause**
int **NOPAUSE**	Deactivates the pausing mode of a Storage Place. To be used with **SetPause**

Map Mirror Mode Constants

Constant	Description
int MAP_MIRROR_NONE	Sets no Mirror 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 horizontally and vertically.
int MAP_MIRROR_D	Mirrors the content of the matrix in the depth.
int MAP_MIRROR_HD	Mirrors the content of the matrix horizontally and in the depth.
int MAP_MIRROR_VD	Mirrors the content of the matrix vertically and in the depth.
int MAP_MIRROR_HVD	Mirrors the content of the matrix horizontally, vertically, and in the depth.

Map Tile Mode Constants

Constant	Description
int MAP_TILE_NONE	Sets no Tile Mode.
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 horizontally and vertically.
int MAP_TILE_MIRROR_D	Mirrors tiles in the depth.
int MAP_TILE_MIRROR_HD	Mirrors tiles horizontally and in the depth.
int MAP_TILE_MIRROR_VD	Mirrors tiles vertically and in the depth.
int MAP_TILE_MIRROR_HVD	Mirrors tiles horizontally, vertically, and in the depth.

Map Rotation Mode Constants

Constant	Description
int MAP_ROTATION_FIXED	The rotation value is fixed and not animated.

Constant	Description
int `MAP_ROTATION_LOOP`	The rotation value is used for a looped animation.

Map Anti-Aliasing Mode Constants

Constant	Description
int `MAP_AA_NONE`	Sets no anti-aliasing.
int `MAP_AA_2X`	Sets simple anti-aliasing (requires some performance).
int `MAP_AA_4X`	Sets complex anti-aliasing (requires a lot performance).

MADRIX LIGHTING CONTROL

//PART G

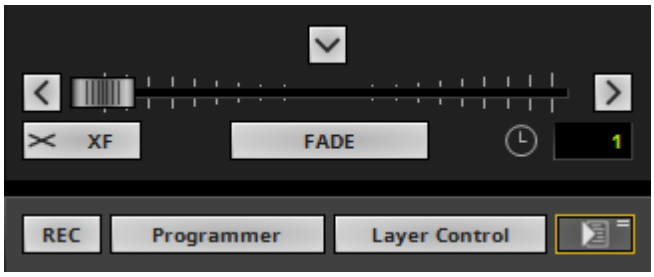
Global Macro

7 Global Macro

7.1 Overview (Global Macro)

Introduction

- Global macros can be used to manipulate the final output of MADRIX. This also includes global features, such as the Cue List, Master, and Audio Input Level among others.
- Global macros are stored together with a MADRIX Setup file.
- 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*.



Editor - Controls the corresponding Macro Editor.

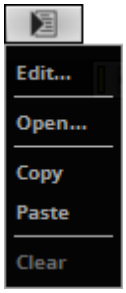
Macro - Allows you to configure and manage a Macro. At the same time this means that no Macro is running or included.

Deactivated [Macro Included] - A macro is included/inserted in the Macro Editor, but the macro is currently not running.

Left Mouse Click - Click once to activate the Macro.

Activated - A macro has been compiled and is currently running.

Left Mouse Click - Click once to deactivate the Macro.



Right Mouse Click - Opens the context menu.

Edit... - Opens the Macro Editor to write, edit, include, and compile Macros.

Open... - Allows you to load a Macro from an external file [of the file type *.mms and *.mcm]. Once loaded, the Macro will automatically be activated.

Copy - Allows you to copy the Macro to the clipboard of the computer.

Paste - Allows you to paste the Macro from the clipboard into the currently selected Macro Editor. If the copied Macro is running, the new Macro will be automatically activated as well. If the copied Macro is deactivated, the new Macro will be automatically deactivated as well.

Clear - Deletes all content of the Macro Editor and thereby deactivates and erases any Macro.

Functions Called By MADRIX

Overview

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 information for the developer of the script.

InitEffect

(Automatically included in a new macro)

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 array 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 arrays, set the speed of an effect, or whatever is necessary to (re)start the macro.

PreRenderEffect

(Automatically included in a new macro)

PreRenderEffect is called before the Main Output is going to be rendered. Changes done here affect the current frame, but may be overwritten by the Main Output itself.

PostRenderEffect

(Automatically included in a new macro)

This function is called after the Main Output has been rendered. Here, the result of the Main Output can be manipulated. You could use a filter on it, for example.

```
void InitEffect()
{

}
void PreRenderEffect()
{
}
void PostRenderEffect()
{
```

```
        color c = {random(0, 255), random(0, 255)};
        Clear(c);
    }
```

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

Note: The matrix, which the macro manipulates, is the same matrix that the Main Output uses to calculate itself. The Main Output may rely on the output being the input for the next frame with undefined behavior.

MartrixSizeChanged

(Automatically included in a new macro)

MatrixSizeChanged is called after the size of the matrix has been changed. This may be due to a change to the matrix settings.

Standard Outline

When you open the Global 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 (Global Macro)

Specific Resources

- »Functions called by MADRIX
- Global Macro: Available Functions
- Constants

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

- For non-specific functions, see »List of Functions (Alphabetical Order)
- The Global Macro offers special functions to reflect the features available to the user in the user interface of MADRIX, like the Freeze function or the Cue List.

Functions Provided By The Global 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
General	

void **SetFade**(int mode)	Initiates an automatic fade of the Crossfader in the set Fade Time. Valid constants for *mode* are **AUTOMATIC**, **TO_LEFT**, **TO_CENTER**, **TO_RIGHT**.
int **GetFadeType**()	Returns the current ***Fade Type***. »Example
void **SetFadeType**(int fadetype)	Sets the ***Fade Type***. See below for valid constants for *fadetype*. » Example
float **GetFadeTime**()	Returns the ***Fade Time*** in seconds that is used for an autofade between Left and Right. »Example
void **SetFadeTime**(float fadetime)	Sets the ***Fade Time*** in seconds. Valid values range from *0* to *3600*. » Example
int **GetFadeValue**()	Returns the value/position of the crossfader. »Example
void **SetFadeValue**(int value)	Sets the value for the position of the crossfader. Valid values range from *0* to *255*. »Example
int **GetFreeze**()	Returns the status of the ***Freeze*** function. »Example
void **SetFreeze**(int freeze/unfreeze)	Sets the ***Freeze*** function to be activated or deactivated. »Example
int **GetMasterFader**()	Returns the value/position of the ***Master***, which determines the brightness of the Main Output. »Example
void **SetMasterFader**(int value)	Sets the value of the ***Master***. Valid values range from *0* to *255*. » Example
int **GetAudioInputFader**()	Returns the value/position of the ***Audio Input Level***. »Example
void **SetAudioInputFader**(int value)	Sets the value of the ***Audio Input Level***. Valid values range from *0* to *255*. »Example
int **GetAudioInputMute**()	Returns 1 (**TRUE**) if the audio input is muted, otherwise 0 (**FALSE**).
void **SetAudioInputMute**(int value)	Use 1 (**TRUE**) to mute the audio input, otherwise 0 (**FALSE**).
int **GetAudioOutputFader**()	Returns the value/position of the ***Audio Output Level***.
void **SetAudioOutputFader**(int value)	Sets the value of the ***Audio Output Level***. Valid values range from *0* to *255*.
int **GetAudioOutputMute**()	Returns 1 (**TRUE**) if the audio output is muted, otherwise 0 (**FALSE**).
void **SetAudioOutputMute**(int value)	Use 1 (**TRUE**) to mute the audio output, otherwise 0 (**FALSE**).
int **GetStorageSpeedMaster**(int storage)	Returns the value of the ***Speed Master*** for Deck A or Deck B. Valid values are **LEFT, RIGHT**, and a speed range from *-10.0* to *10.0* for the return value. »Example
void **SetStorageSpeedMaster**(int storage, float speed)	Sets the value of the ***Speed Master*** for Deck A or Deck B. Valid values are **LEFT, RIGHT**. »Example
int **GetStoragePause**(int storage)	Returns the status of the ***Pause*** function of Deck A or Deck B. Valid values are **LEFT, RIGHT**. »Example
void **SetStoragePause**(int storage, int mode)	Sets the ***Pause*** mode to be activated or deactivated for Deck A or Deck B. Valid values are **LEFT, RIGHT** for *storage* as well as **PAUSE, NOPAUSE**. »Example
void **GetStoragePlace**(int storage)	Returns the currently selected ***Storage Place*** of Deck A or Deck B with a value between *0* and *255* (Indexing starts with 0). Valid values are for *storage* **LEFT, RIGHT**. »Example

void **SetStoragePlace**(int storage, int place, int mode)	Set a specific ***Storage Place*** to be activated/selected in Deck A or Deck B. Valid values are **LEFT, RIGHT** for *storage*, and a *place* between *0* and *255* (Indexing starts with 0). The third value is optional and valid parameters are: **WITH_AUTOFADE, WITHOUT_AUTOFADE**. Activated by default is **WITH_AUTOFADE**. (When the currently selected Storage Place can be seen in the output window and a new Storage Place without automatic fade is set with this function, automatic fade will still be used (**WITHOUT_AUTOFADE** will be overwritten) to provide a smooth fade on the output.) »**Example**
void **SetStorageSubMaster**(int storage, int value)	Sets the ***Submaster*** of Deck A or Deck B. Valid values for *storage* are **LEFT, RIGHT**, and a value range from *0* to *255* for *value*. »**Example**
int **GetStorageSubMaster**(int storage)	Returns the value of the ***Submaster*** of Deck A or Deck B. Valid values are **LEFT, RIGHT**. »**Example**
void **SetFilter**(int side, int filter)	Applies a ***Filter Effect (FX)*** to the Main Output, Deck A, or Deck B. Valid values for *side* are **LEFT, RIGHT, MAIN**. Valid values for *filter* are » **Filters**
int **GetFilter**(int side)	Returns which ***Filter Effect (FX)*** is applied to the Main Output, Deck A, or Deck B.
void **SetFilterColor**(int side, color c)	Sets the color that will be used for the ***Color Filter*** of the Main Output, Deck A, or Deck B. Valid values for *side* are **LEFT, RIGHT, MAIN** »**List Of Structures** »**List of Global Constants**
color **GetFilterColor**(int side)	Retrieves the color that is currently set for the ***Color Filter*** of the Main Output, Deck A, or Deck B. »**List Of Structures** »**List of Global Constants**
void **SetStoragePlaceSubMaster**(int storage, int value)	Sets the ***Submaster*** for the currently selected Storage Place on Deck A or Deck B. Valid values for *storage* are **LEFT, RIGHT**, and a value range from *0* to *255* for the *value*.
int **GetStoragePlaceSubMaster**(int storage)	Returns the value of the ***Submaster*** of the currently selected Storage Place on Deck A or Deck B. Valid values are **LEFT, RIGHT**.
int **GetStoragePlaceFullState**(int storage, int place)	Returns *1* if the standard settings of a Storage Place of either Deck A or Deck B have been modified, i.e. if an effect has been set up on a Storage Place. Valid values for *storage* are **LEFT, RIGHT**. *place* describes the Storage Place between *0* and *255* (Indexing starts with 0). Returns *0* if the specified Storage Place remains unchanged.
void **SetStoragePlaceFilter**(int storage, int filter)	Applies a ***Filter Effect (FX)*** to the currently selected Storage Place. Valid values for *storage* are **LEFT, RIGHT**. Valid values for *filter* are » **Filters**
int **GetStoragePlaceFilter**(int index)	Returns which ***Filter Effect (FX)*** is applied to the currently selected Storage Place.
void **SetBlackout**(int mode)	Activates or deactivates a ***Blackout*** of the Main Output. Valid values for mode are **BLACKOUT_OFF** or **BLACKOUT_ON**.
int **GetBlackout**()	Returns if the ***Blackout*** is active or not.
int **ImportFixtureGroupController**(string file)	Automatically loads a ***MADRIX Group Control file***. Valid values for *file* are *.mgcz file names and the specific location on the harddisk; for example: "C:/Dimmed50Percent.mgcz". The function returns *1* (loaded successfully) or *0* (failed to load). It is recommended to only use this function in InitEffect().

int **ImportPatch**(string file)	Automatically loads a ***MADRIX Patch file***. Valid values for *file* are *.mpz file names and the specific location on the harddisk; for example: "C:/Patch01.mpz". The function returns *1* (loaded successfully) or *0* (failed to load). It is recommended to only use this function in InitEffect() - Please remove InitEffect() from MatrixSizeChanged() in this case! If used in other parts of the Macro, MatrixSizeChanged() should be included in InitEffect().
void **ImportStorage**(int storage, string file)	Automatically loads a complete ***MADRIX Storage file***. Valid values for *storage* are **LEFT, RIGHT**. Valid values for *file* are *.mstz file names and the specific location on the harddisk; for example: "C :/Storage.mstz". It is recommended to only use this function in InitEffect().
void **ImportStoragePlace**(int storage, int place, string file)	Automatically loads a single ***MADRIX Storage Place file***. Valid values for *storage* are **LEFT, RIGHT**. Valid values for *place* are *0* to *255* (Indexing starts with 0). Valid values for *file* are *.mspz file names and the specific location on the harddisk; for example: "C:/Effect.mspz". It is recommended to only use this function in InitEffect().
void **Filter**(int filter)	Renders a filter over the matrix. »**Valid parameters (Filters)** » **Description**
int **GetAutoGainControl**()	Returns if the Automatic Gain Control (***AGC***) is enabled (**AGC**) or not (**NOAGC**).
void **SetAutoGainControl**(int mode)	Sets the Automatic Gain Control (***AGC***) function to be activated or deactivated. Valid constants for *mode* are **AGC** or **NOAGC**.
int **GetStrobe**()	Returns if the Main Output ***Strobe*** is activated (**STROBE**) or not (**NOSTROBE**).
void **SetStrobe**(int mode)	Sets the Main Output ***Strobe*** function to be activated or deactivated. Valid constants for *mode* are **STROBE** or **NOSTROBE**.
color **GetStrobeColor**()	Returns the currently used color for the Main Output ***Strobe***.
void **SetStrobeColor**(struct color)	Sets the Main Output ***Strobe*** color. »**List Of Structures** »**List of Global Constants**
int **GetStrobeValue**()	Returns the Main Output Strobe frequency factor. Valid values range from *2* to *10*. 2 represents a frequency of 25 Hz. 10 represents a frequency of 5 Hz.
void **SetStrobeValue**(int value)	Sets the Main Output Strobe frequency factor. Valid values for *value* range from *2* to *10*. 2 represents a frequency of 25 Hz. 10 represents a frequency of 5 Hz.
int **GetStorage**(int storage)	Returns the currently selected Storage ID of Deck A or Deck B with a value between *0* and *255* (Indexing starts with 0). Valid values for *storage* are **LEFT, RIGHT**.
void **SetStorage**(int storage, int storageID)	Sets the Storage ID. Valid values for *storage* are **LEFT, RIGHT**, and a value range from *0* to *255* for *storageID* (Indexing starts with 0).
Cue List	
void **CuelistStop**()	***Stop*** the Cue List. »**Example**
void **CuelistPlay**()	***Start*** the Cue List. Only applies when a Duration is set, otherwise nothing happens. »**Example**
void **CuelistGo**()	***Jump*** ahead one step in the Cue List. »**Example**
void **CuelistBack**()	Goes ***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. »**Example**
void **CuelistCueAllOccupied**()	Performs a ***Cue All***.
void **CuelistNew**(int cues)	Creates a new and empty Cue List and adds a number of new Cues as defined by *cues*. By default, *cues* has a value of *0*. »**Example**
int **CuelistProgress**()	Returns the current location of the progress bar. Values range from *0* to *100*.
void **CuelistSetTimecodeSource**(int source)	Sets the ***Time Code Source*** for the Cue List. See **below** for valid constants for *source*.
int **CuelistGetTimecodeSource**()	Returns the currently set ***Time Code Source*** of the Cue List.
void **CuelistSetTimecodeFormat**(int format)	Sets the ***Time Code Format*** for the Cue List. See **below** for valid constants for *format*.
int **CuelistGetTimecodeFormat**()	Returns the currently set ***Time Code Format*** of the Cue List.
void **CueAdd**()	Adds a new Cue to the Cue List with default (i.e. empty) settings.
void **CueDelete**(int index)	Removes the specified Cue with *index* as Cue number. *index* starts with *0*.
void **CueDeleteAll**()	Removes all Cues from the Cue List.
void **CueDeleteCurrent**()	Removes the currently played back/paused Cue from the Cue List.
void **CueSetDescription**(int index, string)	Sets a new ***Description*** *string* for the specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
string **CueGetDescription**(int index)	Returns the ***Description*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetDate**(int index , int year, int month, int day)	Sets the ***Date*** with *year*, *month*, *day* for a specified Cue with *index* as Cue number. *index* starts with *0*.
int **CueGetDateYear**(int index)	Returns the year that is set as ***Date*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetDateMonth**(int index)	Returns the month that is set as ***Date*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetDateDay**(int index)	Returns the day that is set as ***Date*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetDateWeekday**(int index, int day)	Sets a weekday as ***Date*** for a specified Cue with *index* as Cue number. *index* starts with *0*. A *day* value of *1* sets Monday, *2* sets Tuesday, *3* sets Wednesday, *4* sets Thursday, *5* sets Friday, *6* sets Saturday, *7* sets Sunday, while *8* sets Daily. »**Example**
int **CueGetDateWeekday**(int index)	Returns the weekday that is set as ***Date*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetDateString**(int index , string)	Sets the ***Date*** for a specified Cue with *index* as Cue number as *string* as shown in the Cue List (e.g. "2014/09/26"). *index* starts with *0*.
string **CueGetDateString**(int index)	Returns the ***Date*** of a specified Cue with *index* as Cue number as *string* as shown in the Cue List. *index* starts with *0*. »**Example**
void **CueSetTimeCode**(int index, int hours, int minutes, int seconds, int frames)	Sets the ***Time Code*** for a specified Cue with *index* as Cue number. *index* starts with *0*. *frames* may automatically be converted depending on the Time Code Format of the Cue List.

int **CueGetTimeCodeHour**(int index)	Returns the hours value of the ***Time Code*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetTimeCodeMinute**(int index)	Returns the minutes value of the ***Time Code*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetTimeCodeSecond**(int index)	Returns the seconds value of the ***Time Code*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetTimeCodeFrame**(int index)	Returns the frames value of the ***Time Code*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetTimeCodeString**(int index, string)	Sets the ***Time Code*** for a specified Cue with *index* as Cue number as *string* as shown in the Cue List (e.g. *"01:10:24:25"*). *index* starts with *0*.
string **CueGetTimeCodeString**(int index)	Returns the ***Time Code*** of a specified Cue with *index* as Cue number as *string* as shown in the Cue List. *index* starts with *0*. »**Example**
void **CueSetDuration**(int index, int hours, int minutes, int seconds, int frames)	Sets the ***Duration*** for a specified Cue with *index* as Cue number. *index* starts with *0*. *frames* may automatically be converted depending on the Time Code Format of the Cue List. »**Example**
int **CueGetDurationHour**(int index)	Returns the hours value of the ***Duration*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetDurationMinute**(int index)	Returns the minutes value of the ***Duration*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetDurationSecond**(int index)	Returns the seconds value of the ***Duration*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
int **CueGetDurationFrame**(int index)	Returns the frames value of the ***Duration*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetDurationString**(int index, string)	Sets the ***Duration*** for a specified Cue with *index* as Cue number as *string* as shown in the Cue List (e.g. *"02:00:00:00"*). *index* starts with *0*.
string **CueGetDurationString**(int index)	Returns the ***Duration*** of a specified Cue with *index* as Cue number as *string* as shown in the Cue List. *index* starts with *0*. »**Example**
void **CueSetFollow**(int index, int followcue)	Activates or deactivates the ***Follow Cue*** for a specified Cue with *index* as Cue number. *index* starts with *0*. A value of *0* for *followcue* turns Follow Cue off. Any other higher value for *followcue* specifies the cue number that should be set as ***Follow Cue***. »**Example**
int **CueGetFollowCue**(int index)	Returns the cue number that is set as ***Follow Cue*** of a specified Cue with *index* as Cue number. *index* starts with *0*.
void **CueSetStorage**(int index, int storage)	Sets the ***Storage*** number for a specified Cue with *index* as Cue number. *index* starts with 0. *storage* starts with *0*. »**Example**
int **CueGetStorage**(int index)	Returns the number that is set as ***Storage*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetPlace**(int index, int storageplace)	Sets the Storage ***Place*** number for a specified Cue with *index* as Cue number. *index* starts with *0*. *storageplace* starts with *0*. »**Example**
int **CueGetPlace**(int index)	Returns the number that is set as Storage ***Place*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
void **CueSetFadeType**(int index, int fadetype)	Sets the ***Fade*** Type for a specified Cue with *index* as Cue number. *index* starts with *0*. See **below** for valid constants for *fadetype*. » **Example**
int **CueGetFadeType**(int index)	Returns the currently set ***Fade*** Type of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**

string **CueGetFadeTypeString**(int index)	Returns the ***Fade*** Type of a specified Cue with *index* as Cue number as *string* as shown in the Cue List. *index* starts with *0*.
void **CueSetFadeTime**(int index , float fadetime)	Sets the ***Fade Time*** for a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
float **CueGetFadeTime**(int index)	Returns the currently set ***Fade Time*** of a specified Cue with *index* as Cue number. *index* starts with *0*. »**Example**
string **CueGetFadeTimeString**(int index)	Returns the ***Fade Time*** of a specified Cue with *index* as Cue number as *string* as shown in the Cue List (e.g. *"1.00"*). *index* starts with *0*.
void **CueSetGroupPreset**(int index, int preset)	Sets the ***Group Preset*** *preset* for a specified Cue with *index* as Cue number. *index* starts with *0*. Valid values for *preset* range from *1* to *256*.
int **CueGetGroupPreset**(int index)	Returns the currently set ***Group Preset*** for a specified Cue with *index* as Cue number. *index* starts with *0*.
Fixture Groups	
int **GetGroupCount**()	Returns the current number of Fixture Groups. »**Example**
int **GetGroupIdByIndex**(int index)	Returns the ***Fixture Group ID*** of a Fixture Group specified with *index*. The index refers to the MADRIX Group Control starting with 0; counted from left to right on the user interface.
int **GetGroupDefaultValue**(int ID)	Returns the ***Default Value*** of a Fixture Group specified with its Group ID as *ID*.
int **GetGroupDefaultValueByIndex**(int index)	Returns the ***Default Value*** of a Fixture Group specified with *index*. The index refers to the Group Control starting with 0; counted from left to right on the user interface.
color **GetGroupDisplayColor**(int ID)	Returns the ***Display Color*** of a Fixture Group specified with its Group ID as *ID*.
color **GetGroupDisplayColorByIndex**(int index)	Returns the ***Display Color*** ofof a Fixture Group specified with *index*. The index refers to the Group Control starting with 0; counted from left to right on the user interface.
string **GetGroupDisplayName**(int ID)	Returns the ***Display Name*** of a Fixture Group specified with its Group ID as *ID*. »**Example**
string **GetGroupDisplayNameByIndex**(int index)	Returns the ***Display Name*** of a Fixture Group specified with *index*. The index refers to the Group Control starting with 0; counted from left to right on the user interface.
void **SetGroupValue**(int ID, int value)	Sets the *value* for a Fixture Group specified with its Group ID as *ID*. Valid values for *ID* and *value* range from *0* to *255*. »**Example**
void **SetGroupValueByIndex**(int index, int value)	Sets the *value* for a Fixture Group specified with *index*. The index refers to the Group Control starting with 0; counted from left to right on the user interface. Valid values for *index* and *value* range from *0* to *255*.
int **GetGroupValue**(int ID)	Returns the currently set value of a Fixture Group specified with its Group ID as *ID*. »**Example**
int **GetGroupValueByIndex**(int index)	Returns the currently set value of a Fixture Group specified with *index*. The index refers to the Group Control starting with 0; counted from left to right on the user interface.
void **SetGroupFlashMode**(int ID, int mode)	Disables ***Flash*** mode for a Fixture Group specified with its Group ID as *ID* if *mode* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**).

void **SetGroupFlashModeByIndex**(int index, int mode)	Disables ***Flash*** mode for a Fixture Group specified with *index* if *mode* is set to 0 (**FALSE**). Otherwise it can be enabled using 1 (**TRUE**). The index refers to the Group Control starting with 0; counted from left to right on the user interface.
int **GetGroupFlashMode**(int ID)	Returns 1 (**TRUE**) if ***Flash*** mode is enabled for a Fixture Group specified with its Group ID as *ID*, otherwise 0 (**FALSE**).
int **GetGroupFlashModeByIndex**(int index)	Returns 1 (**TRUE**) if ***Flash*** mode is enabled for a Fixture Group specified with *index*, otherwise 0 (**FALSE**). The index refers to the Group Control starting with 0; counted from left to right on the user interface.
void **ToggleGroupFlashMode**(int ID)	Enables ***Flash*** mode or uses the default setting for a Fixture Group specified with its Group ID as *ID*, depending on the current state.
void **ToggleGroupFlashModeByIndex**(int index)	Enables ***Flash*** mode or uses the default setting for a Fixture Group specified with *index*, depending on the current state. The index refers to the Group Control starting with 0; counted from left to right on the user interface.
void **SetGroupFadeTime**(float time)	Sets the ***Fade-In Time*** for the Group Control.
float **GetGroupFadeTime**()	Returns the currently set ***Fade-In Time*** for the Group Control.
void **SetGroupPreset**(int preset)	Sets all current Fixture Group values as a ***Group Preset*** specified by *preset*. Indexing starts with *0*.
void **CallGroupPreset**(int preset)	Calls a ***Group Preset*** specified by *preset*. Indexing starts with *0*.

Constants

The following values (defines) must be used as parameters for the functions provided above.

Value	Description
Side Selection	
int **LEFT**	Selects Deck A. To be used with miscellaneous functions, e.g. SetFilterColor
int **RIGHT**	Selects Deck B. To be used with miscellaneous functions, e.g. SetFilterColor
int **MAIN**	Selects the Main Output. To be used with miscellaneous functions, e.g. SetFilterColor
Fade Types	
int **CROSSFADE**	Selects the Cross-fade ***XF*** as Fade Type.
int **WHITEFADE**	Selects the White-fade ***WF*** as Fade Type.
int **BLACKFADE**	Selects the Black-fade ***BF*** as Fade Type.
int **X_WIPE**	Selects the ***Wipe X*** as Fade Type.
int **Y_WIPE**	Selects the ***Wipe Y*** as Fade Type.
int **Z_WIPE**	Selects the ***Wipe Z*** as Fade Type.
int **X_CROSS_WIPE**	Selects the ***Wipe XC*** as Fade Type.
int **Y_CROSS_WIPE**	Selects the ***Wipe YC*** as Fade Type.
int **Z_CROSS_WIPE**	Selects the ***Wipe ZC*** as Fade Type.
int **X_SLIDE**	Selects the ***Slide X*** as Fade Type.
int **Y_SLIDE**	Selects the ***Slide Y*** Slide as Fade Type.
int **Z_SLIDE**	Selects the ***Slide Z*** Slide as Fade Type.
int **X_CROSS_SLIDE**	Selects the ***Slide XC*** as Fade Type.
int **Y_CROSS_SLIDE**	Selects the ***Slide YC*** as Fade Type.
int **Z_CROSS_SLIDE**	Selects the ***Slide ZC*** as Fade Type.
Fade Functions	
int **WITH_AUTOFADE**	Activates automatic fade. To be used with SetStoragePlace
int **WITHOUT_AUTOFADE**	Deactivates automatic fade. To be used with SetStoragePlace
int **AUTOMATIC**	Activates an automatic crossfade. To be used with SetFade
int **TO_LEFT**	Activates an automatic crossfade to the left. To be used with SetFade
int **TO_CENTER**	Activates an automatic crossfade to the middle. To be used with SetFade
int **TO_RIGHT**	Activates an automatic crossfade to the right. To be used with SetFade
Freeze Function	
int **FREEZE**	Freezes the main output instantly

MADRIX 3.X To MADRIX 5.X Migration Hints

The following constants and functions are not supported anymore. Please follow the hints to migrate your macros.

Function	Description
int **GetAudioFader**()	Use **GetAudioInputFader**() instead.
void **SetAudioFader**(int value)	Use **SetAudioInputFader**(int value) instead.
int **STORAGE_LEFT**	Use **LEFT** instead.
int **STORAGE_RIGHT**	Use **RIGHT** instead.
int **GetStorageFullState**(int storage, int place)	Use **GetStoragePlaceFullState**(int storage, int place) instead.
int **GetStorageFilter**(int storage)	Use **GetStoragePlaceFilter**(int storage) instead.
int **SetStorageFilter**(int storage, int filter)	Use **SetStoragePlaceFilter**(int storage, int filter) instead.
void **SetFilterColor**(color c, int side)	Use **SetFilterColor**(int side, color c) instead.
int **COLORFADE**	This constant is not supported anymore.
color **GetFadeColor**()	This function is not supported anymore.
void **SetFadeColor**(color fadecolor)	This function is not supported anymore.
color **CueGetFadeColor**(int index)	This function is not supported anymore.
void **CueSetFadeColor**(int index, color fadecolor)	This function is not supported anymore.

7.3 Examples (Global Macro)

GetFadeType

Select a different fade type while running this Global Macro 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 X_WIPE: WriteText("X Wipe");break;
     case Y_WIPE: WriteText("Y Wipe");break;
     case Z_WIPE: WriteText("Z Wipe");break;
     case X_CROSS_WIPE: WriteText("XC Wipe");break;
     case Y_CROSS_WIPE: WriteText("YC Wipe");break;
     case Z_CROSS_WIPE: WriteText("ZC Wipe");break;
     case X_SLIDE: WriteText("X Slide");break;
     case Y_SLIDE: WriteText("Y Slide");break;
     case Z_SLIDE: WriteText("Z Slide");break;
     case X_CROSS_SLIDE: WriteText("XC Slide");break;
     case Y_CROSS_SLIDE: WriteText("YC Slide");break;
     case Z_CROSS_SLIDE: WriteText("ZC Slide");break;
  }
}

void PostRenderEffect()
{}
```

»Description

SetFadeType

Watch the fade area while running the script to see how the fade types are selected in succession.

```
@scriptname="set fade type test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{
 Value%=400;// range 0...399
 switch(Value)
 {
    case 0: SetFadeType(CROSSFADE);break;
    case 25: SetFadeType(WHITEFADE);break;
    case 50: SetFadeType(BLACKFADE);break;
    case 75: SetFadeType(X_WIPE);break;
    case 100: SetFadeType(Y_WIPE);break;
    case 125: SetFadeType(Z_WIPE);break;
    case 150: SetFadeType(X_CROSS_WIPE);break;
    case 175: SetFadeType(Y_CROSS_WIPE);break;
    case 200: SetFadeType(Z_CROSS_WIPE);break;
    case 225: SetFadeType(X_SLIDE);break;
    case 250: SetFadeType(Y_SLIDE);break;
    case 275: SetFadeType(Z_SLIDE);break;
```

```
        case 300: SetFadeType(X_CROSS_SLIDE);break;
        case 325: SetFadeType(Y_CROSS_SLIDE);break;
        case 350: SetFadeType(Z_CROSS_SLIDE);break;
    }
    Value++;
}

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 value 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 crossfader from Left to Right or back and monitor the ***Script Output*** while running the script.

```
@scriptname="get fader value Left-Right";
@author="";
@version="";
@description="";

int fValue;
void InitEffect()
{}

void PreRenderEffect()
{
 fValue=GetFadeValue();
 WriteText((string)fValue);
}

void PostRenderEffect()
{}
```

»Description

SetFadeValue

Watch the crossfader to see the result of the script.

```
@scriptname="set fader value Left-Right";
@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...99
 switch(Value)
 {
    case 5: SetFreeze(FREEZE);break;
    case 55: SetFreeze(UNFREEZE);break;
 }
}

void PostRenderEffect()
{}
```

»Description

GetMasterFader

Adjust the ***Master*** while running the script and monitor the ***Script Output***.

```
@scriptname="get fader value from master";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
 WriteText("Master: "+(string)GetMasterFader());
}

void PostRenderEffect()
{}
```

»Description

SetMasterFader

Watch the ***Master*** 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

GetAudioInputFader

Adjust the ***Audio Input Level*** while running the script and monitor the ***Script Output***.

```
@scriptname="get fader value from audio input";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
 WriteText("Audio Input Fader: "+(string)GetAudioInputFader());
}

void PostRenderEffect()
{}
```

»Description

SetAudioInputFader

Monitor the ***Audio Input Level*** closely to see the result of the script.

```
@scriptname="set fader value of audio input";
```

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

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{
 Value++;
 Value%=256;// range 0...255
 SetAudioInputFader(Value);
}

void PostRenderEffect()
{}
```

»Description

GetStorageSpeedMaster

Adjust the ***Speed Master*** Left or Right while running the script and monitor the ***Script Output***.

```
@scriptname="get Speed Master test";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
WriteText("SpeedLeft:"+(string)GetStorageSpeedMaster(LEFT)
+" SpeedRight:"+(string)GetStorageSpeedMaster(RIGHT));
}

void PostRenderEffect()
{}
```

»Description

SetStorageSpeedMaster

This is script is best tested with the SCE Color Scroll effect. Monitor ***Preview Left*** and ***Preview Right*** as well as the ***Speed Masters*** 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 -10.0...10.0

 SetStorageSpeedMaster(LEFT,Value);       // Deck A
 SetStorageSpeedMaster(RIGHT,Value/2.0);  // Deck B
}

void PostRenderEffect()
{}
```

»Description

GetStoragePause

To test this script, press the 'Pause' button of Storage Area Left or Right and monitor the ***Script Output***.

```
@scriptname="get storage pause test";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
if(GetStoragePause(LEFT))
  WriteText("Deck A Paused");
```

```
else
  WriteText("Deck A Running");
if(GetStoragePause(RIGHT))
  WriteText("Deck B Paused");
else
  WriteText("Deck B Running");
}

void PostRenderEffect()
{}
```

»Description

SetStoragePause

This is script is best tested with the SCE Color Scroll effect. Monitor the ***Pause*** buttons of Deck A and Deck 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...199

 switch(Value)
 {
        case 10: SetStoragePause(LEFT,PAUSE);break;// Deck A
        case 60: SetStoragePause(RIGHT,PAUSE);break;// Deck B
        case 110: SetStoragePause(LEFT,NOPAUSE);break;// Deck A
        case 160: SetStoragePause(RIGHT,NOPAUSE);break;// Deck B
 }
}
void PostRenderEffect()
{}
```

»Description

GetStoragePlace

While running the script, select different Storage Places across Deck A or Deck B and monitor the ***Script Output***. The currently selected Storage Place number minus 1 should be displayed.

```
@scriptname="get storage place test";
@author="";
@version="";
@description="";

void InitEffect()
{}

void PreRenderEffect()
{
  WriteText("Deck A:"+(string)GetStoragePlace(LEFT)
  +" Deck B:"+(string)GetStoragePlace(RIGHT));
}

void PostRenderEffect()
{}
```

»Description

SetStoragePlace

Monitor the Effect Areas Left and Right as well as the two Storage Areas (Left and Right) 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...999
  switch(Value)
  {
  case 150: SetStoragePlace(LEFT,place,WITH_AUTOFADE);break;
  // Deck A with Autofade
  case 300: SetStoragePlace(RIGHT,place,WITH_AUTOFADE);break;
  // Deck B with Autofade
  case 450: place++;
  case 600: SetStoragePlace(LEFT,place,WITHOUT_AUTOFADE);break;
  // Deck A without Autofade; if FaderValue not a place on Deck A
  case 750: SetStoragePlace(RIGHT,place,WITHOUT_AUTOFADE);break;
  // Deck B without autofade; if FaderValue not a place on Deck B
  case 900: place++;
  }
  if(place>255)   // place range 0...255
         place=0;
}

void PostRenderEffect()
{}
```

»Description

GetStorageSubMaster

Adjust the ***Submasters*** of Deck A and Deck B (***SUB***) and monitor the ***Script Output*** while running the script.

```
@scriptname="storage submaster get test";
@author="";
@version="";
@description="";

int subLeft;
int subRight;
void InitEffect()
{}

void PreRenderEffect()
{
 subLeft=GetStorageSubMaster(LEFT);// Deck A
 subRight=GetStorageSubMaster(RIGHT);// Deck B
 WriteText("Submaster Deck A:" +(string)subLeft +", Submaster Deck B:" +(string)subRight);
}

void PostRenderEffect()
{}
```

»Description

SetStorageSubMaster

Select an SCE Effect in both Effect Areas and monitor the two previews (Preview Left and Preview Right) 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(LEFT,Value);// Deck A
 SetStorageSubMaster(RIGHT,255-Value);// Deck B
 WriteText("SubLeft: "+(string)Value+", SubRight: "+(string)(255-Value));
}

void PostRenderEffect()
{}
```

»Description

CuelistStop/CuelistPlay

To test this script, a ***Cue List*** with Duration and a minimum of 2 Cues is required.

```
@scriptname="cue list stop/start test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{
```

```
  Value++;
  Value%=500;// range 0...499
  switch(Value)
  {
     case 10: CuelistStop();break;
     case 260: CuelistPlay();break;
  }
}

void PostRenderEffect()
{}
```

»Description

CuelistGo/CuelistBack

To test this script, a ***Cue List*** with Duration and a minimum of 2 Cues is required.

```
@scriptname="cue list go/back test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{

  Value++;
  Value%=500;// range 0...499
  switch(Value)
  {
     case 10: CuelistGo();break;
     case 260: CuelistBack();break;
  }
}

void PostRenderEffect()
{}
```

»Description

CuelistGoto

To test this script, a ***Cue List*** with Duration and a minimum of 9 Cues is required.

```
@scriptname="cue list goto test";
@author="";
@version="";
@description="";

int Value=0;
void InitEffect()
{}

void PreRenderEffect()
{

 Value++;
 Value%=250;// range 0...249
 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="inoage / info@madrix.com";
@version="";
@description="View current item from Cue List";

void InitEffect()
{}

void PreRenderEffect()
{}

void PostRenderEffect()
{
     WriteText((string)CuelistCurrentCue());
}
```

»Description

Cue List - Various 1

This Macro creates a new Cue List and adds 8 new Cues. All 8 Cues will be given different settings that are increased each time. Make sure to open the Cue List before or after running this Script to see the result.

```
@scriptname="creating a cue list";
@author="inoage";
@version="MADRIX 3.3";
@description="";

void InitEffect()
{
    CuelistNew(8);    //create a new Cue List and add 8 Cues

    for(int i=0;i<8;i++)     // change the following settings for each Cue
    {
        CueSetDescription(i, "Color"+ (string)(i+1));
        CueSetTimeCode(i, i+1,i+2,i+3,i+4);
        CueSetDuration(i, i+5,i+6,i+7,i+8);
        CueSetDateWeekday(i, i+1);
        CueSetFollow(i, (i+2)%9);
        CueSetStorage(i, (i%2));
        CueSetPlace(i, (i%4));
        CueSetFadeType(i, i);
        CueSetFadeTime(i, (float)i*10.0+10.0);
    }
}

void PreRenderEffect()
{

}

void PostRenderEffect()
{

}

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

»Description

Cue List - Various 2

Make sure to create or load a Cue List first. This Global Macro analyzes the current Cue List and shows all Cues and their settings in the ***Script Output***.

```
@scriptname="simple cue list view 1";
@author="inoage";
@version="MADRIX 3.3";
@description="";

void InitEffect()
{
    string sCueCurr;
    const int count = CuelistCount();
    const int CueCurrent = CuelistCurrentCue();
    if(count>0)
        WriteText("Cue List View");
    else
        WriteText("Cue List is empty!");

    for(int i=0;i<count;i++) //analyze all Cues
    {
        if(CueCurrent==i)
            sCueCurr = ">";
        else
            sCueCurr = " ";

        WriteText(  sCueCurr                        + "  "  + // select current cue
                    (string)i                       + " | " + // number
                    CueGetDescription(i)            + " | " + // cue description
                    CueGetDateString(i)             + " | " + // get ready date string
                    CueGetTimeCodeString(i)         + " | " + // get ready time code string
                    CueGetDurationString(i)         + " | " + // get ready duration string
                    (string)CueGetStorage(i)        + " | " +
                    (string)CueGetPlace(i)          + " | " +
                    (string)CueGetFadeType(i)       + " | " +
                    (string)CueGetFadeTime(i)       +"s");
    }

}

void PreRenderEffect()
{
}

void PostRenderEffect()
{
}

void MatrixSizeChanged()
{
```

```
        InitEffect();
    }
```

»Description

Cue List - Various 3

Make sure to create or load a Cue List first. This Global Macro analyzes the current Cue List and shows all Cues and their settings in the ***Script Output***.

```
@scriptname="simple cue list view 2";
@author="inoage";
@version="MADRIX 3.3";
@description="";

void InitEffect()
{
    string sCueCurr;
    const int count = CuelistCount();
    const int CueCurrent = CuelistCurrentCue();
    if(count>0)
        WriteText("Cuelist View");
    else
        WriteText("Cuelist is empty");

    for(int i=0;i<count;i++) //analyze all Cues
    {
        if(CueCurrent==i)
            sCueCurr = ">";
        else
            sCueCurr = " ";

        string sTimecode ="--:--:--:--";
        if(CueGetTimeCodeHour(i)>=0)
            sTimecode = (string)CueGetTimeCodeHour(i)  + ":" +
                        (string)CueGetTimeCodeMinute(i)+ ":" +
                        (string)CueGetTimeCodeSecond(i)+ ":" +
                        (string)CueGetTimeCodeFrame(i);

        string sDuration ="--:--:--:--";
        if(CueGetDurationHour(i)>=0)
            sDuration = (string)CueGetDurationHour(i)  + ":" +
                        (string)CueGetDurationMinute(i)+ ":" +
                        (string)CueGetDurationSecond(i)+ ":" +
                        (string)CueGetDurationFrame(i);

        string sDate ="--/--/--";

        switch(CueGetDateWeekday(i))
        {
         default:sDate =   (string)CueGetDateYear(i)  + "/" +
                           (string)CueGetDateMonth(i) + "/" +
                           (string)CueGetDateDay(i);
```

```
                                break;
            case 8: sDate="daily"; break;
            case 7: sDate="Sun";   break;
            case 6: sDate="Sat";   break;
            case 5: sDate="Fri";   break;
            case 4: sDate="Thu";   break;
            case 3: sDate="Wed";   break;
            case 2: sDate="Tue";   break;
            case 1: sDate="Mon";   break;
            case 0: sDate="ERROR"; break;
           }

           WriteText(  sCueCurr                        + "  "  + // select current cue
                       (string)i                       + " | " + // number
                       CueGetDescription(i)            + " | " + // cue description
                       sDate                           + " | " + // date string
                       sTimecode                       + " | " + // time code string
                       sDuration                       + " | " + // duration string
                       (string)CueGetStorage(i)        + " | " +
                       (string)CueGetPlace(i)          + " | " +
                       (string)CueGetFadeType(i)       + " | " +
                       (string)CueGetFadeTime(i)       +"s");
        }

    }

    void PreRenderEffect()
    {
    }

    void PostRenderEffect()
    {
    }

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

»**Description**

Fixture Groups - Various

Set up different Fixture Groups first. This Global Macro analyzes specific Fixture Group information and shows their settings in the ***Script Output***.

```
@scriptname="";
@author="inoage";
@version="3.4";
@description="get fixture groups' data";

int GroupValue;
int GroupId;
```

```
string GroupName;

void InitEffect()
{
    if(GetGroupCount()<1)
        WriteText("No groups have been set up!");

}

void PreRenderEffect()
{
    const int GroupCount = GetGroupCount();
    for(int i=0; i<GroupCount; i++)
    {
        GroupId = GetGroupIdByIndex(i);
        GroupName = GetGroupDisplayName(i);
        GroupValue = GetGroupValue(i);
        WriteText("Group: " + (string)(i)+ ", ID: " + (string)GroupId+ ", Name: " + (string)GroupNam
    }
}

void PostRenderEffect()
{

}

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

»Description

SetGroupValue

Set up different Fixture Groups first. This Global Macro steadily increases the values for your Fixture Groups in the Group Control and starts from the beginning again when the maximum value is reached.

```
@scriptname="";
@author="";
@version="";
@description="set groups value";

int FrameCount;
int Value;

void InitEffect()
{
    if(GetGroupCount()<1)
        WriteText("No groups have been set up!");
    FrameCount=0;
}
```

```
void PreRenderEffect()
{
    const int GroupCount = GetGroupCount();
    for(int i=0; i<GroupCount; i++)
    {
        Value = FrameCount/(i+1);
        SetGroupValue(i, Value);
    }

    FrameCount++;
    FrameCount= FrameCount%256;
}

void PostRenderEffect()
{

}

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

»Description

MADRIX LIGHTING CONTROL

//PART H

Imprint And Copyright

8 Imprint And Copyright

Company And Address

inoage GmbH
Wiener Straße 56
01219 Dresden
Germany

Managing Directors: Christian Hertel, Sebastian Pinzer, Sebastian Wissmann

Phone: +49 351 862 6869 0
Fax: +49 351 862 6869 68

Web: www.madrix.com
E-mail: info@madrix.com

Copyright

Under no circumstances does inoage GmbH take on the responsibility for liabilities for faults for loses 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.