GraphicsMagick in FileMaker

GraphicsMagick in FileMaker, part 1

1 of 24

Welcome to the first door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component. Every day I will present you one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw on them and much much more. In the end, you can also take the magic of GraphicsMagick to your images. I wish you a lot of fun with it.

Today I will show you how to load an image from a file or container so that you can edit it in the later doors. When loading an image we have the possibility to load it from a container. For this we use the GMImage.NewFromContainer function. We pass the container to this function in the parameters. With this we load the image into the working memory and get a reference number as return which we can use in the other steps.

```
Set Variable [ $ref1 ; Value:
MBS( "GMImage.NewFromContainer"; GraphicsMagick
Advent::Image) ]
```

If we have an image file with multiple images in a container, we have the function GMImage.NewImagesFromContainer that gives us a list of reference numbers. Each image in the container is loaded into memory and has its own reference number.

```
Set Variable [ $ref2 ; Value:
MBS( "GMImage.NewImagesFromContainer"; GraphicsMagick
Advent::Image ) ]
```

The image cannot only be a container value, but also a file on your computer. You can then load this via the file path. We use the GMImage.NewFromFile function for this. We specify the native path in the parameters. If you have a FileMaker path, the path must first be converted to a native path. Then use the Path.FileMakerPathToNativePath function to do this. If you want, you can optionally specify a codec here that will take care of decoding the image.

```
Set Variable [ $ref3 ; Value: MBS( "GMImage.NewFromFile"; "/
Users/sj/Desktop/IMG_4420.jpeg"; "JPEG") ]
```

Also to this function there is of course a function that if we have several images in a file, loads these images into memory and returns us a list of references. This function is called GMImage.NewImagesFromFile.

Images can now not only be a file or container value, but also encoded as a string. We can use the functions GMImage.NewFromBase64 and GMImage.NewFromHex to receive image data as a hexadecimal string or a base64 string. The images are decoded and loaded into memory in the same way as the other functions and as return we get the reference number.

```
Set Variable [ $ref4 ; Value: MBS( "GMImage.NewFromBase64";
GraphicsMagick Advent::Text) ]
Set Variable [ $Ref5 ; Value: MBS( "GMImage.NewFromHex";
GraphicsMagick Advent::Text) ]
```

Before I leave you excited about door number two, I would like to explain one very important thing about working with GraphicsMagick. We store our image references in memory. If we create a new image reference then again a piece of memory is allocated. So that we don't block our memory now we free the memory when we don't need the reference anymore. We can do this with the function GMImage.Release for a single reference by specifying the reference number to be released in the parameters, or for all reference numbers with the function GMImage.ReleaseAll.

```
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $ref1 ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

Tomorrow we will continue. I wish you a nice first of December.

GraphicsMagick in FileMaker, part 2

2 of 24

Welcome to the second door of our advent calendar in this advent calendar. I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, you can draw them and much much more. In the end, you can also take the magic of GraphicsMagick to your images. I wish you a lot of fun with it.

Today I want to find out with you some information about the loaded images. We will start with the size of the images. So we want to find out height and width. First we load our image into memory as we saw it yesterday. In our example there is a single image in the container.

```
Set Variable [ $GM ; Value:
MBS("GMImage.NewFromContainer"; GraphicsMagick
Advent::Image) ]
```

Now we can retrieve different information. The width with the function GMImage.GetWidth and the height with GMImage.GetHeight function. The reference is passed to the functions in the parameters and we then get back the desired value.

```
Set Variable [ $Width ; Value: MBS( "GMImage.GetWidth";
$GM ) ]
Set Variable [ $Height ; Value: MBS( "GMImage.GetHeight";
$GM ) ]
```

We can also get the height and width directly in one step. For this we use the function GMImage.GetSize. Here we get width and height separated with an X as result.

```
Set Variable [ $Size ; Value: MBS( "GMImage.GetSize";
$GM ) ]
```

Besides the size, we can also query other information about the image, e.g. what file format does the image have? We can easily answer this question with the function GMImage.GetMagick, because it gives us the codec used.

```
Set Variable [ $FileTyp ; Value: MBS( "GMImage.GetMagick";
$GM ) ]
```

With the function GMImage.GetDensity we query the resolution. We get the resolution of the width and the height again separated by an x.

```
Set Variable [ $Resolution ; Value:
MBS( "GMImage.GetDensity";$GM ) ]
```

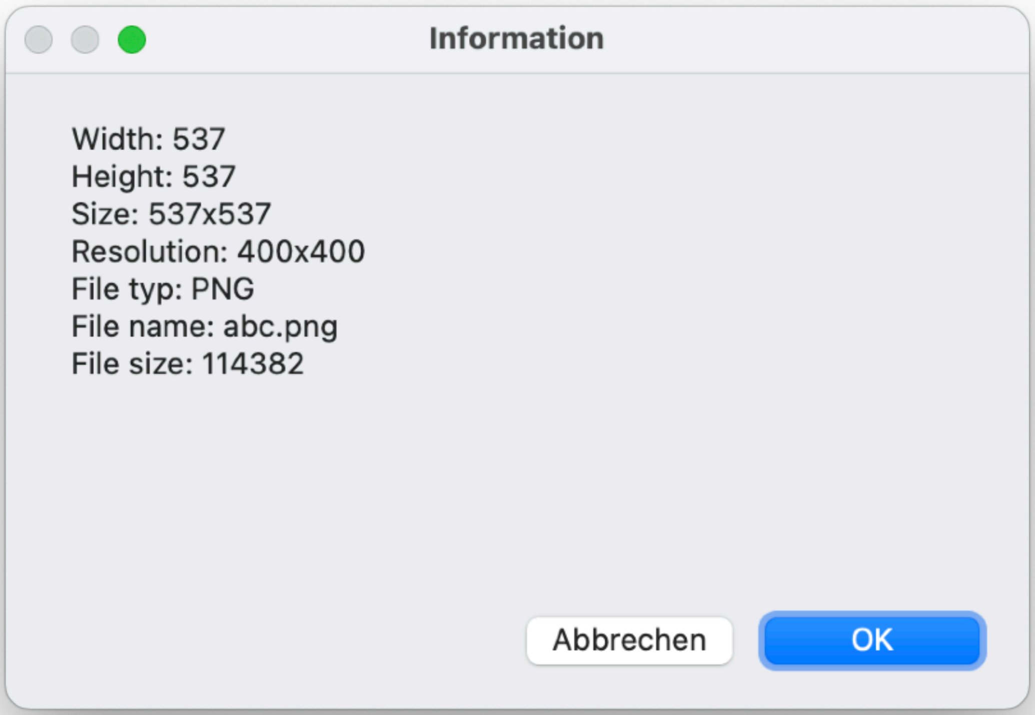
The file name can be queried with GMImage.GetFileName.

```
Set Variable [ $FileName ; Value:
MBS( "GMImage.GetFileName"; $GM ) ]
```

Also the file size can be determined with the function GMImage.GetFileSize.

```
Set Variable [ $FileSize ; Value:
MBS( "GMImage.GetFileSize"; $GM ) ]
```

Here you can see how a dialog can look like if we print the information in the dialog.



```
Width: 537
Height: 537
Size: 537x537
Resolution: 400x400
File typ: PNG
File name: abc.png
File size: 114382
```

GraphicsMagick has many more functions that you can use to get information about your images. Please have a look. I wish you a nice 2nd December and I am looking forward to seeing you tomorrow.

GraphicsMagick in FileMaker, part 3

3 of 24

Welcome to the third door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

So far we have requested information from images, But later in December we would like to change the images and would like to see this change when we put a filter on the image, for example. For this we need to be able to write the edited image which is in our working memory back to a file. How this works I will show you today. First we think about where we want to have our file. Do we want to write it into a file or into a container, for both we need again different functions.

Let's start with the case that we want to write the image into a file. For this we use the function GMImage.WriteToFile. This function writes a single image to the disk in the parameters we specify the image reference and the path to which the image should be saved. If you want to let the user choose in a dialog where to save the file you can use the functions from the FileDialog component. With these functions you can customize your save dialog by adding a text, a heading and many other settings. After selecting a file we can then get the file path and save the image with the GMImage.WriteToFile function to the specified location. In the script example we have used such a dialog.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "FileDialog.Reset" ) ]
Set Variable [ $r ; Value: MBS( "FileDialog.SetWindowTitle";
"Save the image") ]
Set Variable [ $r ; Value: MBS( "FileDialog.SetMessage";
"Where should the image be saved?" ) ]
```

```
Set Variable [ $r ; Value:
MBS( "FileDialog.SaveFileDialog" ) ]
Set Variable [ $Path ; Value: MBS("FileDialog.GetPath"; 0) ]
Set Variable [ $Path ; Value: $Path & ".png" ]
Set Variable [ $r ; Value: MBS( "GMImage.WriteToFile"; $GM;
$Path ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```

We have our image reference in the variable $GM. First, we reset all the existing settings of the dialog so that we don't take any legacy with us. The dialog gets a window title with the text *"Save the image""* and the message *"Where should the image be saved?"*
We have decided to use the save dialog because we want to write a file to the disk and so we only adapt the standard save dialog from the operating system to our wishes. For this we use the function FileDialog.SaveFileDialog. We get the path we selected with the function FileDialog.GetPath. Since you can also select multiple files directly with a file dialog we have to specify an index with this function to determine how many paths we want. We want the first one and for this reason we specify 0, because in the MBS FileMaker Plugin we start counting indexes at 0. We still need to specify what type our file should be, otherwise a text file will be written with the image. For file path we simply append the sufix, here png. Now we can use the function to write it to the disk and of course release the reference again.

If we have several references at the same time, we can also write them directly to the disk in one step. For this we use the GMImage.WriteImages function. In the parameters we specify a list with the references, then again the path to which we want to save the images. Last but not least we can decide if we want to save the images as single files or if we want to write the images into a multi image tiff file.

We can also put an image directly into a container. We use the GMImage.WriteToContainer function. In the parameters we first specify the reference and then the file name. With this function we can choose before which format our image should have. For this we use GMImage.SetMagick before this function. Here we also specify the reference and the type.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetMagick"; $GM;
"PNG") ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
```

If you already know in detail which file format your image should have in the container, then you can use one of our special functions that we offer for some file formats. The functions have the same parameters as the main function. Here is a list of these special functions:

- GMImage.WriteToBMPContainer
- GMImage.WriteToGIFContainer
- GMImage.WriteToJPEGContainer
- GMImage.WriteToPDFContainer
- GMImage.WriteToPNGContainer
- GMImage.WriteToTiffContainer

We'll be using the functions we learned about today often during the Advent calendar. I hope you enjoyed the third door and I will see you here again tomorrow for door number four.

GraphicsMagick in FileMaker, part 4

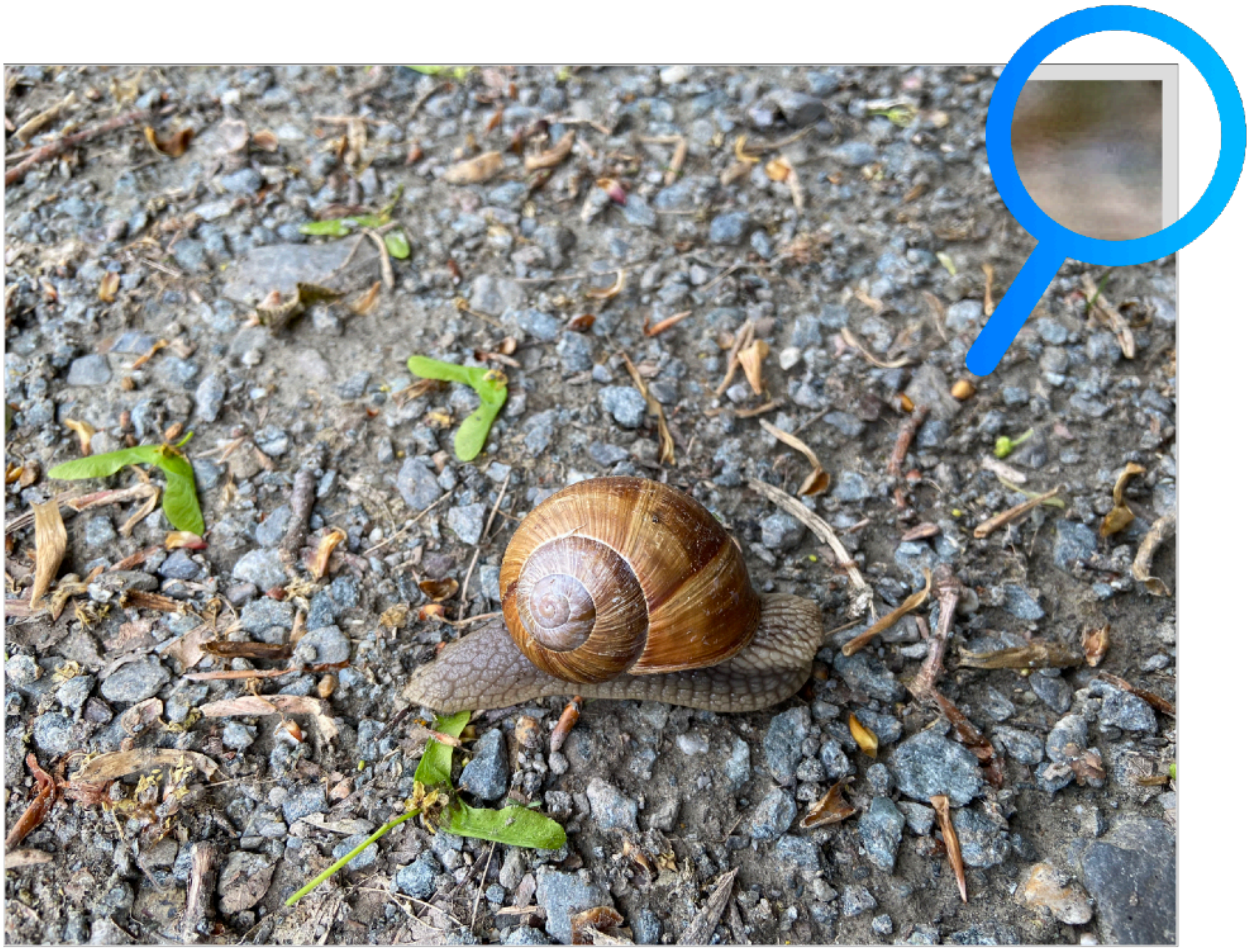
4 of 24

Welcome to the fourth door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I will show you how to paint a border around an image. The focus is on the GMImage.Border function. This function again requires our reference.

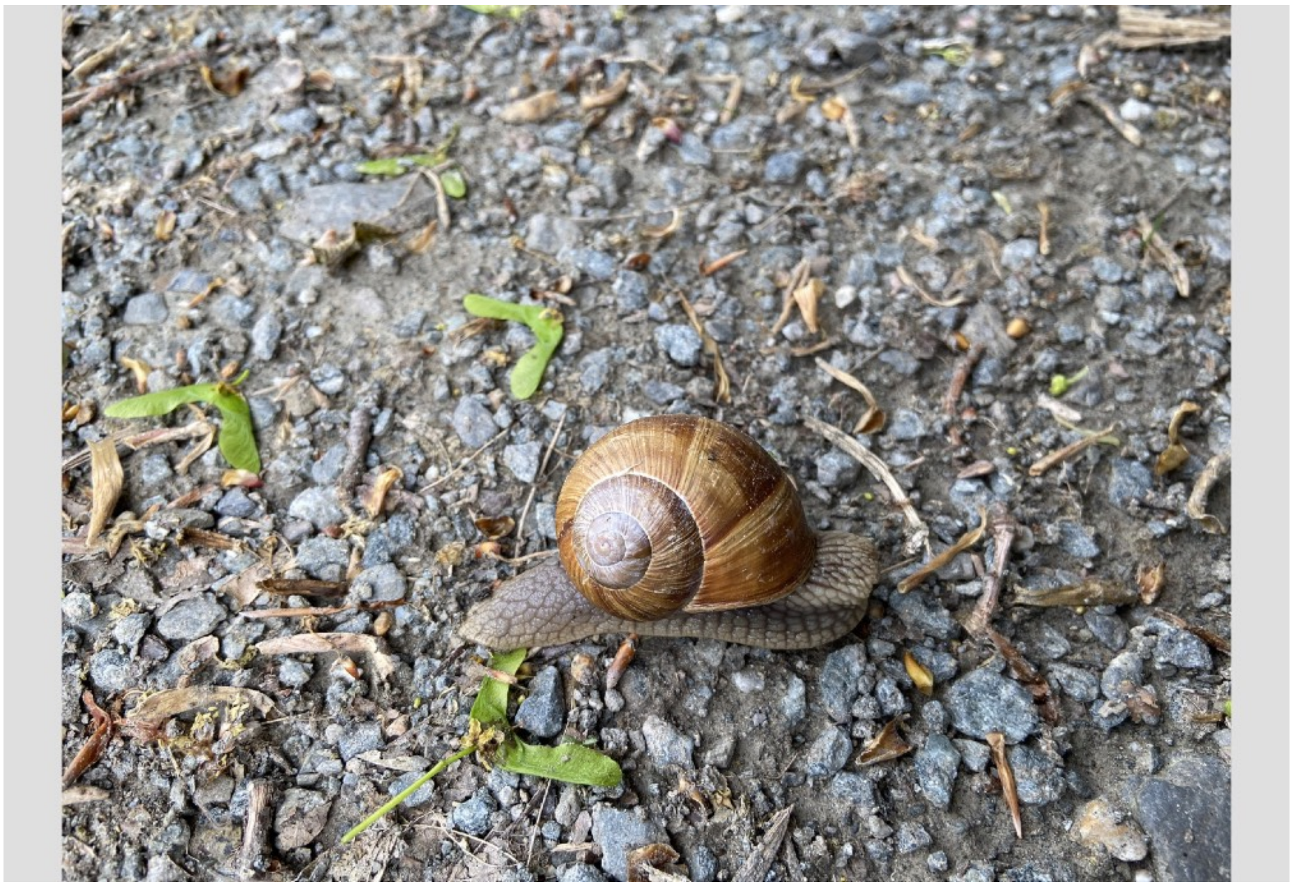
```
Set Variable [ $r ; Value:
MBS( "GMImage.Border"; $GM) ]
```

When we use the function like this, a gray fine border is drawn around the image by default. If the image is very large, as you can see here, it is not so noticeable.



But we can change that. In the function itself we can change the thickness of the frame by specifying a geometry and we can change the color of the frame. Let's start with the thickness of the border. In a geometry we specify a width and height. In our case the thickness of the border. The two values are separated from each other with an x. If I enter as geometry e.g. 200x0 then our frame looks like this.

```
Set Variable [ $r ; Value: MBS( "GMImage.Border";$GM;
"200x0") ]
```



Left and right we have now a frame of 200 pixel. If we invert the values, we get a frame at the top and bottom.

But we can change not only the thickness of the border, but also the color. The color must be set before we paint the border. For this we use the GMImage.SetBorderColor function. In the parameters we specify the reference and the color. The color can be specified in different color spaces. Here you can see a list of the color spaces that are possible:

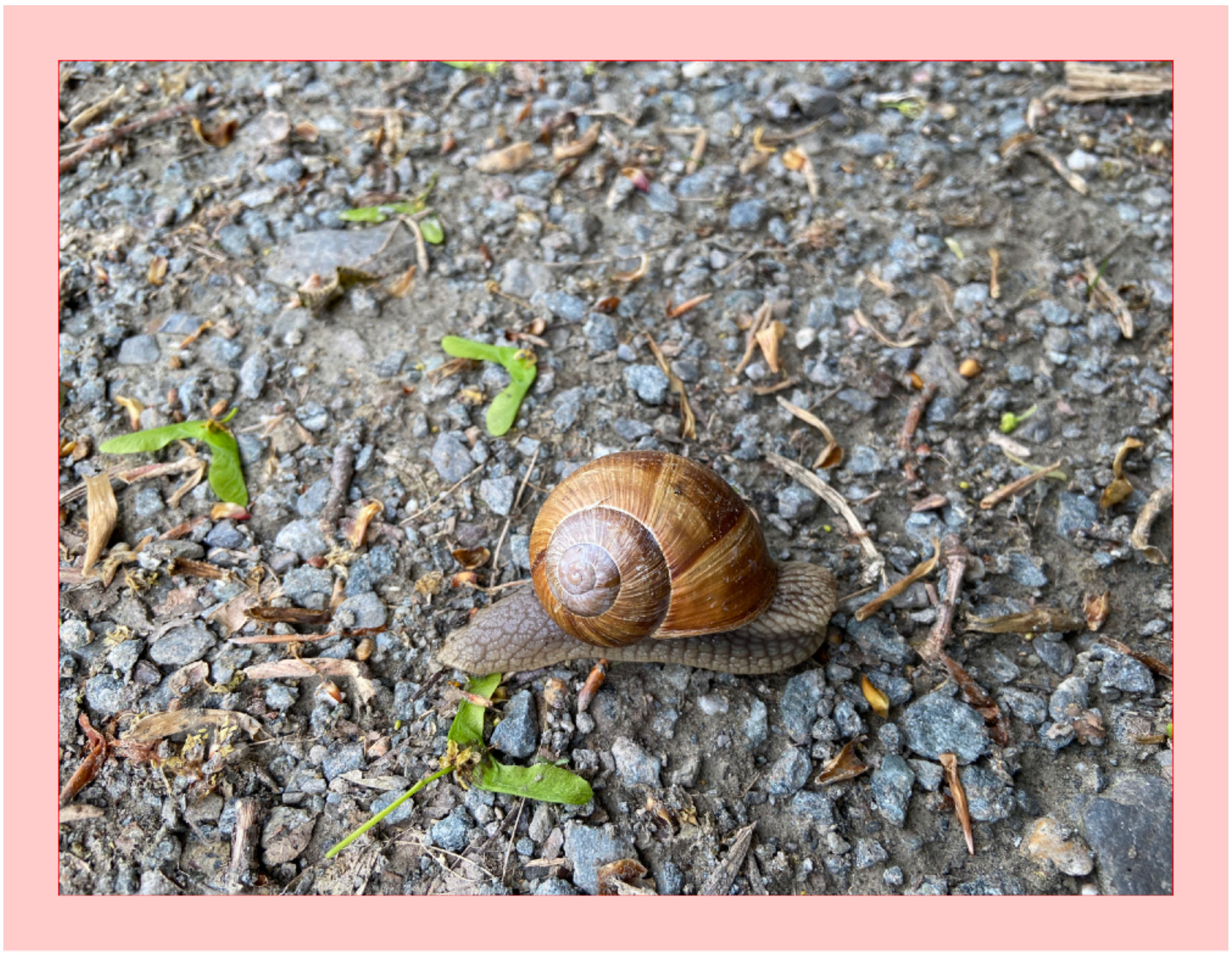
- HSL h s l a
- YUV y u v a
- RGB r g b a
- MONO m a
- GRAY g a
- COLOR R G B a

also you can specify a color value as a hexadecimal number. As you can see, you can also always define an alpha value, which means we define the transparency. This value is between 0.0 and 1.0.

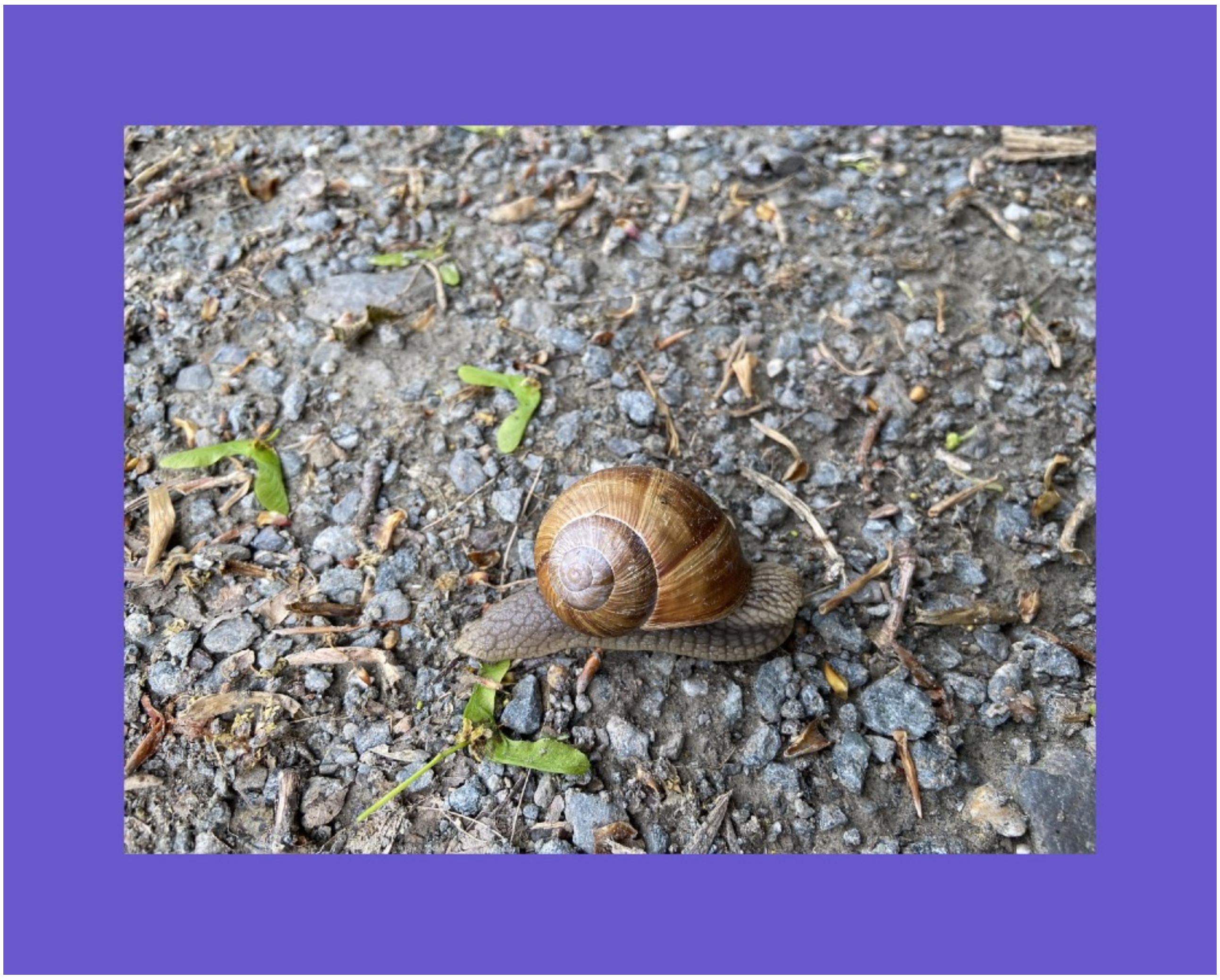
```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromFile"; "/
Users/sj/Desktop/abc.png") ]
Set Variable [ $r ; Value: MBS( "GMImage.SetBorderColor";
$GM; "#FF000034" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Border";$GM;
"200x200") ]
```

```
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```

In the example code I have defined a hexadecimal number that describes a color that is red and has a high transparency. The result looks like this.



Of course, you can choose the colors that you like. Here you see for example a picture with a purple (#6959CD) frame. The frame has the size 500x500.



So give your pictures the frame you need. Tomorrow we will continue. Until then have a nice 4th of December

GraphicsMagick in FileMaker, part 5

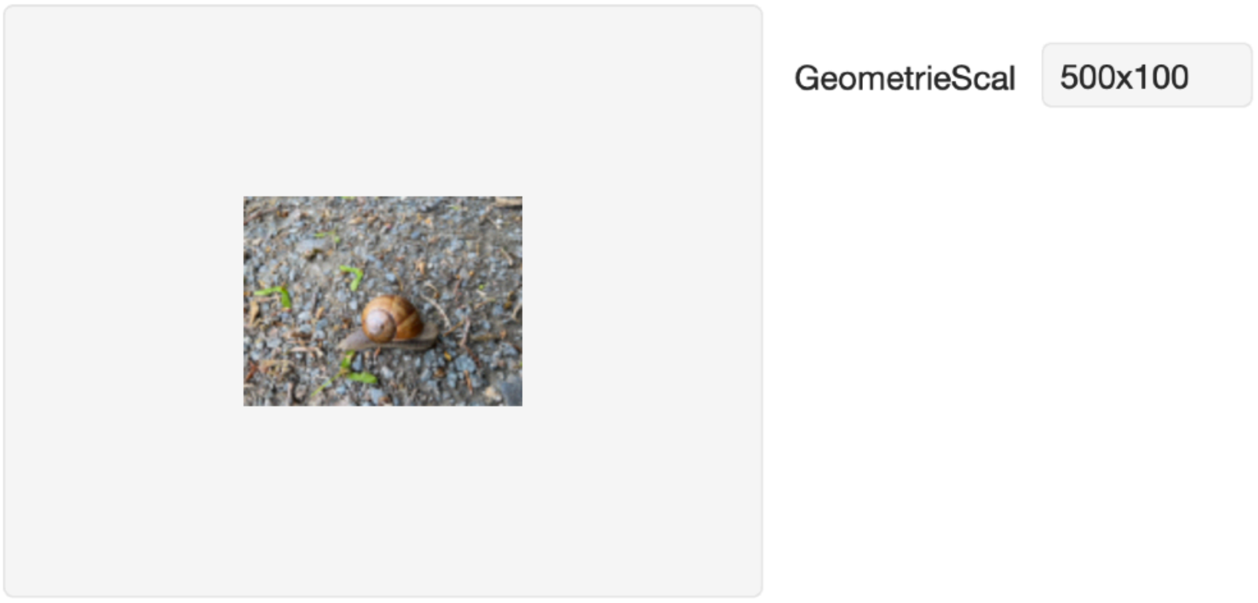
Welcome to the 5th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

5 of 24

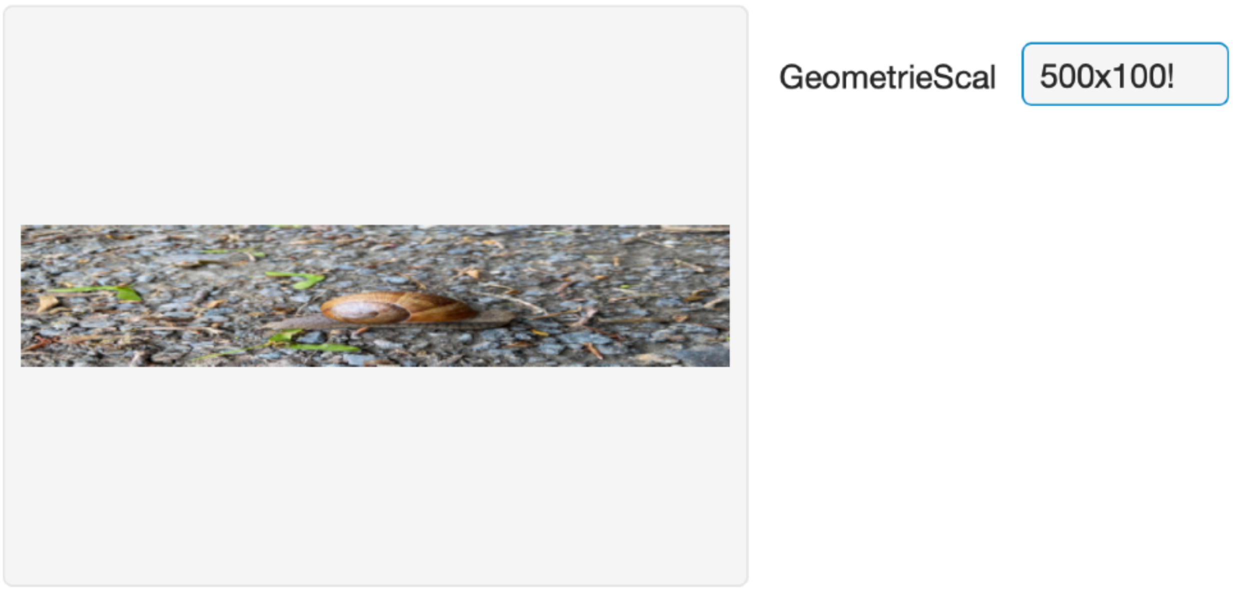
Today I would like to show you how you can scale an image according to your wishes. For this we use the GMImage.Scale function. In the parameters you have to enter the reference and a geometry. The geometry can look like the geometry we already know (width x height), but we can now also define other things in the geometry and are not limited to a fixed size of the image. So that you can try it yourself we have created a field in the sample file (This will be available on December 24) in which you can enter the geometry to test it. The script looks like this. We always load a fresh image from a file and then scale it with the function. In our container, the image is then scaled, we save the image there as usual and release the reference again.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromFile"; "/
Users/sj/Desktop/abc.png") ]
Set Variable [ $r ; Value: MBS( "GMImage.Scale"; $GM;
GraphicsMagick Advent::GeometrieScal ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```

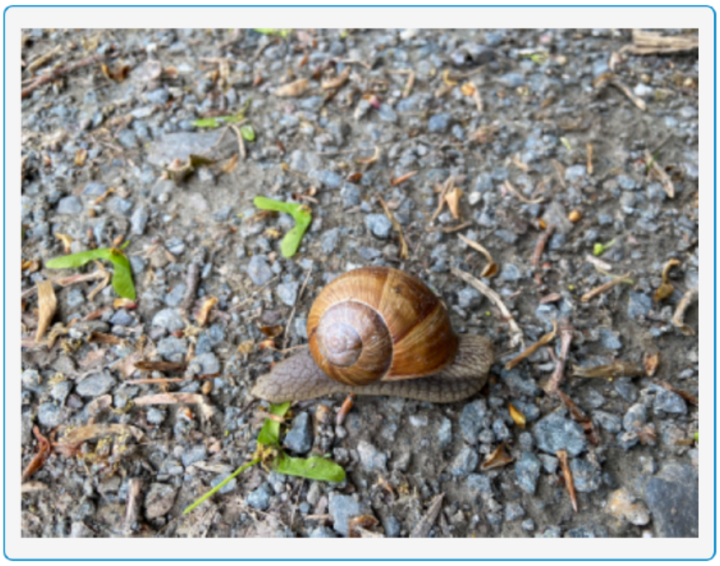
Let's first try the geometry we already know. I have entered 500x100 in the text field. The image is then scaled so that the smaller of the two values is assumed, so the image gets a height of 100 and the width is scaled proportionally. For example, if we were to specify 50 for the width and 100 for the height, then the image would adjust to the 50 and scale the height proportionally.


GeometrieScal 500x100

What do we do now, if we really want to bring the picture exactly to the size, without keeping the proportions? In this case we add an exclamation mark at the end. This means that the image will be scaled exactly to this size.

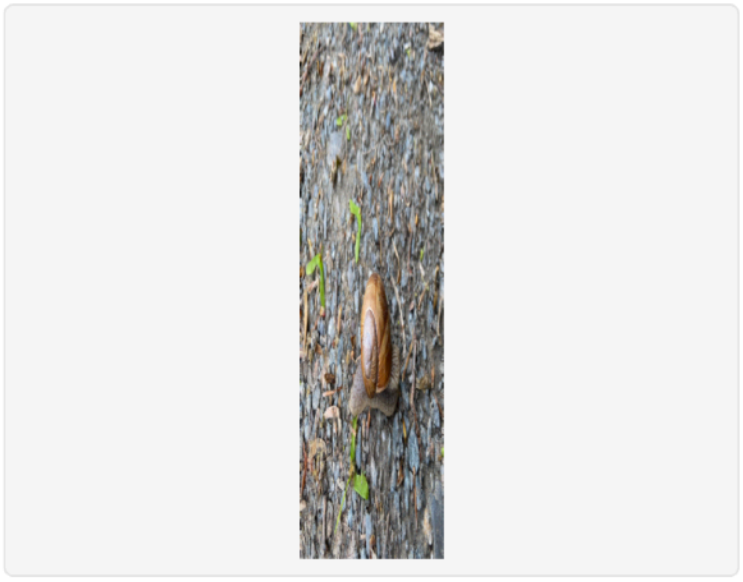

GeometrieScal 500x100!

We can work not only with fixed values but also with percentage values. For example, if we want the side lengths to be only 10% of the actual lengths, we enter 10% here. Both sides are then only 1/10 of the actual size. If we have an image with the dimensions 4032x3024, the image will be 403x302. With one value the proportions are kept.



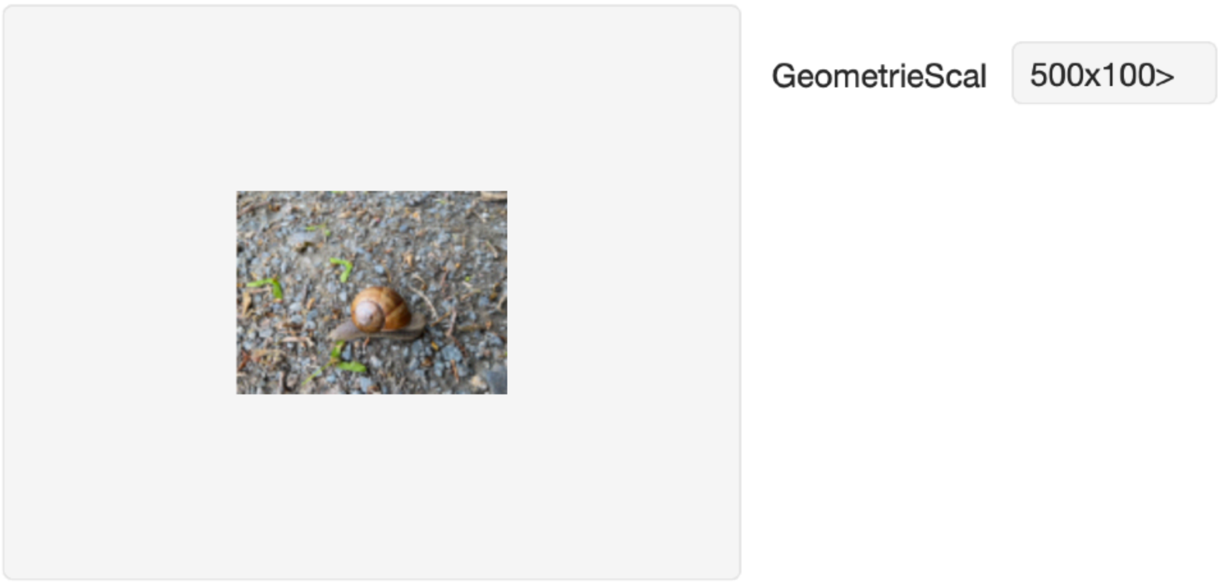
GeometrieScal 10%

However, we can also specify two values, e.g. if the image should have 10% of the original width and 50% of the original height. In this case, the proportions are not kept.



GeometrieScal 10x50%

We can also specify that an image should only be scaled if it is larger or smaller than the specified geometry. For this we specify < and> at the end. If the image should be scaled to an image if it is larger than the specified large, then we specify a > sign. If the image should be scaled when it is smaller than the specified size, we take a < sign. Our original is 4032x3024 so the image will be scaled if we define the geometry as: 500x100>.



GeometrieScal 500x100>

I hope your pictures get into shape with this multifaceted function. Be there again tomorrow when everything revolves around the image again 😉

GraphicsMagick in FileMaker, part 6

6 of 24

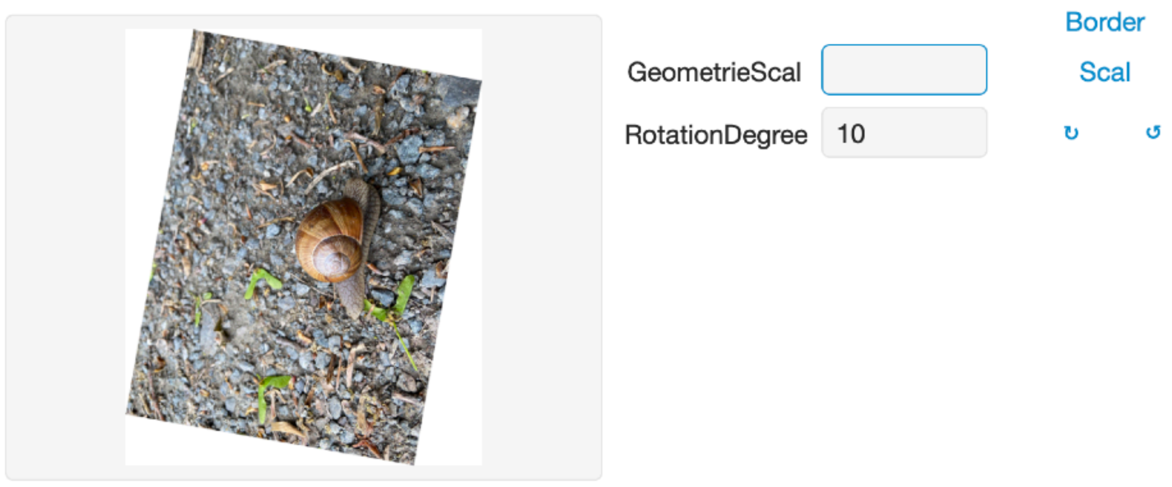
Welcome to the 6th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today we want to rotate the image and for this we use the function GMImage.Rotate. In the parameters we first specify the reference and then the number of degrees by which we want to rotate the image. The image will be rotated clockwise if the degree is positive and counterclockwise if the degree is negative. In our example file, I have included two buttons that rotate the image clockwise and counterclockwise. Both buttons call the same script. When pressing the buttons, different script parameters are passed, so we know which button was pressed. Clockwise passes the 0 and counterclockwise passes the 1. By how many degrees we want to rotate the image we specify in the appropriate field. The script looks like this:

```
Set Variable [ $direction ; Value: Get(ScriptParameter) ]
If [ $direction=1 ]
     Set Variable [ $degrees ; Value: Abs ( GraphicsMagick
Advent::RotationDegree )*-1 ]
Else
     Set Variable [ $degrees ; Value: Abs ( GraphicsMagick
Advent::RotationDegree ) ]
End If
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.Rotate"; $GM;
$degrees ) ]
```

```
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```

So we first get the parameter that gives us the direction. If it is a 1 then we get the amount of degrees that is in the field and multiply it by -1 since the button was pressed counterclockwise. We take the amount so that in case the user entered a negative number we would still rotate counterclockwise. A negative number multiplied by -1 would otherwise result in a positive number and we would rotate clockwise again. If the script parameter was not 1, it can only be a zero or the script was started otherwise and in these two cases we want to rotate clockwise. Before we can rotate the image, we must first load the image from the container into a reference and then we can apply the rotation to it. As we already know, the image is then saved again and the reference is released.



GeometrieScal

RotationDegree 10

I hope you are looking at your pictures from the right angle now. I hope you have fun trying it out. Let's see tomorrow if our image fits as well as today.

GraphicsMagick in FileMaker, part 7

7 of 24

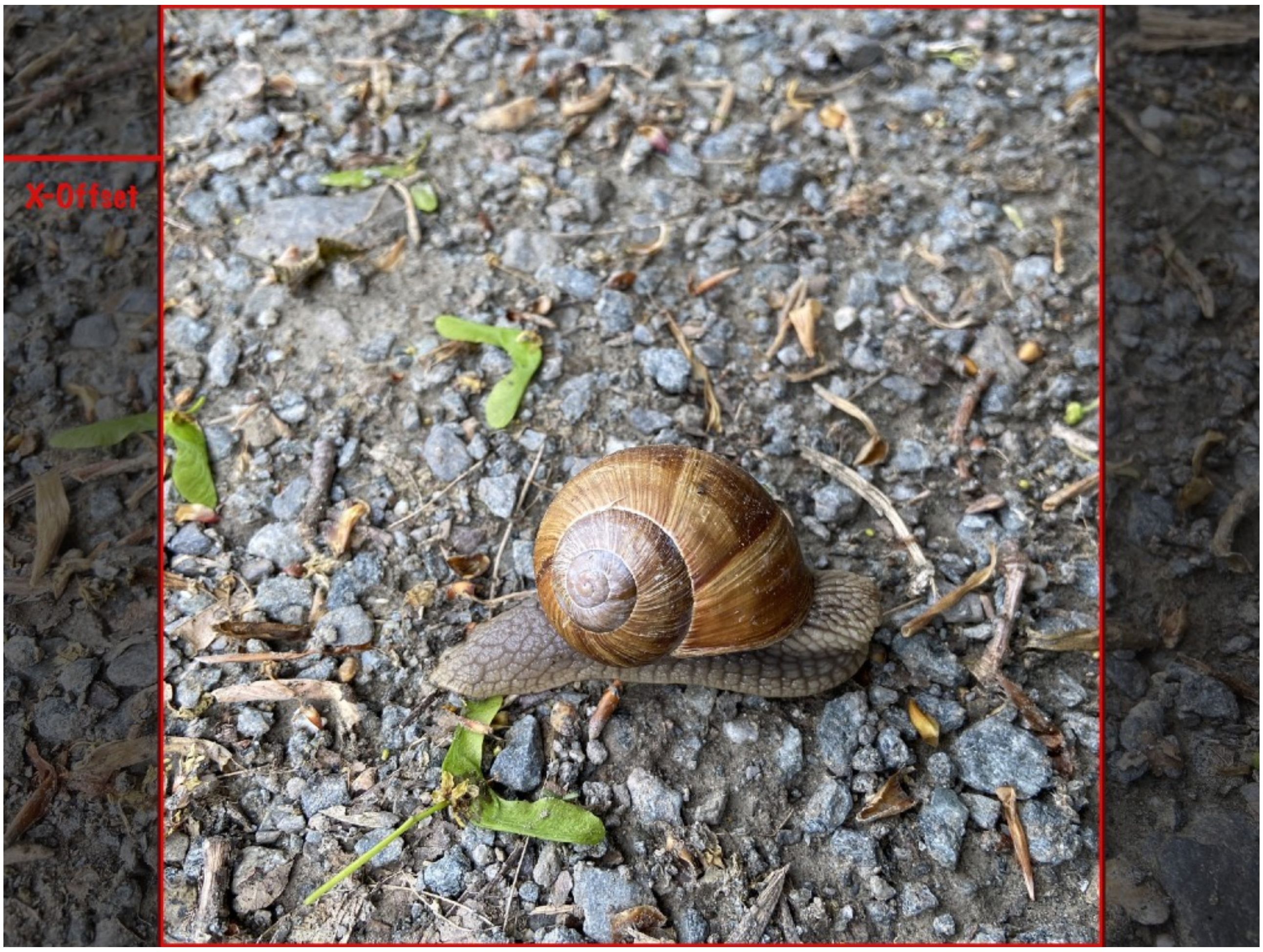
Welcome to the 7th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I want to show you how to crop an image. For this we use the GMImage.Crop function. This function returns an image section that you have to define in the parameters before. As with the other functions, we first specify the reference and then a geometry. The geometry describes the cut-out from the image. First we can define how big such a section should be - again in pixels. First the width and then the height which we separate from each other with an x. But now we don't only want to define the size of the section, but also the position. For this reason we have to set the offsets for X and Y in addition to the size information. The offsets determine the distance from the upper left edge. If we set a value for X we move away from the side edge, if we set a value for Y we move away from the top border. These two values can be appended with a plus. We will make an example now. We would like to crop the image so that it is square and shows the center of the image. Here we see the code:

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromFile"; "/
Users/sj/Desktop/abc.png") ]
Set Variable [ $Width ; Value: MBS( "GMImage.GetWidth";
$GM ) ]
Set Variable [ $Height ; Value: MBS( "GMImage.GetHeight";
$GM ) ]
If [ $Width>$Height ]
     # Landscape
     Set Variable [ $Size ; Value: $Height & "x" & $Height ]
```

```
    Set Variable [ $OffsetX ; Value: Round(($Width - 
$Height) / 2; 0) ]
    Set Variable [ $OffsetY ; Value: 0 ]
Else
    #  Portrait
    Set Variable [ $Size ; Value: $Width & "x" & $Width ]
    Set Variable [ $OffsetX ; Value: 0 ]
    Set Variable [ $OffsetY ; Value: Round(($Height - $Width
) / 2; 0) ]
End If
Set Variable [ $Geometry ; Value: $Size & "+" &$OffsetX& "+"
&$OffsetY ]
Set Variable [ $r ; Value: MBS( "GMImage.Crop"; $GM;
$Geometry ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```

First, we again get an image file as a reference and queries height and width. We must of course be able to distinguish whether the image is in portrait or landscape orientation. If it is a landscape format image, so the width is greater than the height, then we want to specify the size of the image as ***height***x***height***. Please note, that we use Round() here to make sure we have a whole number.



As the picture is wider, we now have to set the crop to the center, so we see in the picture that we have to move a certain number of pixels away from the edge. This is then the X Offset. The value for the X Offset results from the difference between width and height. Since we want to have the same distance on both sides, we divide this distance by two and get our value. Since we don't need a Y-Offset in this case, we set the value to 0. For a portrait image we have exactly the opposite case, we set the size with ***width***x***width***.The X offset is 0 and the Y offset is the difference between height and width divided by 2

The geometry is then composed as described:
Side dimension x ***Side dimension***+***OffsetX***+***OffsetY***

Then the crop function is applied, the image is placed in the container, and then the reference is released again.

Our picture is now square. I hope you enjoyed this door and see you tomorrow at the next door

GraphicsMagick in FileMaker, part 8

8 of 24

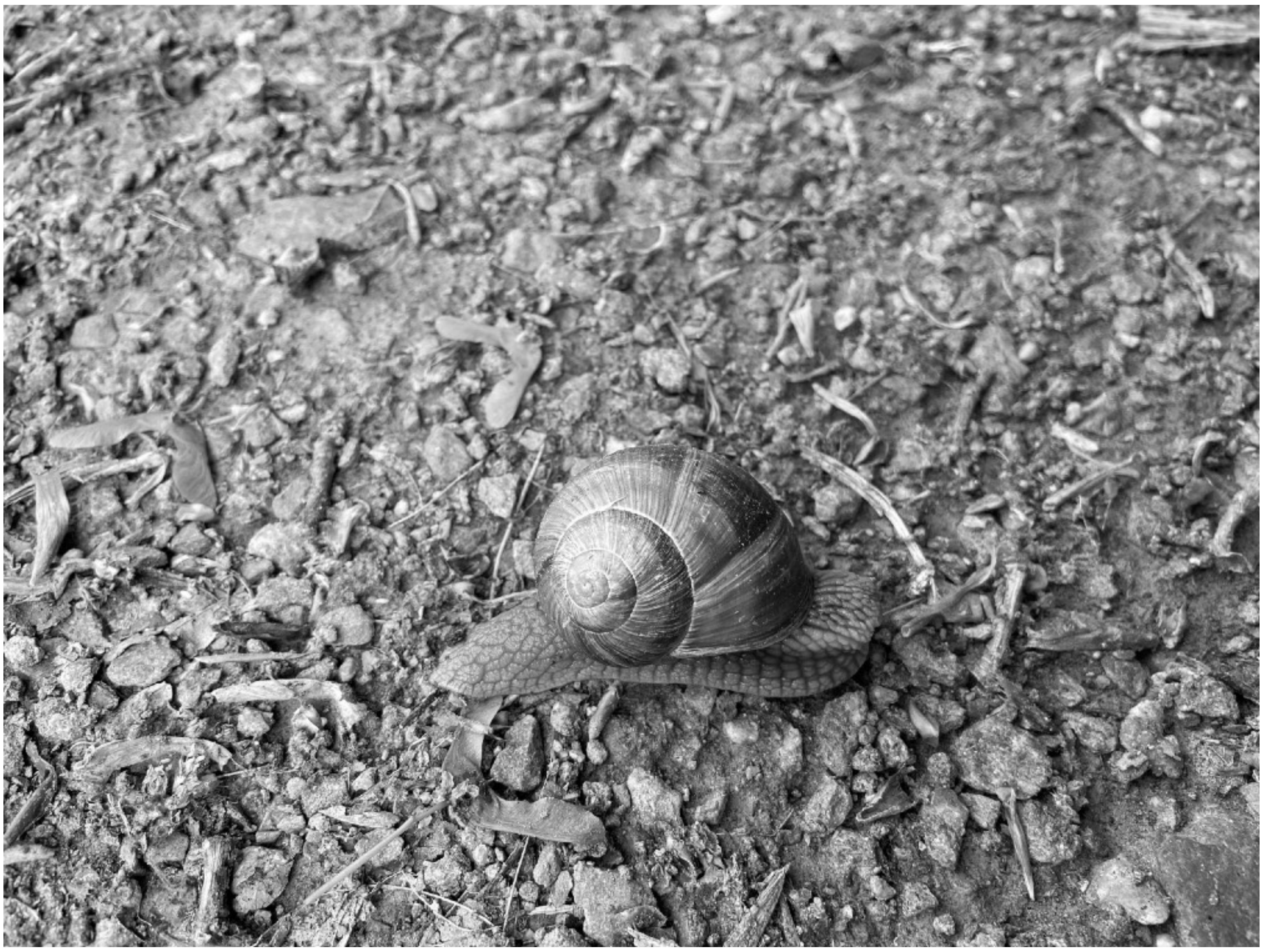
Welcome to the 8th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I would like to show you how to convert your image into a black and white image or how to display it in grayscale. Let's start with the grayscale. For this we use the function GMImage.SetType. This function sets the repression type of the image. We can specify the following values:

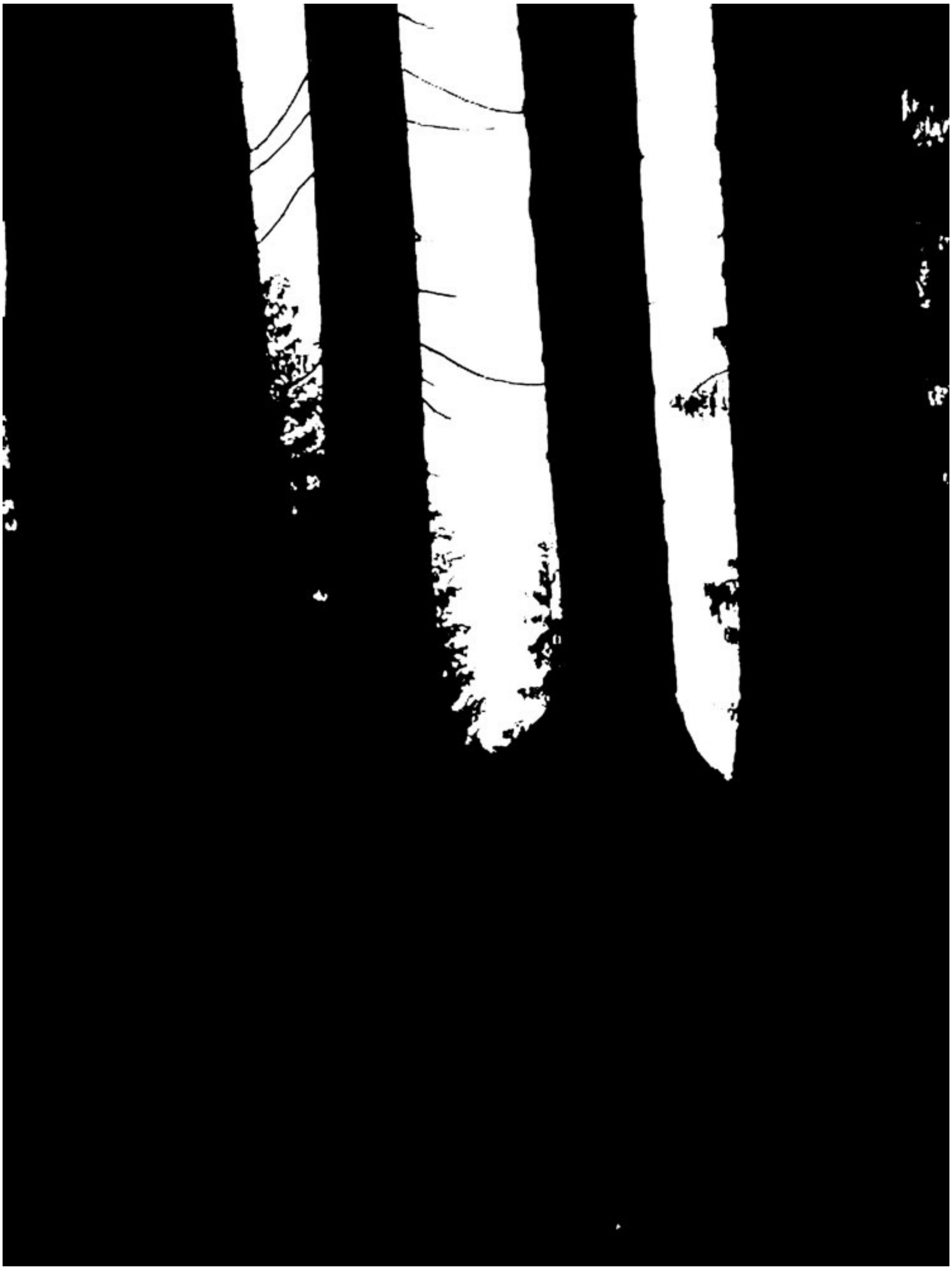
- UndefinedType = 0
- BilevelType = 1
- GrayscaleType = 2
- GrayscaleMatteType = 3
- PaletteType = 4
- PaletteMatteType = 5
- TrueColorType = 6
- TrueColorMatteType = 7
- ColorSeparationType = 8
- ColorSeparationMatteType = 9
- OptimizeType = 10

You can do very cool things with these types. We are only interested in the value two, because with it we can convert an image from a color image to a grayscale image. You can see the script for this here:

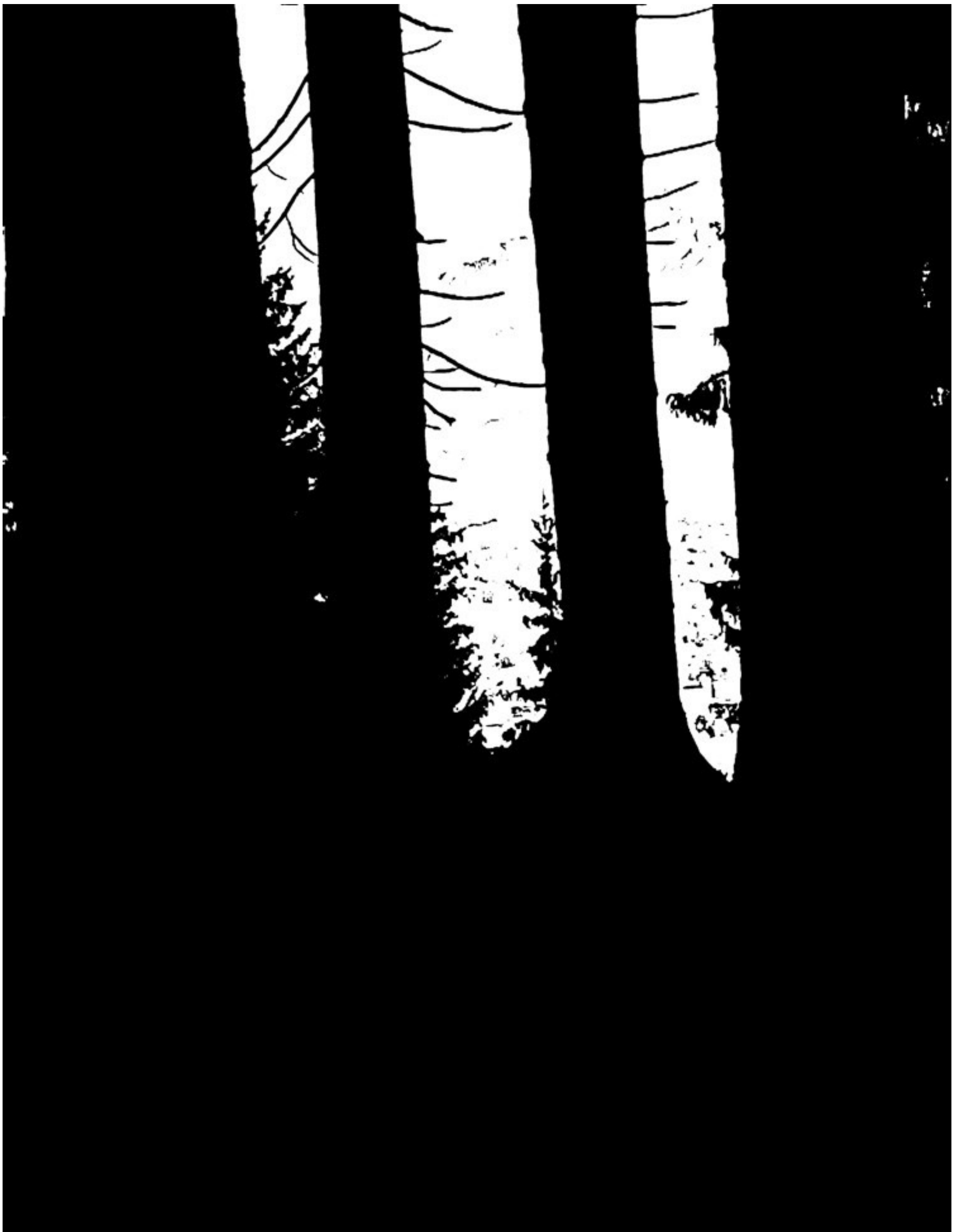
```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetType"; $GM;
2 ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```



But with this function we can't make the image only black and white, for that we need another function, the GMImage.Threshold function. If you want to convert an image into a black and white image then it is enough to pass the image reference to this function.



With this function we can set the value from which a color is interpreted as white. For this we have the optional parameter Threshold. The values of this threshold can be between 0 (everything is interpreted as white) and 65535 (everything is interpreted as black). Here you can see the image with a threshold of 50,000. The branches which are paler and cannot be seen at all with the default value are clearly displayed.



In the next image we have chosen a threshold of 12,500 here we see lighter areas in the forest floor.

Try it out and get creative. Hope to see you again tomorrow for the opening of the next door.

GraphicsMagick in FileMaker, part 9

9 of 24

Welcome to the 9th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I'm going to tell you a little bit about a function that is quite inconspicuous, but once provided me with great services. I got from a customer at the very beginning of my FileMaker time the order that I should sort out all pages from a scanned PDF file that are empty. I accepted the order and thought, that can't be difficult, there must be a simple function you can use to detect if the page is empty or not. What can I say it wasn't that easy and I was thinking for a long time about how to do it until I found the GMImage.AveragePixelValue function. This function gets the average color value of pixels in an image or a certain area which can be defined with optional parameters in the function.

```
...
Set Variable [ $AveragePixelValue ; Value:
MBS( "GMImage.AveragePixelValue"; $GM; 0; 0; $imageWidth;
$imageHeight ) ]
     If [ $AveragePixelValue   <  ,99 ]
          Set Variable [ $r ; Value:
MBS( "DynaPDF.AppendImagePages"; $pdf; $currentPicturePath )
]
...
```

Before inserting the pages as images into a PDF with DynaPDF, the images of the pages scanned by the scanner were available as container values, so I could simply load them into memory as reference. Of course you can use the GMImage.Threshold function again to convert an image to a black and white copy. The

GMImage.AveragePixelValue function returns a value between 0 and 1 that tells us if the image is more or less black or white. Of course, if you scan with a normal scanner, not everything is white when the page is blank. It could be a bend of the paper, it could be because the paper had a yellow tone and so some pixels are interpreted as black pixels. So we have to define a limit from when we say the page is blank or the page is printed. The best way to find such a threshold is to look at the values of scanned sample pages. The more white is on the page the higher is the value. I chose a value of 0.99, which still allows some wrinkles, but if there is a certain amount of text on the page, the page will not be sorted out.

After I finished this check the image was added to a DynaPDF document with DynaPDF.AppendImagePages. When using the GMImage.AveragePixelValue function you have to make sure that the image is in RGB format. For this reason it is recommended to use the GMImage.SetType function which is passed the type 6 for TrueColorType in the parameters. We have already seen the use of this function yesterday in connection with a grayscale image.

I hope the function can help you, as it helped me back then. I would be happy if we meet again tomorrow and make you text confident.

GraphicsMagick in FileMaker, part 10

10 of 24

Welcome to the 10th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today we want to write a text in our image. For this we use the function GMImage.Annotate. As in most functions, we first specify the reference, then the text follows. Optionally we can specify where exactly the text should be, the area we specify in the same way as in door 7 for the image section. First the size in pixels and then the x and y offset (*widthxheight+offsetX+offsetY*). We can also set the text alignment. Last but not least, we can even specify that the text should be rotated by a certain number of degrees. If you want you can set not only the text, but also the font, the color of the text, the font size, the color of the text border and the thickness of the text border. For these individual settings we have one function each. It is best if we look directly at the sample code together.

```
Set Variable [ $GM ; Value: MBS( "GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetFont"; $GM;
"Georgia" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetFillColor";$GM;
"#FF0000" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetLineWidth"; $GM;
2) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetStrokeColor";
$GM; "#287233") ]
```

```
Set Variable [ $r ; Value: MBS( "GMImage.SetFontPointsize";
$GM; 100 ) ]
Set Variable [ $Width ; Value: MBS( "GMImage.GetWidth";
$GM ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Annotate"; $GM;
"Merry Christmas"; $Width&"x200+0+20"; "CenterGravity"; -10)
]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

We want to display a text centered on the upper part of an image. First we set the font we want to use for the text. For this we can use the function GMImage.SetFont. In addition to the reference, we specify the name of the font here. We can also set the color of the font. The color of the font consists of two components: the fill color, which we can specify with GMImage.SetFillColor by specifying the reference and the color, in this case red, and the line color. The line color is the border of the font, here dark green. We can set this in the function "GMImage.SetStrokeColor. Also we can set the thickness of the border with GMImage.SetLineWidth. To set the size of the font we use the GMImage.SetFontPointsize in our example the font is 100 pt. Before we write to the image we determine the width of the image, because we define our text field in the function GMImage.Annotate over the whole width of the image. In addition, the writing area should have a height of 200 pixels. Since we have defined the width of the text box across the entire width of the image, we don't need an x-offset here, but we do want to move 20 pixels away from the top edge. We set the text aligment to CenterGravity so that we display the text centered. We also want the text to run at a little bit of an upward slant. For this reason we rotate the text 10 degrees counterclockwise. For this we have -10 in our script and we set the text to Merry Christmas. Our result now looks like this:

Now you can give your pictures a font I hope you enjoy it. See you tomorrow

GraphicsMagick in FileMaker, part 11

11 of 24

Welcome to the 11th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

The GraphicsMagick also provides many effects that we can apply to our images. In the Advent calendar we would like to introduce a few of these effects. Today I will show you how to add a noise effect to your image. So we want to bring in a noise effect. For this we have two functions that we can use. We have the simpler function GMImage.AddNoise which adds a noise to the image which we can define in the parameters. We have a total of 6 noise types that we can specify.

- UniformNoise = 0
- GaussianNoise = 1
- MultiplicativeGaussianNoise = 2
- ImpulseNoise = 3
- LaplacianNoise = 4
- PoissonNoise = 5

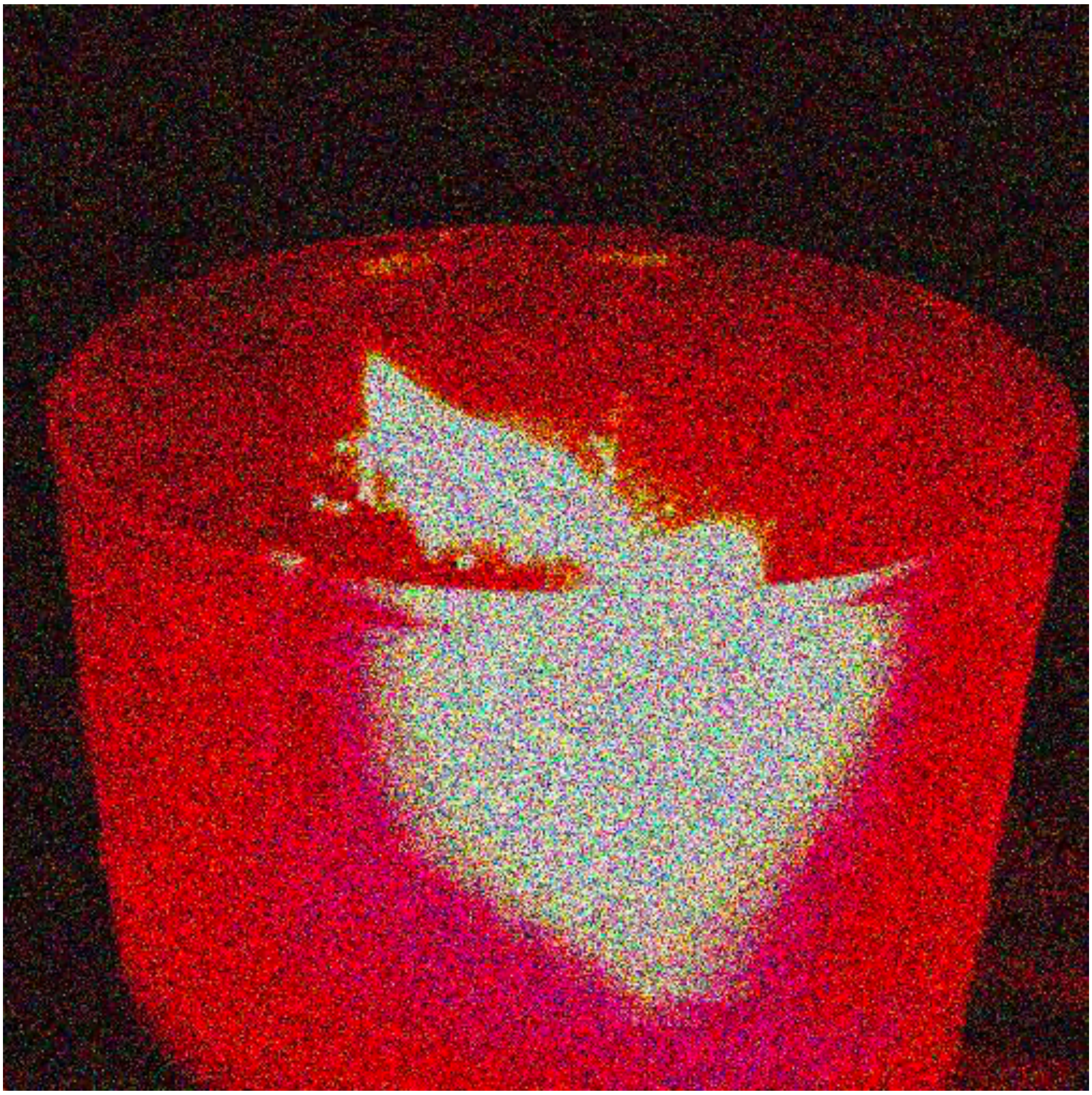
In the parameters we specify the appropriate number. In this script line, for example, the noise type ImpulseNoise was selected.

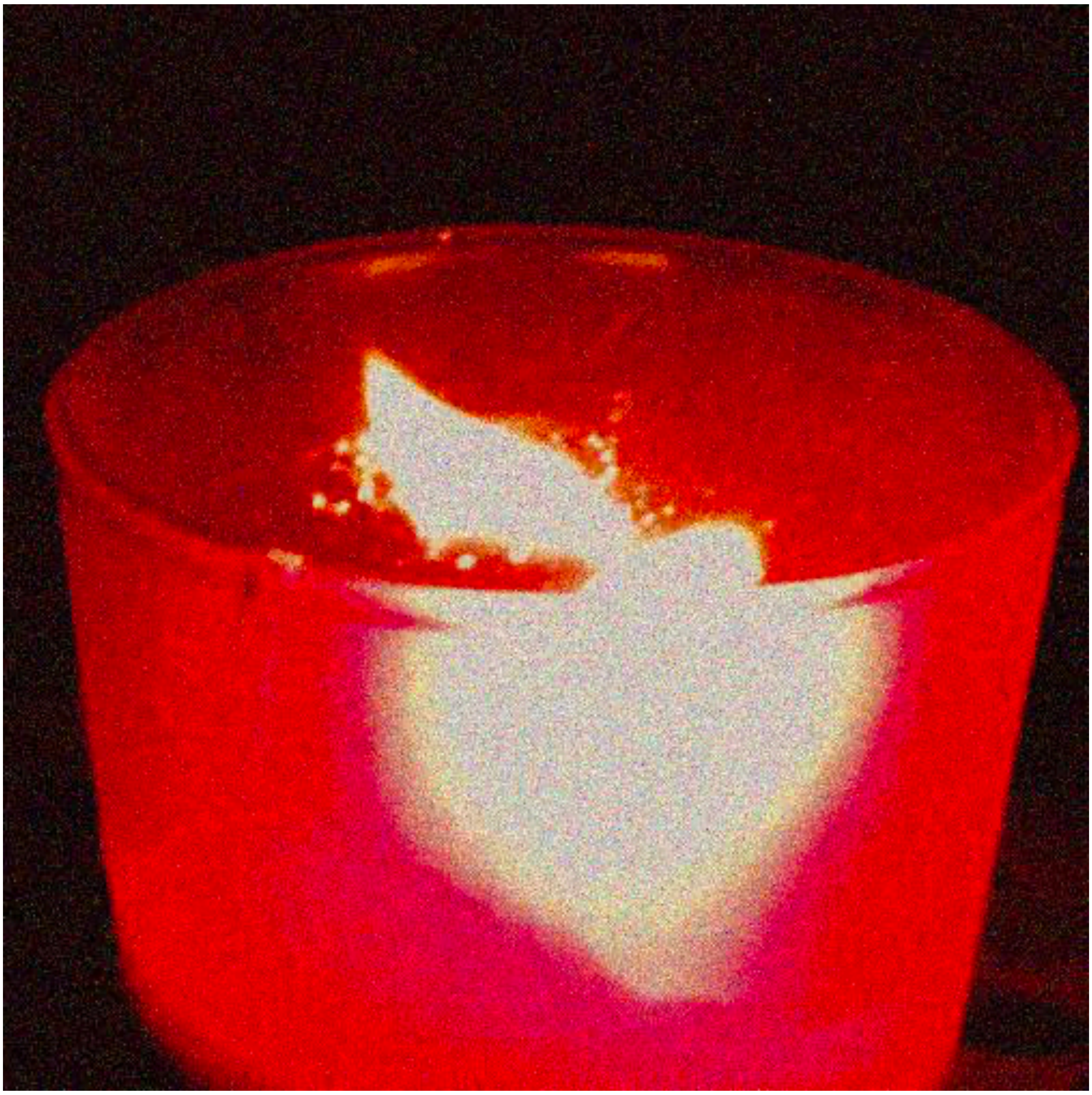
```
Set Variable [ $r ; Value: MBS( "GMImage.AddNoise"; $GM;
3) ]
```

On the following images a couple of the mentioned noise methods were applied 3 times in a row.

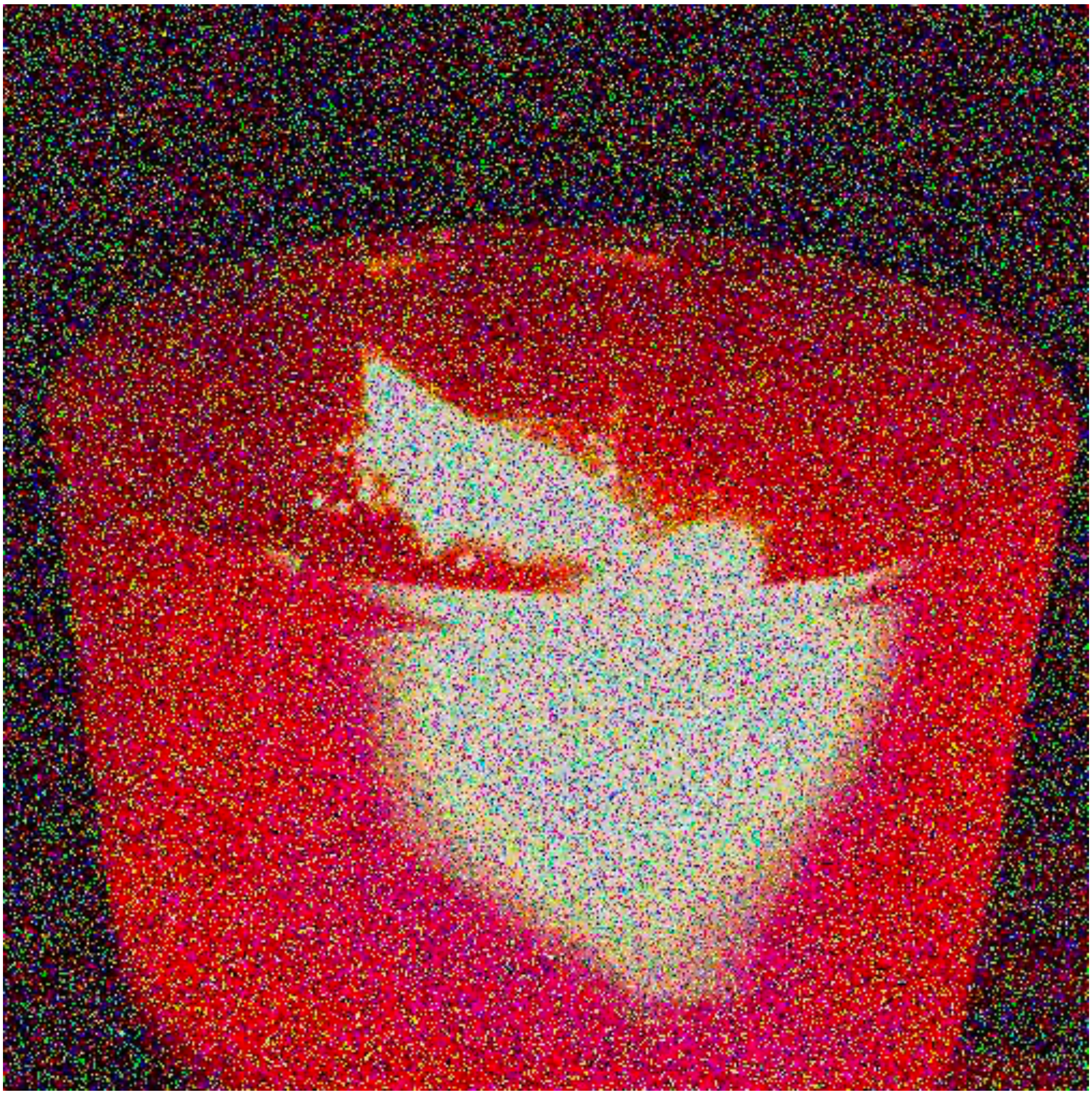
First of all here is the original image

The Gaussian noise mode was applied to this image

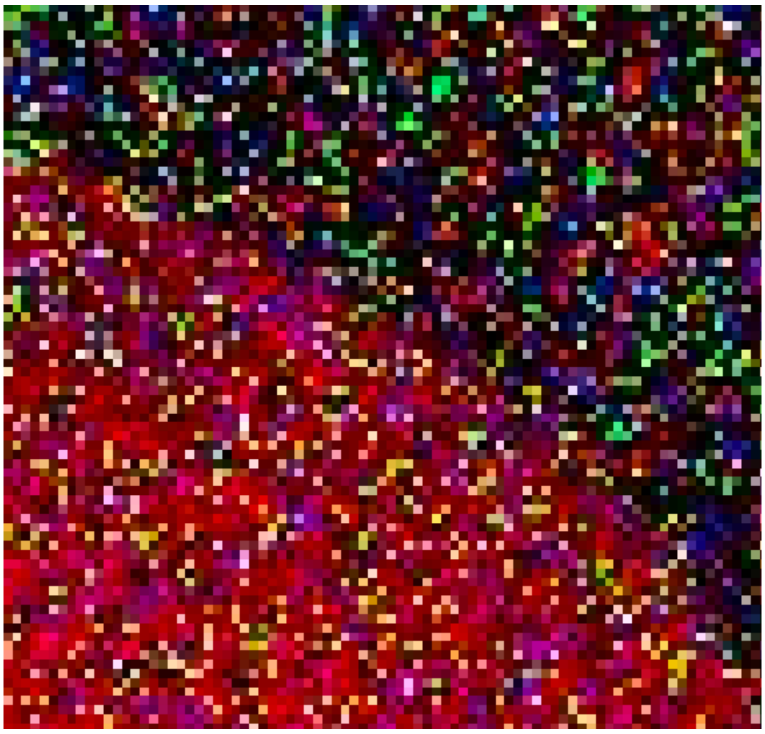

This image has the Laplacian noise mode applied



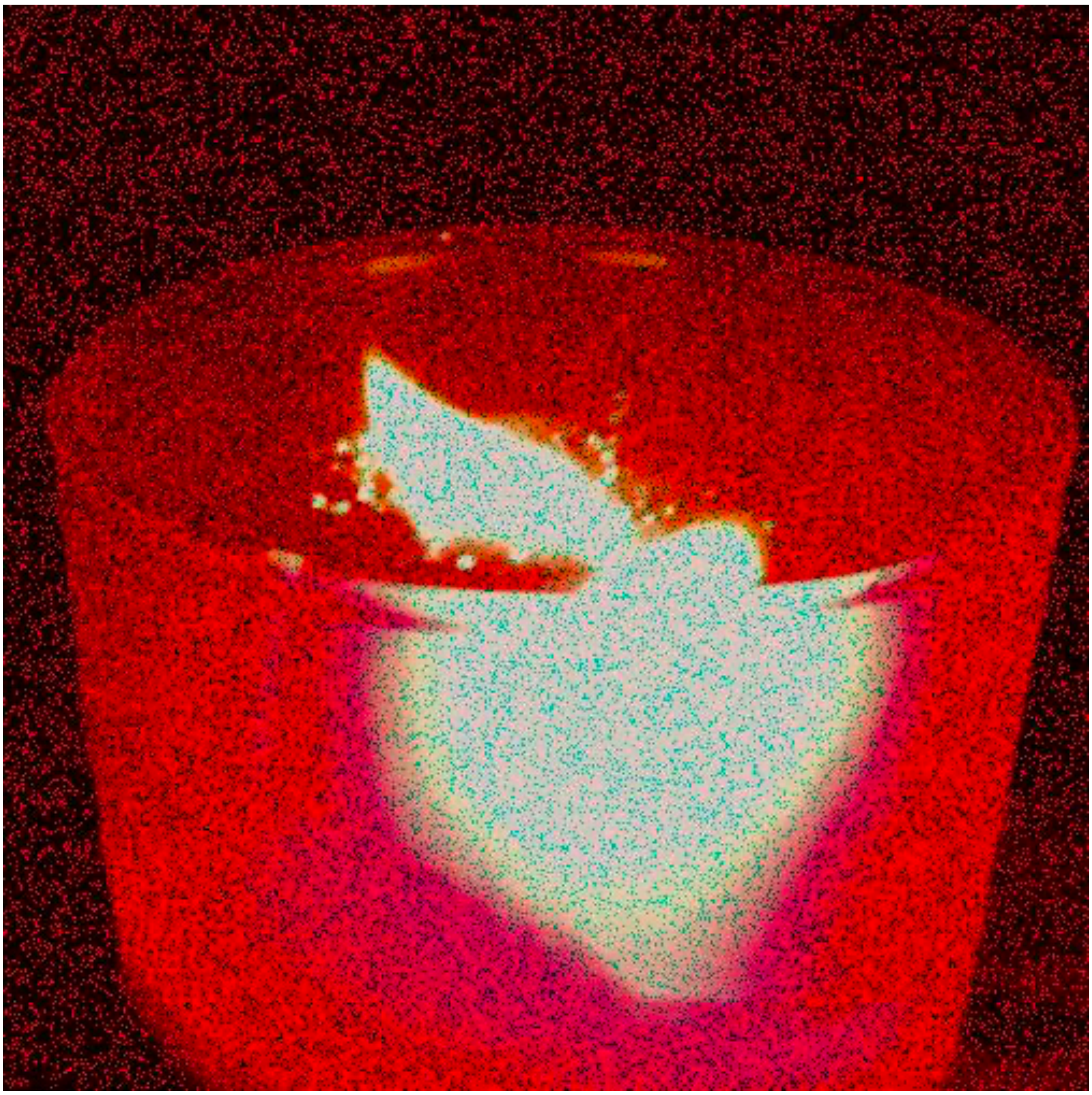
And for this image we use the impulse noise



But we can specify the noise even further. Noise is pixels that normally do not belong to the image.



Here we intentionally apply them to an image as an effect. In photography they are image noise caused for example by photo sensors. You can further define the noise with the function GMImage.AddNoiseChannel, for example you can show only the noise pixels from the Red Channel. Thus we see here on the black only red blinkers.



In the GMImage.AddNoiseChannel function, the noise selection is still the same, but we also specify a channel in the function. You can choose from these channels:

- RedChannel = 1
- CyanChannel = 2
- GreenChannel = 3
- MagentaChannel = 4
- BlueChannel = 5
- YellowChannel = 6
- OpacityChannel = 7
- BlackChannel = 8

- MatteChannel = 9
- AllChannels = 10
- GrayChannel = 11

The code then looks like this for RedChannel and ImpulseNoise noise:

```
 Set Variable [ $r ; Value: MBS( "GMImage.AddNoiseChannel"; 
$GM; 1; 3 ) ]
```

I hope you enjoyed this and we will meet again tomorrow

GraphicsMagick in FileMaker, part 12

12 of 24

Welcome to the 12th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

The GraphicsMagick also provides many effects that we can apply to our images. In the Advent calendar we would like to present a few of these effects. Today I will show you the effect that turns your image into an oil painting. For this we use the function GMImage.OilPaint. Here we have the original image that we want to change.

If we pass only the image reference to the GMImage.OilPaint function,

```
 Set Variable [ $r ; Value: MBS( "GMImage.OilPaint";$GM) ]
```

our image will look like this.

Small circles are drawn around the pixels. This creates this oil dab effect. But you can choose the brush thickness, that is the radii of the circles, yourself.

```
 Set Variable [ $r ; Value: MBS( "GMImage.OilPaint";$GM;
30 ) ]
```

Here you can see the effect in the different strengths 5, 10 and 30.

The effect is of course also very dependent on the image size due to the radius. The image with double side length with a value of 10 looks different like value 10 in a smaller image and is more similar to the small image with a radius of 5.

So, for example, make a pretty big oil painting from a photo of you and give it as a gift to your family. I wish you a lot of fun with it.

GraphicsMagick in FileMaker, part 13

13 of 24

Welcome to the 13th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

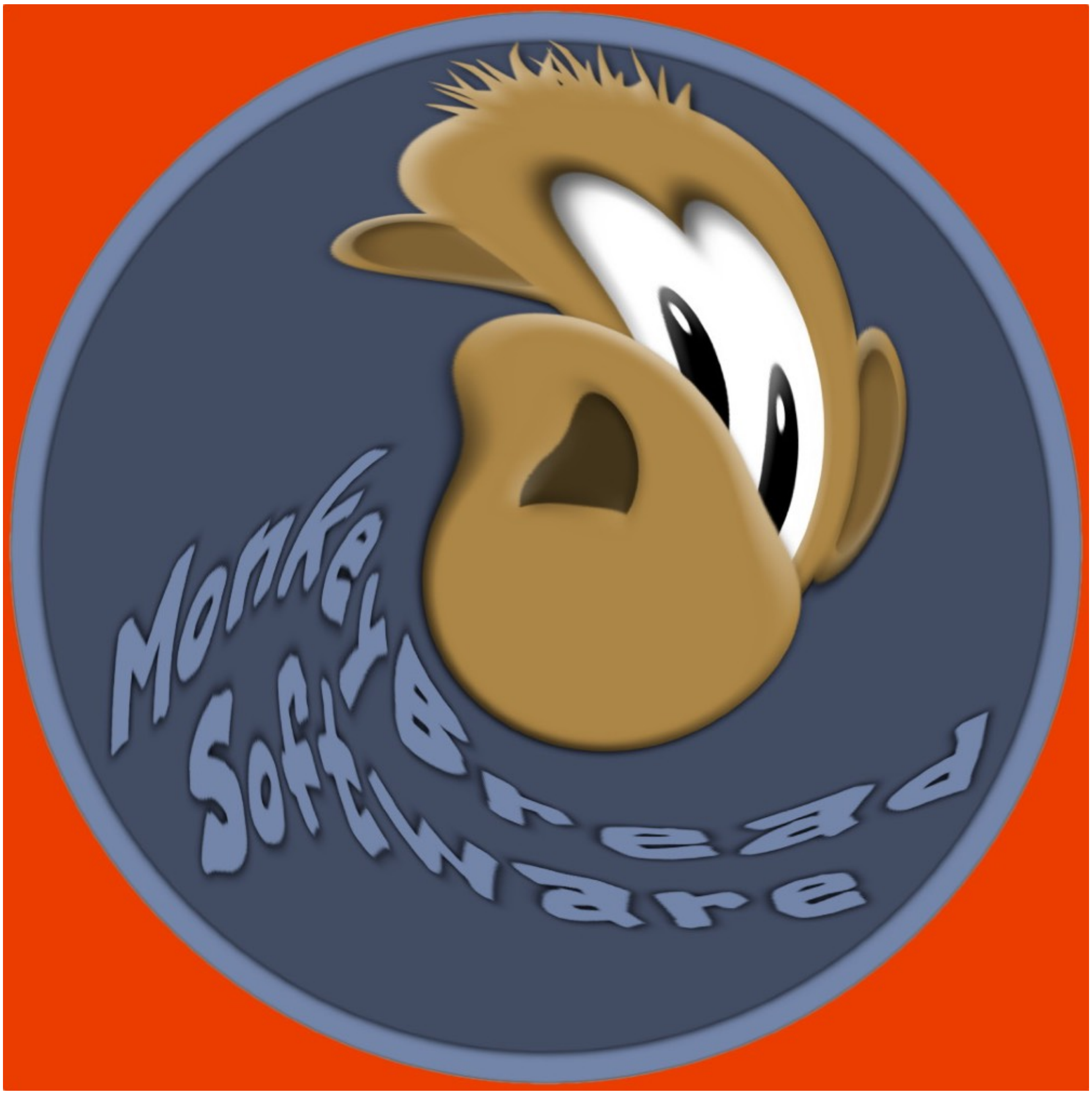
The GraphicsMagick also provides many effects that we can apply to our images. In the Advent calendar we would like to introduce a few of these effects. Today I will introduce you to the Swirl effect. With this effect you can turn the center of your image. We use the function GMImage.Swirl for this purpose. In the parameters we can specify the reference and the angle by which the image should be rotated.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.Swirl"; $GM;
-90 ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

When we specify a positive value we create a counterclockwise rotation.

Monkeybread
Software

In case of a negative value, a clockwise rotation.



The effect is reversible. This means that if you first use the function with a value of +90 degrees and then apply the function to the image with -90 degrees, we will see the original image again. The function can also be applied several times in succession.

I wish you a lot of fun turning the head of your portraits

GraphicsMagick in FileMaker, part 14

14 of 24

Welcome to the 14th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

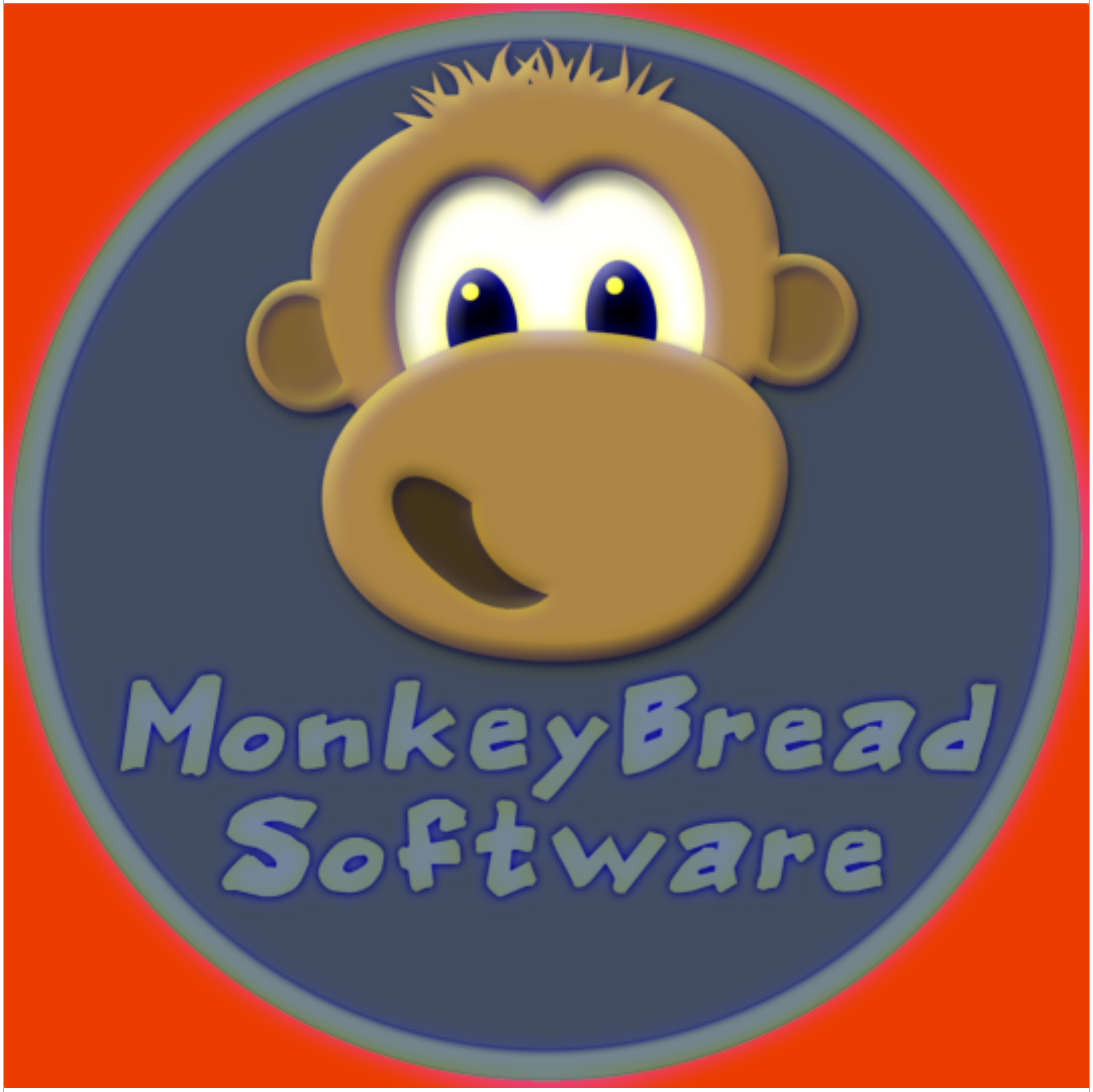
The GraphicsMagick functions also provides many effects that we can apply to our images. In the Advent Calendar we would like to introduce a few of these effects. Today I will introduce you to the Blur effect. With this effect you blur the image. You can use the function GMImage.Blur for this purpose. We can pass three different parameters to this function. First of all again our reference, then the Radius of the Gaussian that means we specify the size of the radius from which we get our information for the respective pixel. We also specify the value Sigma, which is the standard deviation of the Laplacian.

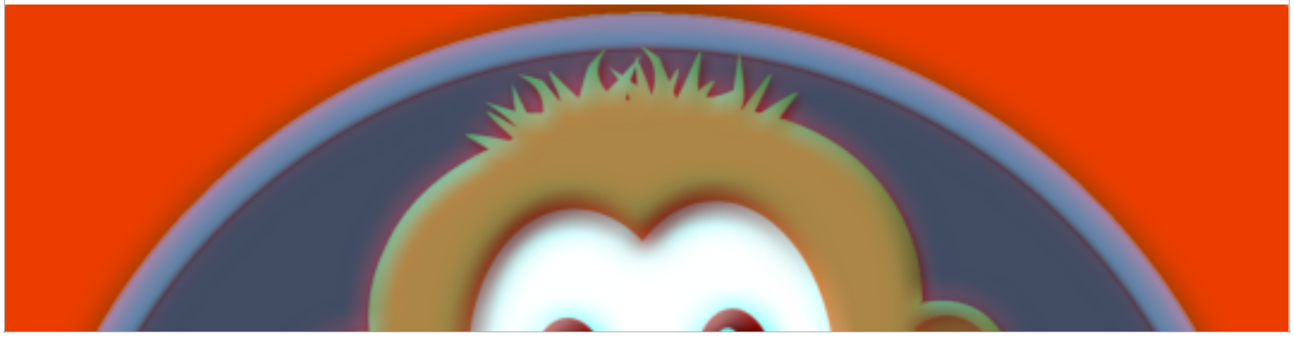
```
Set Variable [ $r ; Value: MBS("GMImage.Blur"; $GM; 50;
10) ]
```

The Blur effect is influenced by the combination of the last two parameters, so you can vary these two parameters to your taste. Here are examples of the variation of values.

However, we cannot only blur the image on all channels, but we can also use the individual channels in the GMImage.BlurChannel function to create a blur. To do this, we also specify the Channel Type parameter. It can be one of our RGB channels or CMYK channels or the BlackChannel.

Here you see the image once with **blue channel** and **radius = 10** and **sigma = 10** on the left.





MonkeyBread Software	MonkeyBread Software

GraphicsMagick in FileMaker, part 15

15 of 24

Welcome to the 15th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Yesterday I introduced you to the Blur effect. Today I will show you the opposite effect: Sharpen. This effect sharpens your image. We use the GMImage.Sharpen function. Again we have the same parameters as with the Blur function. First our reference, then the radius which indicates from which area we use the information for the effect and our sigma which describes the effect in more detail.

```
Set Variable [ $r ; Value: MBS( "GMImage.Sharpen"; $GM ; 10;
10 ) ]
```

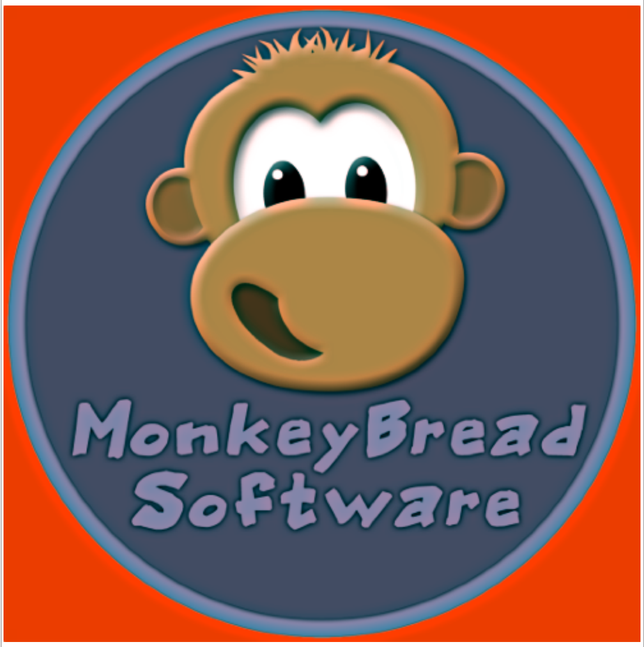
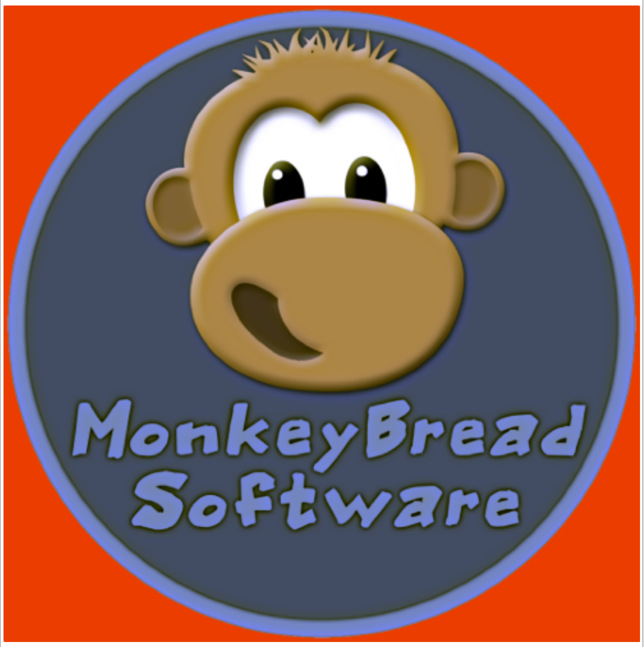
Here we see a few examples of how the effect can affect:

By the way, the Sharpen effect cannot undo the Blur effect. If I first apply a Blur effect and then apply a Sharpen effect with the same values for Radius and Sigma, the result is not the same image as before (left).

MonkeyBread
Software

Radius = 10
Sigma = 10

MonkeyBread Software Radius = 10 Sigma = 10	MonkeyBread Software Radius = 10 Sigma = 10

 **red Channel** **Radius = 10** **Sigma = 10**	 **blue Channel** **Radius = 10** **Sigma = 10**

GraphicsMagick in FileMaker, part 16

16 of 24

Welcome to the 16th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I'll introduce you to the GMImage.Channel function that allows us to extract individual channels from an image. We have already seen in some effects that we can use individual channels, whether we want to apply blur or if we want to give the image a noise with red pixels. Channels are a great thing. Let's take a closer look at them. In the function GMImage.Channel we first specify the reference in the parameters and then the ChannelType. Each channel type is assigned a number which must be specified in the parameters. Here we see a list of channels that are available to us:

- RedChannel = 1
- CyanChannel = 2
- GreenChannel = 3
- MagentaChannel = 4
- BlueChannel = 5
- YellowChannel = 6
- OpacityChannel = 7
- BlackChannel = 8
- MatteChannel = 9
- AllChannels = 10
- GrayChannel = 11

Channels 1,3 and 5 belong to the RGB color space, so that we can successfully apply the GMImage.Channel function to the images, they must be in the RGB color space or first converted with GMImage.SetColorSpace.

```
Variable setzen [ $r ; Wert: MBS("GMImage.SetColorSpace"; 
$GM; 1) ]
Variable setzen [ $r ; Wert: MBS( "GMImage.Channel"; $GM; 
3 ) ]
```

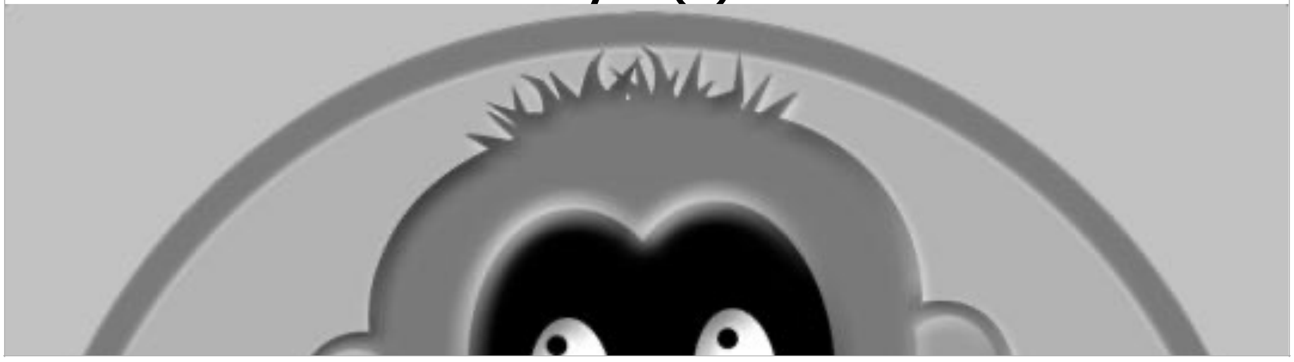
Here you can see our logo in the individual RGB channels

Conversion to the correct color space is needed for the CMYK color space for channels 2,4,6 and 8.

```
Variable setzen [ $r ; Wert: MBS("GMImage.SetColorSpace";
$GM; 10) ]
Variable setzen [ $r ; Wert: MBS( "GMImage.Channel"; $GM; 11
) ]
```

MonkeyBread
Software

Cyan (2)



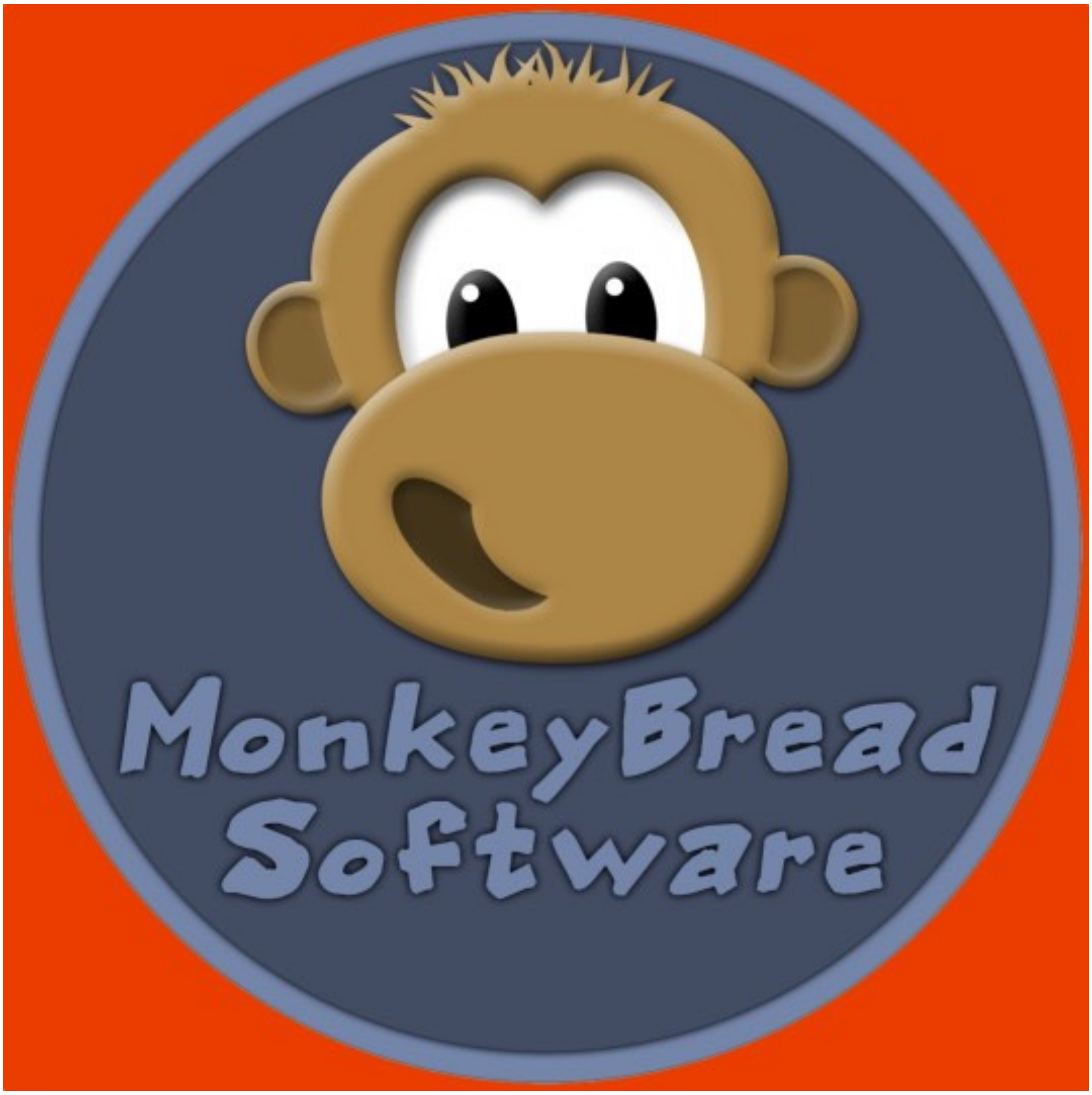
Cyan (2)	Magenta (4)

Yellow (6)	Black (8)

GraphicsMagick in FileMaker, part 17

Welcome to the 17th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

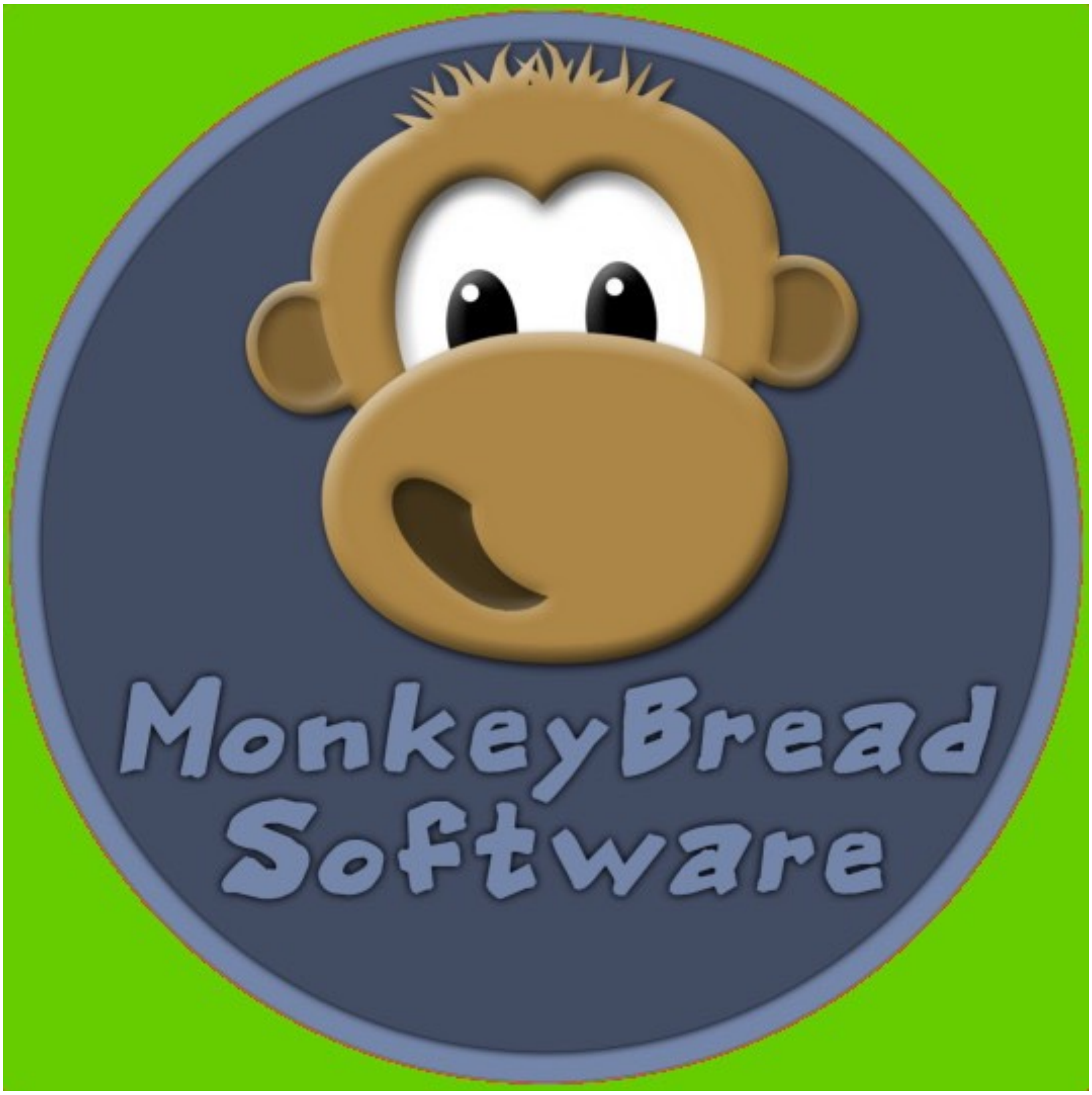
17 of 24



Today it will be really colorful, at least today we want to try some backgrounds for our logo. Our original image has a red background.

Now we want to replace it with another color. For this we use the GMImage.ReplaceColor. In this function we have again our reference in the parameters and then we specify the color that should be replaced. We also specify a second color to replace the color we just defined. We can also specify a factor for how much the color value that should be replaced can deviate so that a replacement is performed. But for now we want to ignore this optional parameter. In our image with the logo the background has the color red. To be more precise #EB3D00 as hexadecimal. We don't have anything else on the image that is red and thus would be replaced as well. Now we want to replace the background with green, so we specify #66CD00 as another color parameter. This color describes a toxic green

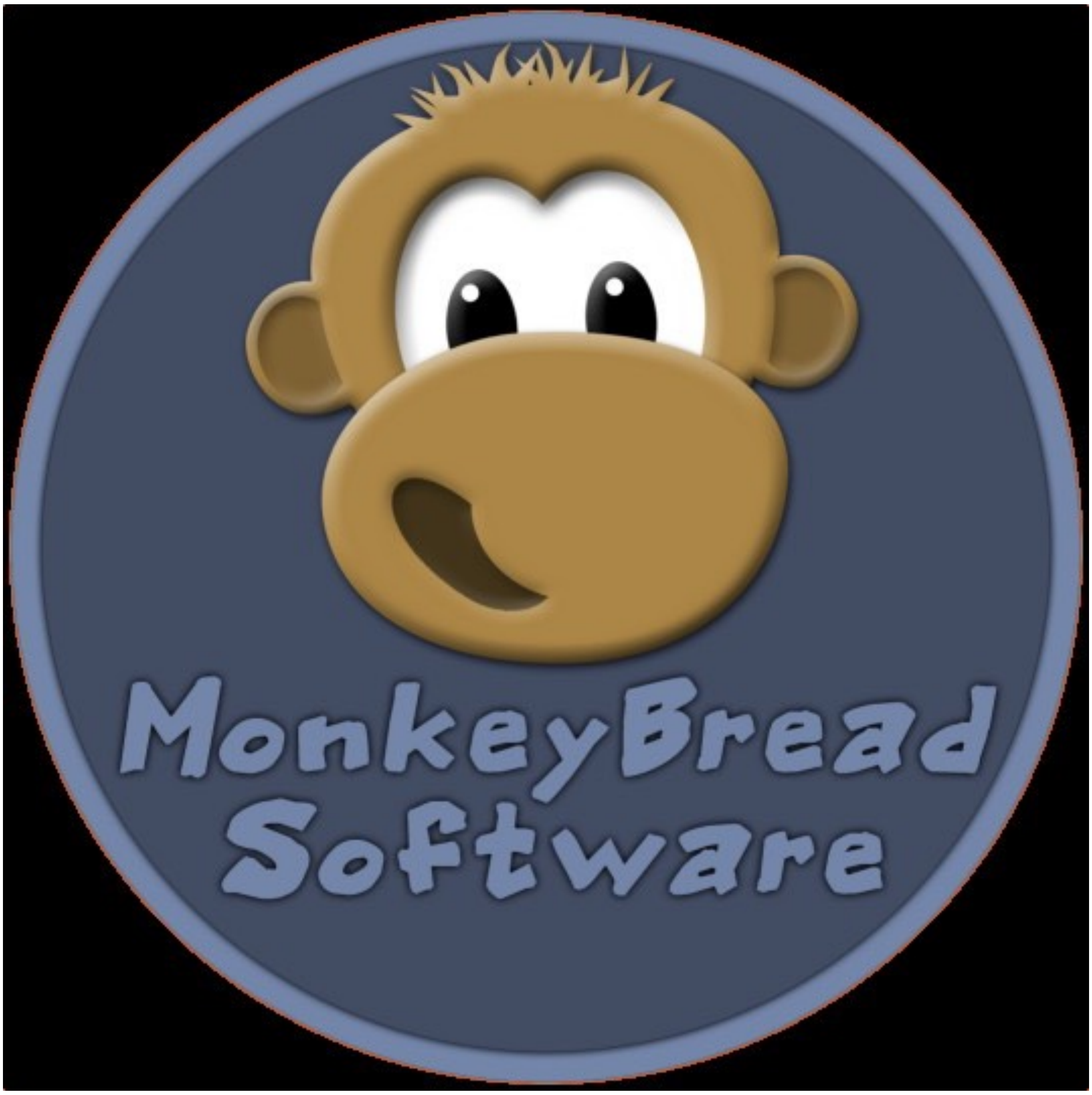
```
 Set Variable [ $r ; Value: MBS( "GMImage.ReplaceColor";
$GM; "#EB3D00"; "#66CD00") ]
```



Of course we can do this with any colors

MonkeyBread Software
MonkeyBread Software
MonkeyBread Software
MonkeyBread Software

If we give the whole thing a black background we can see it best. Around the logo a very fine red border remains.



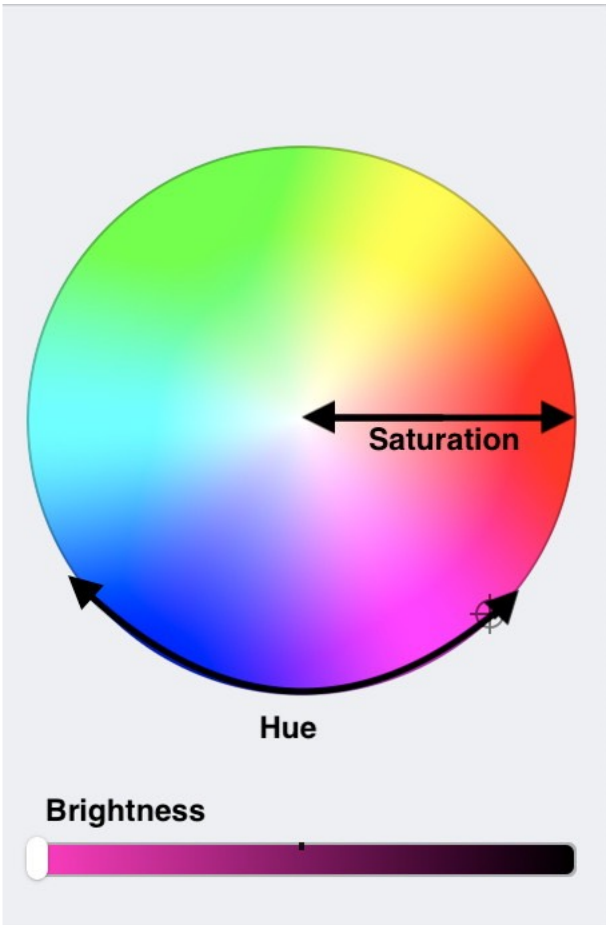
The reason is that the logo pixels mix slightly with the color of the background to create a smooth transition in the image. Since these pixels are not exactly the color we specified in the function, they are not replaced the way the function is currently used. But we can change this by defining a deviation factor for the exchange color value. This factor must be multiplied by 257 since version 9 of the plugin uses 16-bit values. So if we specify a tolerance of 1*257, the specified color in your channel can deviate by 1. To make the red border disappear we choose a 50*257

```
 Set Variable [ $r ; Value: MBS( "GMImage.ReplaceColor"; 
$GM; "#EB3D00"; "#000000"; 50 * 257 ) ]
```

MonkeyBread Software

I hope you enjoyed it again this time and now you bring a little buzz in your pictures we'll see you again tomorrow. See you soon.

GraphicsMagick in FileMaker, part 18



18 of 24

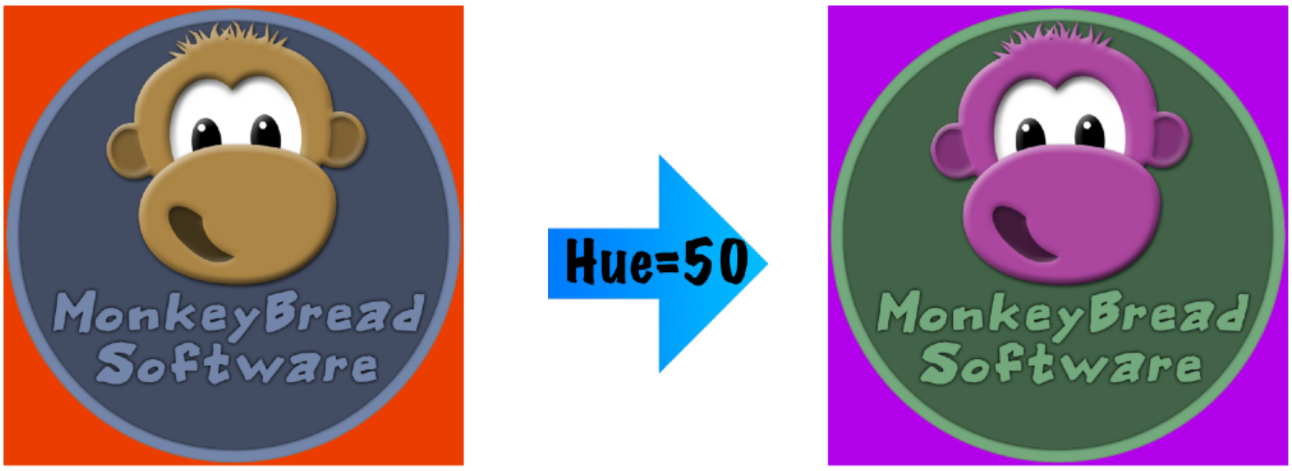
Welcome to the 18th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Do you think that our monkey is a bit pale around the nose? Today we want to change that, because we also have a function that can change the saturation of an image. The GMImage.Modulate function. But not only the saturation can this function influence but also the brightness of the image and the hue. So we can change the HSB values of an image. In the parameters we then specify the corresponding values. If one or more of these three values should not be changed in an image, then we write a 100 in the parameters. The 100 is the neutal value.

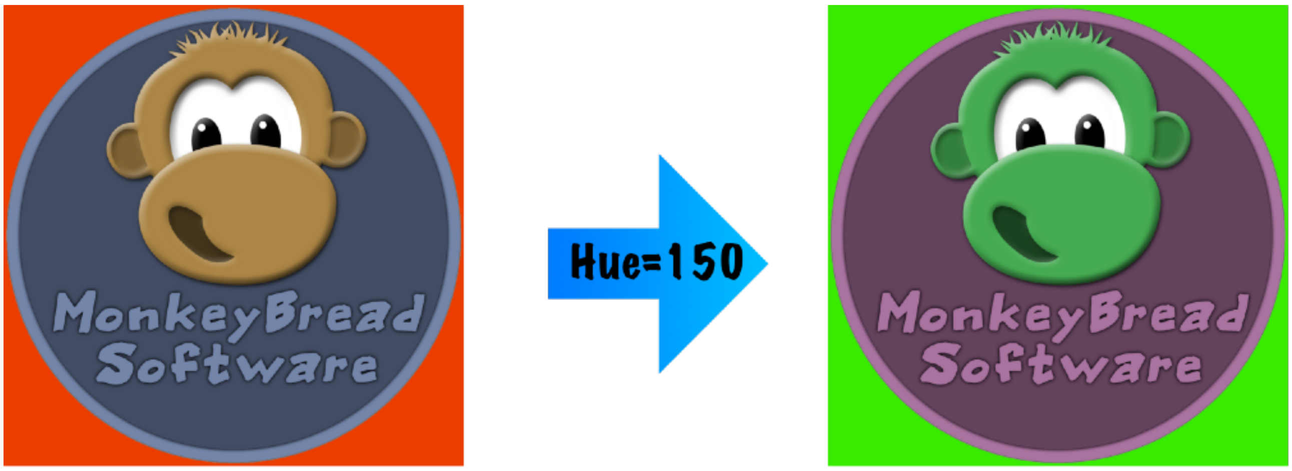
```
Set Variable [ $r ; Value: MBS( "GMImage.Modulate"; $GM;
100; 100; 100) ]
```

Brightness and saturation are relatively easy to understand if we set the brightness to a value greater than 100 then the image becomes brighter and if the value is less than 100 then it becomes darker. Similarly, if the saturation is greater than 100, the saturation will be higher, and if it is less than 100, the saturation will be lower. We can move within the value range from 0 to 200. With the Hue factor it is not so clear what we are actually doing. Here, too, the values move between 0 and 200. We are familiar from many programs that we can specify colors with the help of the color wheel.

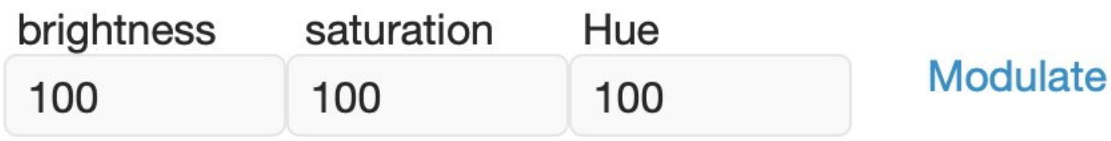
The color can be determined on this color wheel by 3 values. Once the brightness of the circle, this would be determined by the slider under the color circle, the saturation of the color, we determine this by the position of the point inside the circle. The more we come to the outer part of the circle the stronger is the saturation. Our hue is determined by the angle on the color circle. The three values together then describe a color. So if we enter a value of Hue <100, we have to imagine that we rotate all colors on the image in our color circle clockwise. If we enter a number greater than 100 then we rotate counterclockwise by a certain value. We don't work with degrees, but with values between 0 and 200 because 100 is the neutral value. If you want to work with degree numbers you have to convert the value accordingly (degree number/1.8). Let's have a look at the monkey. We want to move 90 degrees clockwise on the color wheel in our image. That means first we calculate 90/1.8 which results in 50. Since we want to move clockwise on the color circle we must now subtract the 50 from the normal value of 100. We get an input of 50, which we can then pass to the function. Before we try this out, let's mentally reproduce this. If we take the red of our background and move it clockwise by 90 degrees on the color circle, we get a purple background. And if we now send the image through the function we see it works.



If we would move our original image 90 degrees counterclockwise on the color wheel (Hue=150) our background would be green. Of course, as you can see, we do this not only with the background, but with every pixel in the image.



So that you can try out the function to your heart's content, three fields have been added to the sample project where you can enter the values for Saturation Brightness and Hue yourself.



brightness	saturation	Hue	
100	100	100	Modulate

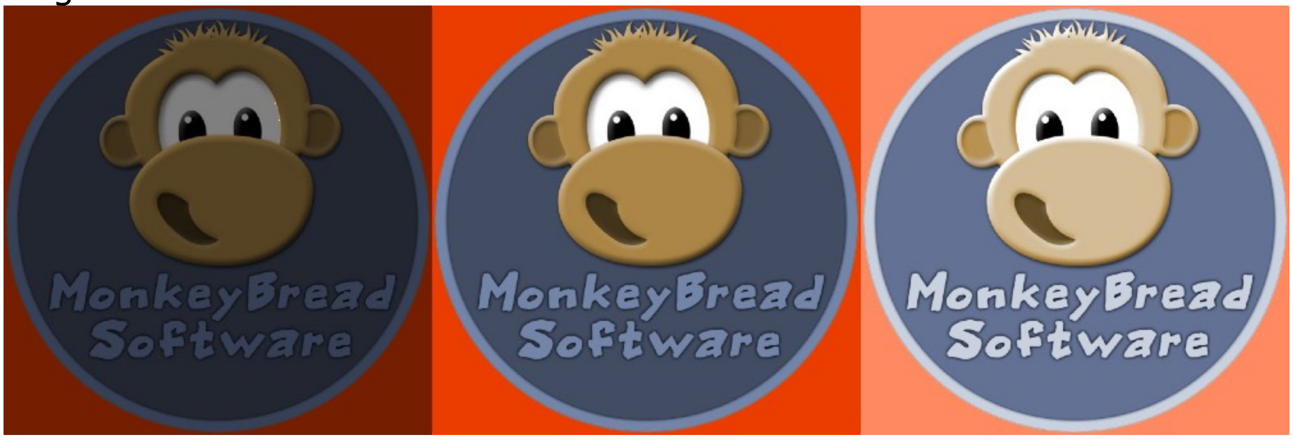
The corresponding script looks like this:

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.Modulate"; $GM;
GraphicsMagick Advent::brightness; GraphicsMagick
Advent::saturation; GraphicsMagick Advent::Hue ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToContainer"; $GM ; "abc.png" ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
Set Field [ GraphicsMagick Advent::Hue ; 100 ]
Set Field [ GraphicsMagick Advent::brightness ; 100 ]
Set Field [ GraphicsMagick Advent::saturation ; 100 ]
```

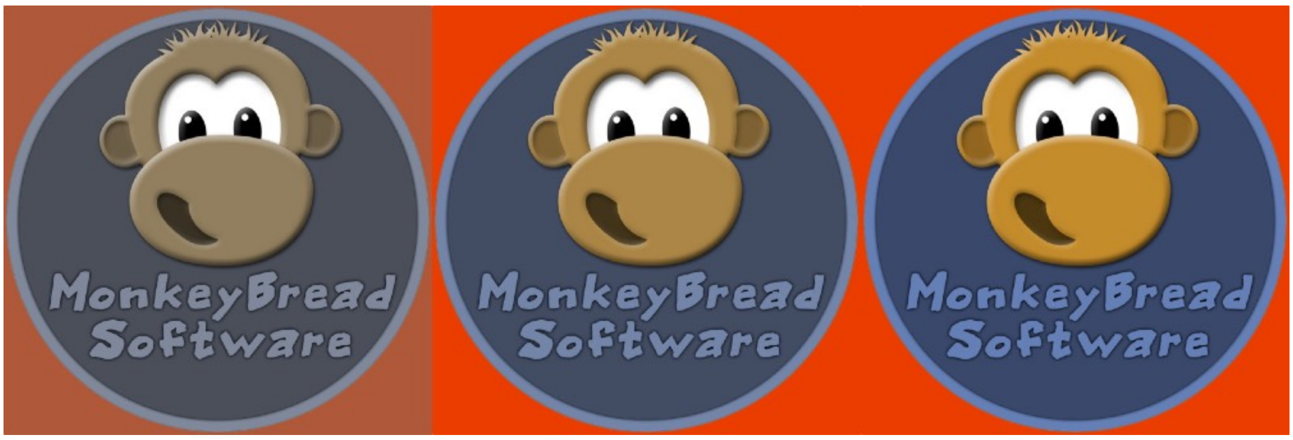
Finally, we see that the field values for the three parameters are set back to the normal value of 100 so that you can reset them if necessary.

Here you can see again how the individual values behave at 50, 100 and 150.

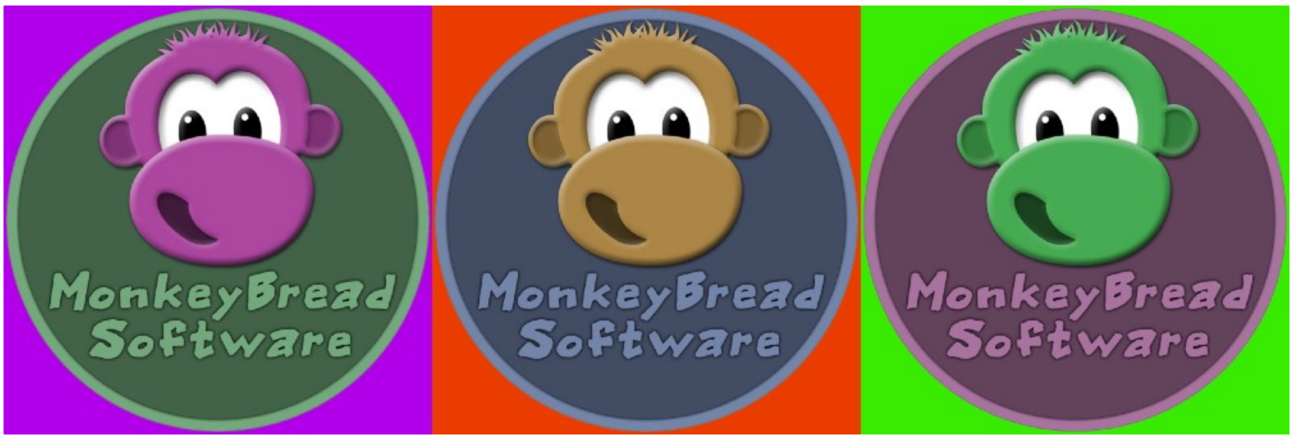
Brightness:



Saturation:



Hue:



Of course, a combination of the values is also possible. Here you can see the difference when all values are set to 50, 100 and 150.

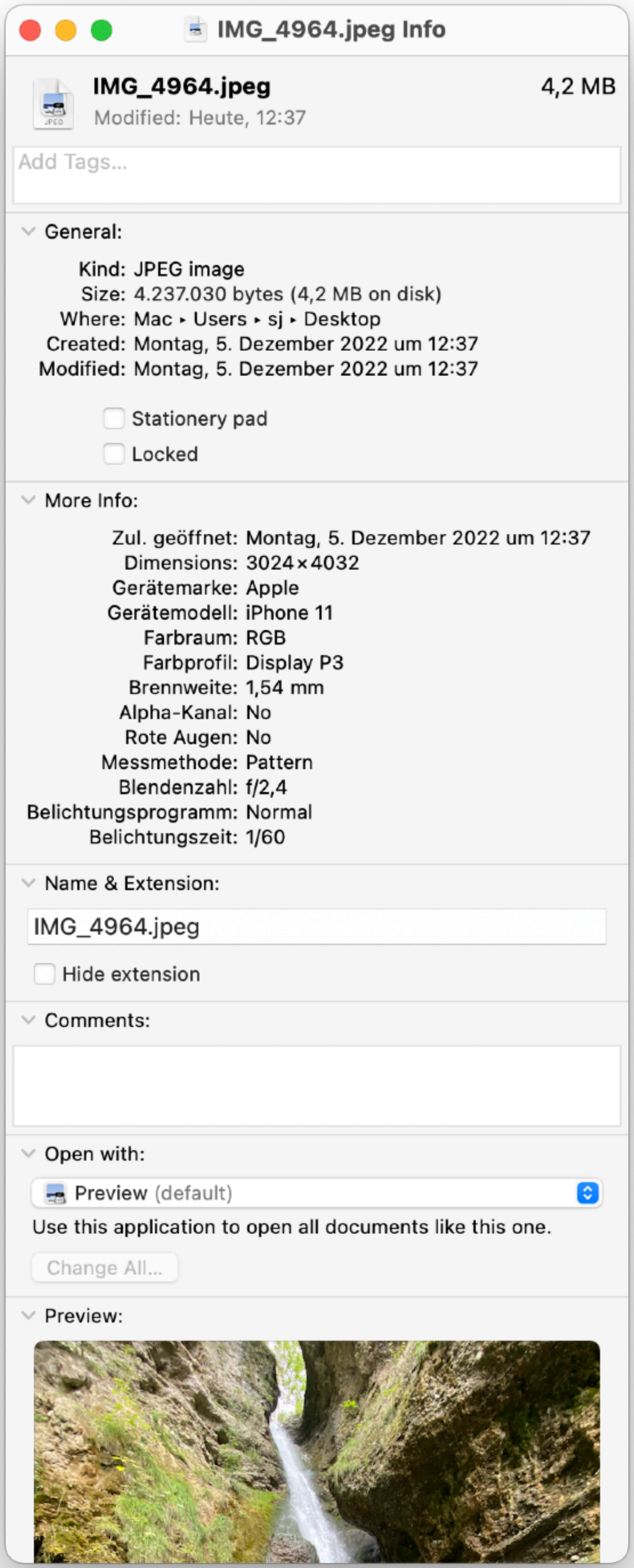
MonkeyBread Software
MonkeyBread Software
MonkeyBread Software

I hope you enjoyed it and I will see you again tomorrow. I wish you a happy fourth advent Sunday.

GraphicsMagick in FileMaker, part 19

Welcome to the 19th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

19 of 24


IMG_4964.jpeg Info

IMG_4964.jpeg 4,2 MB
Modified: Heute, 12:37

Add Tags...

General:

Kind: JPEG image
Size: 4.237.030 bytes (4,2 MB on disk)
Where: Mac ▸ Users ▸ sj ▸ Desktop
Created: Montag, 5. Dezember 2022 um 12:37
Modified: Montag, 5. Dezember 2022 um 12:37

☐ Stationery pad
☐ Locked

More Info:

Zul. geöffnet:	Montag, 5. Dezember 2022 um 12:37
Dimensions:	3024 × 4032
Gerätemarke:	Apple
Gerätemodell:	iPhone 11
Farbraum:	RGB
Farbprofil:	Display P3
Brennweite:	1,54 mm
Alpha-Kanal:	No
Rote Augen:	No
Messmethode:	Pattern
Blendenzahl:	f/2,4
Belichtungsprogramm:	Normal
Belichtungszeit:	1/60

Name & Extension:

IMG_4964.jpeg

☐ Hide extension

Comments:

Open with:

Preview (default)

Use this application to open all documents like this one.

Change All...

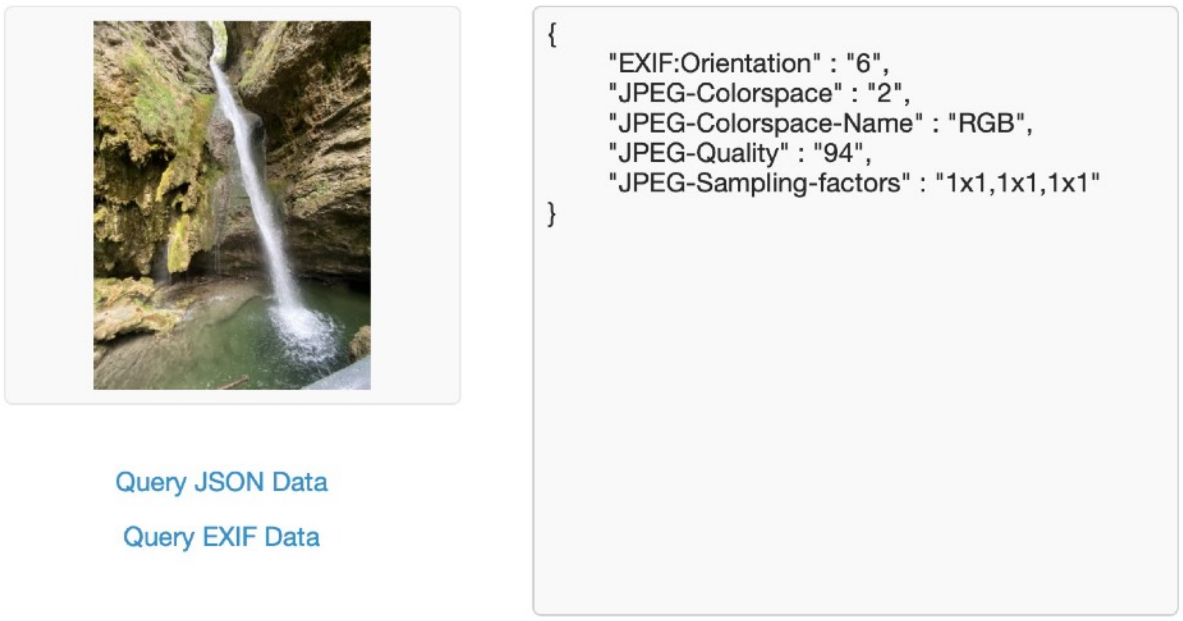
Preview:

We have already seen in door 2 that we can query some information about an image. If we have photos, but also the metadata created by the device when taking the picture are very interesting. Today I want to show you how you can retrieve some of this information.

We use the functions GMImage.GetAttribute for this purpose. But before we start with this function I would like to introduce you to name related functions. On the one hand we have the GMImage.GetAttributesJSON function. It returns the attributes that have been created before as JSON.

```
Set Variable [ $JSON ; Value:
MBS( "GMImage.GetAttributesJSON"; $GM ) ]
```

In the image we can see in the field how this JSON can look like.

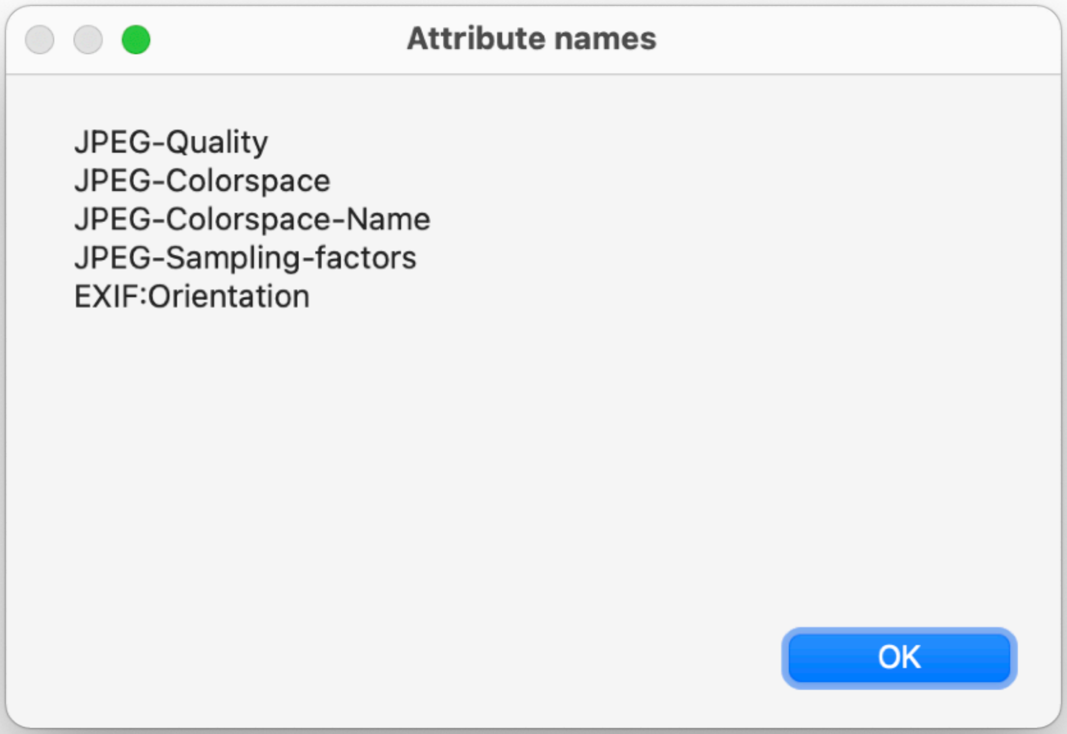


```
{
        "EXIF:Orientation" : "6",
        "JPEG-Colorspace" : "2",
        "JPEG-Colorspace-Name" : "RGB",
        "JPEG-Quality" : "94",
        "JPEG-Sampling-factors" : "1x1,1x1,1x1"
}
```

These values can then be read out with the JSON function of FileMaker or the plugin.

If we want to have the names of the single attributes we can use the GMImage.GetAttributeNames. For example, we can output the attribute names in a dialog. These are returned by the function as a list.

```
Show Custom Dialog [ "Attribute names" ;
MBS( "GMImage.GetAttributeNames";$GM) ]
```

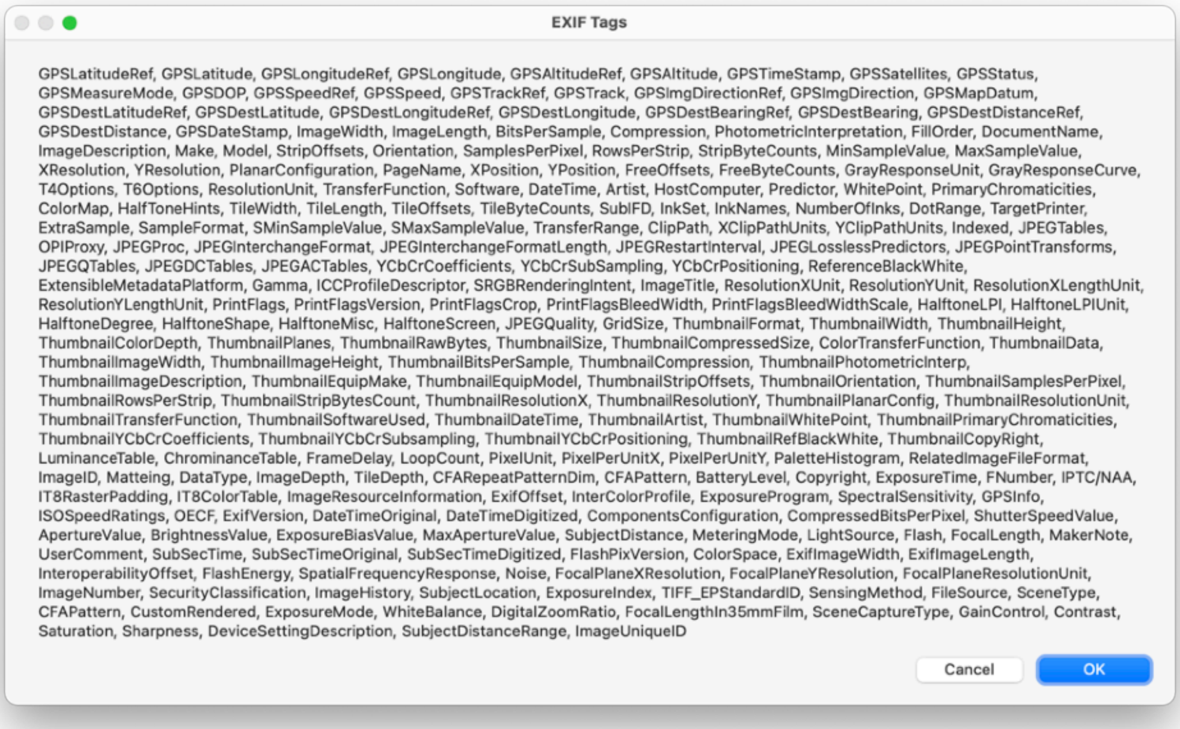


```
JPEG-Quality
JPEG-Colorspace
JPEG-Colorspace-Name
JPEG-Sampling-factors
EXIF:Orientation
```

We see when we open the info of the image on the computer that there is still a lot of information behind it that we don't get through the JSON.

This is because in both functions only attributes created before are visible, so e.g. EXIF data may not yet be loaded and parsed. But with the GMImage.GetAttribute function we can retrieve EXIF data by specifically querying the individual attributes. In the function we first specify our reference and then the name of the attribute. Optionally we can specify the encoding of the text. As return we get our value to the matching attribute. If we want to retrieve a value from the EXIF data, it is not enough to specify the tag name, but we have to specify the tag further, so that the attribute name has the following form:**EXIF:*Attribute tag***

In our example we have stored a list of some possible EXIF attribute names.



GPSLatitudeRef, GPSLatitude, GPSLongitudeRef, GPSLongitude, GPSAltitudeRef, GPSAltitude, GPSTimeStamp, GPSSatellites, GPSStatus, GPSMeasureMode, GPSDOP, GPSSpeedRef, GPSSpeed, GPSTrackRef, GPSTrack, GPSImgDirectionRef, GPSImgDirection, GPSMapDatum, GPSDestLatitudeRef, GPSDestLatitude, GPSDestLongitudeRef, GPSDestLongitude, GPSDestBearingRef, GPSDestBearing, GPSDestDistanceRef, GPSDestDistance, GPSDateStamp, ImageWidth, ImageLength, BitsPerSample, Compression, PhotometricInterpretation, FillOrder, DocumentName, ImageDescription, Make, Model, StripOffsets, Orientation, SamplesPerPixel, RowsPerStrip, StripByteCounts, MinSampleValue, MaxSampleValue, XResolution, YResolution, PlanarConfiguration, PageName, XPosition, YPosition, FreeOffsets, FreeByteCounts, GrayResponseUnit, GrayResponseCurve, T4Options, T6Options, ResolutionUnit, TransferFunction, Software, DateTime, Artist, HostComputer, Predictor, WhitePoint, PrimaryChromaticities, ColorMap, HalfToneHints, TileWidth, TileLength, TileOffsets, TileByteCounts, SubIFD, InkSet, InkNames, NumberOfInks, DotRange, TargetPrinter, ExtraSample, SampleFormat, SMinSampleValue, SMaxSampleValue, TransferRange, ClipPath, XClipPathUnits, YClipPathUnits, Indexed, JPEGTables, OPIProxy, JPEGProc, JPEGInterchangeFormat, JPEGInterchangeFormatLength, JPEGRestartInterval, JPEGLosslessPredictors, JPEGPointTransforms, JPEGQTables, JPEGDCTables, JPEGACTables, YCbCrCoefficients, YCbCrSubSampling, YCbCrPositioning, ReferenceBlackWhite, ExtensibleMetadataPlatform, Gamma, ICCProfileDescriptor, SRGBRenderingIntent, ImageTitle, ResolutionXUnit, ResolutionYUnit, ResolutionXLengthUnit, ResolutionYLengthUnit, PrintFlags, PrintFlagsVersion, PrintFlagsCrop, PrintFlagsBleedWidth, PrintFlagsBleedWidthScale, HalftoneLPI, HalftoneLPIUnit, HalftoneDegree, HalftoneShape, HalftoneMisc, HalftoneScreen, JPEGQuality, GridSize, ThumbnailFormat, ThumbnailWidth, ThumbnailHeight, ThumbnailColorDepth, ThumbnailPlanes, ThumbnailRawBytes, ThumbnailSize, ThumbnailCompressedSize, ColorTransferFunction, ThumbnailData, ThumbnailImageWidth, ThumbnailImageHeight, ThumbnailBitsPerSample, ThumbnailCompression, ThumbnailPhotometricInterp, ThumbnailImageDescription, ThumbnailEquipMake, ThumbnailEquipModel, ThumbnailStripOffsets, ThumbnailOrientation, ThumbnailSamplesPerPixel, ThumbnailRowsPerStrip, ThumbnailStripBytesCount, ThumbnailResolutionX, ThumbnailResolutionY, ThumbnailPlanarConfig, ThumbnailResolutionUnit, ThumbnailTransferFunction, ThumbnailSoftwareUsed, ThumbnailDateTime, ThumbnailArtist, ThumbnailWhitePoint, ThumbnailPrimaryChromaticities, ThumbnailYCbCrCoefficients, ThumbnailYCbCrSubsampling, ThumbnailYCbCrPositioning, ThumbnailRefBlackWhite, ThumbnailCopyRight, LuminanceTable, ChrominanceTable, FrameDelay, LoopCount, PixelUnit, PixelPerUnitX, PixelPerUnitY, PaletteHistogram, RelatedImageFileFormat, ImageID, Matteing, DataType, ImageDepth, TileDepth, CFARepeatPatternDim, CFAPattern, BatteryLevel, Copyright, ExposureTime, FNumber, IPTC/NAA, IT8RasterPadding, IT8ColorTable, ImageResourceInformation, ExifOffset, InterColorProfile, ExposureProgram, SpectralSensitivity, GPSInfo, ISOSpeedRatings, OECF, ExifVersion, DateTimeOriginal, DateTimeDigitized, ComponentsConfiguration, CompressedBitsPerPixel, ShutterSpeedValue, ApertureValue, BrightnessValue, ExposureBiasValue, MaxApertureValue, SubjectDistance, MeteringMode, LightSource, Flash, FocalLength, MakerNote, UserComment, SubSecTime, SubSecTimeOriginal, SubSecTimeDigitized, FlashPixVersion, ColorSpace, ExifImageWidth, ExifImageLength, InteroperabilityOffset, FlashEnergy, SpatialFrequencyResponse, Noise, FocalPlaneXResolution, FocalPlaneYResolution, FocalPlaneResolutionUnit, ImageNumber, SecurityClassification, ImageHistory, SubjectLocation, ExposureIndex, TIFF_EPStandardID, SensingMethod, FileSource, SceneType, CFAPattern, CustomRendered, ExposureMode, WhiteBalance, DigitalZoomRatio, FocalLengthIn35mmFilm, SceneCaptureType, GainControl, Contrast, Saturation, Sharpness, DeviceSettingDescription, SubjectDistanceRange, ImageUniqueID

In our example we go through this list and append the tag and the corresponding value to the result string which we then output in a field. Here is the code for this:

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer"; 
GraphicsMagick Advent::Image) ]
Set Variable [ $tags ; Value: 
"GPSLatitudeRef¶GPSLatitude¶GPSLongitudeRef¶GPSLongitude¶
GPSAltitudeRef¶GPSAltitude¶GPSTimeStamp¶GPSSatellites¶GPSSta
tus¶GPSMeasureMode¶GPSDOP¶
GPSSpeedRef¶GPSSpeed¶GPSTrackRef¶GPSTrack¶GPSImgDirectionRef
¶GPSImgDirection¶
GPSMapDatum¶GPSDestLatitudeRef¶GPS..." ]

Set Variable [ $count ; Value: ValueCount ( $tags ) ]
Set Variable [ $index ; Value: 1 ]
Set Variable [ $result ; Value: "" ]
Loop
     # get the property name from the list
     Set Variable [ $tag ; Value: GetValue ( $tags; 
$index ) ]
     # Get the value of the property
     Set Variable [ $value ; Value: 
MBS("GMImage.GetAttribute"; $GM; "EXIF:" & $tag) ]
     If [ $value  ≠ "unknown" ]
          # add the property and the value to the result
          Set Variable [ $result ; Value: $result & $tag & ": 
" & $value  & "¶" ]
```

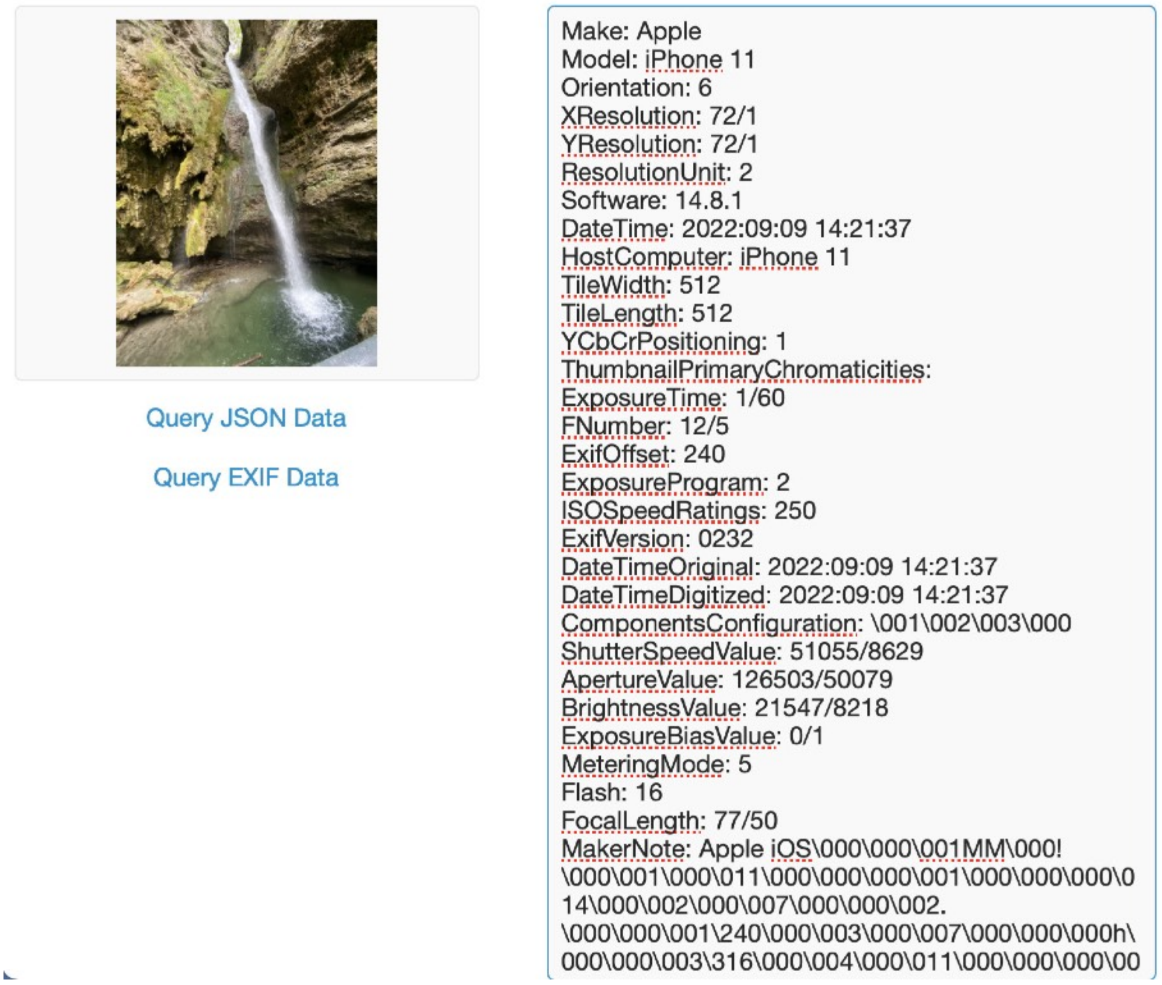
```
    End If
    Set Variable [ $index ; Value: $index +1 ]
    Exit Loop If [ $index > $count ]
End Loop
Set Field [ GraphicsMagick Advent::Result ; $result ]
Set Variable [ $r ; Value: MBS("GMImage.Free"; $GM) ]
```

We then first look how many entries we have in this list. Set our index to 1 because we start with a FileMaker list with an index of 1. In the loop we get each Tag from the list, put the string together appropriately in the function call and then get the appropriate value. We can see that in this line:

```
Set Variable [ $value ; Value: MBS("GMImage.GetAttribute";
$GM; "EXIF:" & $tag) ]
```

If we have a value that is not *unknown*, then we append this information to our result.

When the loop is finished, we write the result text in the field and release our reference.



```
Make: Apple
Model: iPhone 11
Orientation: 6
XResolution: 72/1
YResolution: 72/1
ResolutionUnit: 2
Software: 14.8.1
DateTime: 2022:09:09 14:21:37
HostComputer: iPhone 11
TileWidth: 512
TileLength: 512
YCbCrPositioning: 1
ThumbnailPrimaryChromaticities:
ExposureTime: 1/60
FNumber: 12/5
ExifOffset: 240
ExposureProgram: 2
ISOSpeedRatings: 250
ExifVersion: 0232
DateTimeOriginal: 2022:09:09 14:21:37
DateTimeDigitized: 2022:09:09 14:21:37
ComponentsConfiguration: \001\002\003\000
ShutterSpeedValue: 51055/8629
ApertureValue: 126503/50079
BrightnessValue: 21547/8218
ExposureBiasValue: 0/1
MeteringMode: 5
Flash: 16
FocalLength: 77/50
MakerNote: Apple iOS\000\000\001MM\000!
\000\001\000\011\000\000\000\001\000\000\000\0
14\000\002\000\007\000\000\002.
\000\000\001\240\000\003\000\007\000\000\000h\
000\000\003\316\000\004\000\011\000\000\000\00
```

I hope you enjoyed this door and I will see you again tomorrow.

GraphicsMagick in FileMaker, part 20

20 of 24

Welcome to the 20th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I want to show you that with GraphicsMagick you can not only edit images, but we can also draw by ourselves. Today I will introduce you to the basic shapes.

To have a working environment on which we can work, we first use the GMImage.New function. This creates a working environment for us. We determine the size and the background color in the parameters. Here we draw a working environment of 500x500 pixels and a white background color.

```
Set Variable [ $GM ; Value: MBS( "GMImage.New"; "500x500";
"#FFFFFF" ) ]
```

We have several basic shapes available in the plugin:

- Line
- Circle
- Ellipse
- Rectangle
- Rounded rectangle
- Arc

I will show you these shapes one after the other.

So that you can bring color directly into action in our example I would like to show you again briefly the functions for the color which you have already learned in the text annotation on pictures.

We have the line color that we can set. We specify the reference and the color, here the line color is red.

```
Set Variable [ $r ; Value: MBS("GMImage.SetStrokeColor";
$GM; "#FF0000") ]
```

Then we have the fill color, which is the color with which our figures will be filled. Here a blue.

```
Set Variable [ $r ; Value: MBS("GMImage.SetFillColor"; $GM;
"#0000FF") ]
```

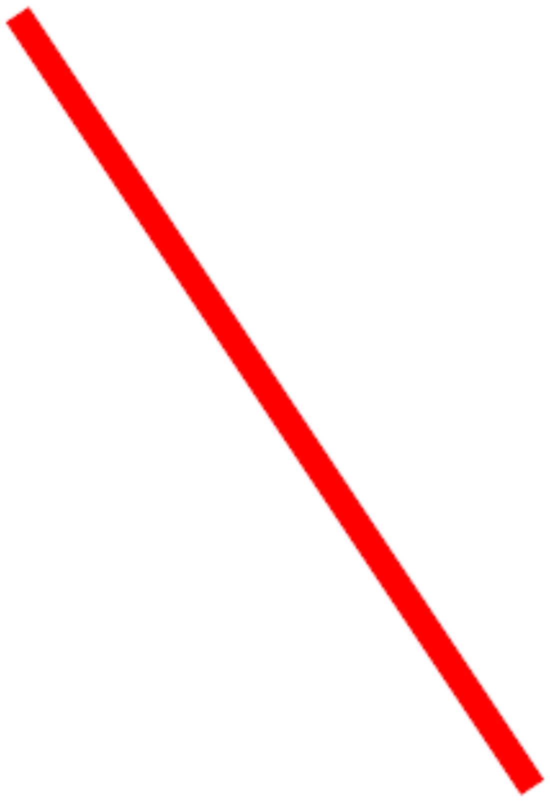
And we have the line thickness that we can set.

```
Set Variable [ $r ; Value: MBS("GMImage.SetLineWidth"; $GM;
10) ]
```

Let's start with the line. For this we have the function GMImage.DrawLine. In the parameters we define the two points with their coordinates that should be connected. The origin of the coordinate system is in the upper left corner. This means that the higher the x value, the more we move to the right. For the y value, the larger it gets, the more we move down. You can also use the function GMImage.TransformOrigin to move the coordinate origin, but I don't want to do that now. In the script step you see here we connect the points (50/50) and (250/350) with each other.

```
Set Variable [ $r ; Value: MBS("GMImage.DrawLine"; $GM; 50;
50; 250; 350) ]
```

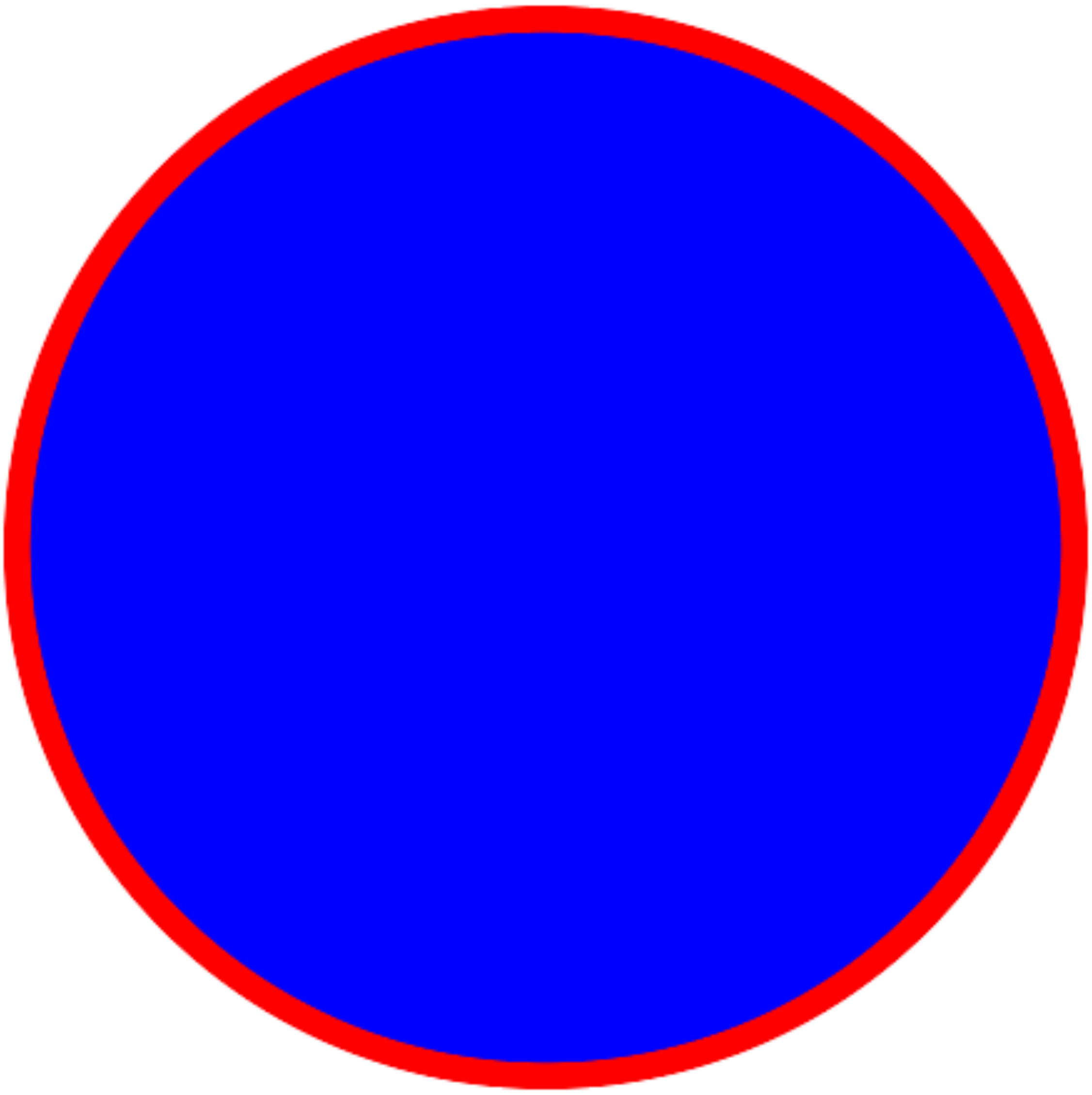
Our result is a line that slopes from top left to bottom right.



Next we come to the circle. For this we use the GMImage.DrawCircle function. We first set the coordinates for the circle center. In our example we want to have it in the center of our workspace (250/250). Also, we set a point that is on the outer surface of the circle. I would like to have a circle of 150 pixels radius (300 px diameter). So the point 100/100 is on my circle outer circle. I also specify this in the parameters.

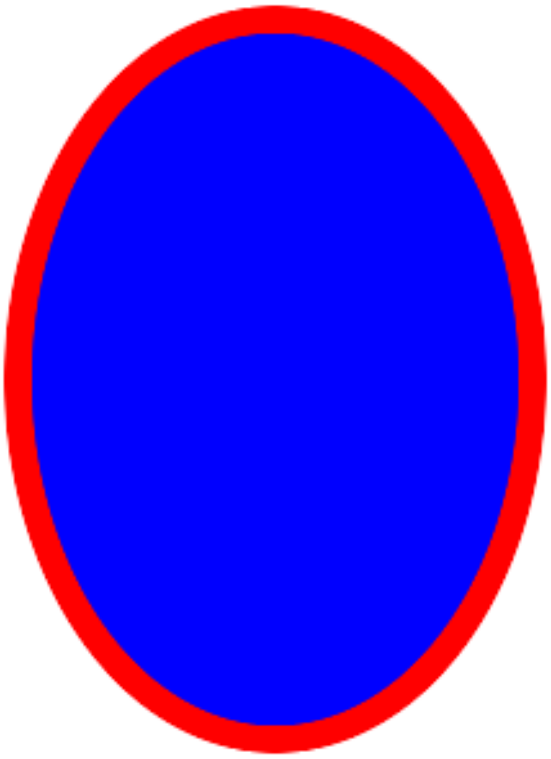
```
Set Variable [ $r ; Value: MBS( "GMImage.DrawCircle";$GM;
250; 250; 100; 100 ) ]
```

Our circle look like that

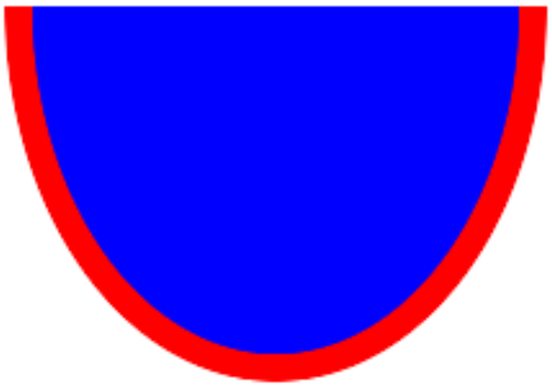


We define the ellipse with the GMImage.DrawEllipse function. Again, we first define the center of the ellipse. With the next value we determine the radius to the right and left to the ellipse edge (100) and with the following parameter (140) the radius of the ellipse to the top and bottom. With the last two parameters we can specify how much of the ellipse should be drawn. So we specify the segment of the circle. If we have a complete ellipse, we specify 0 as the start angle and 360 as the target angle.

```
Set Variable [ $r ; Value: MBS("GMImage.DrawEllipse"; $GM;
250; 250; 100; 140; 0; 360) ]
```

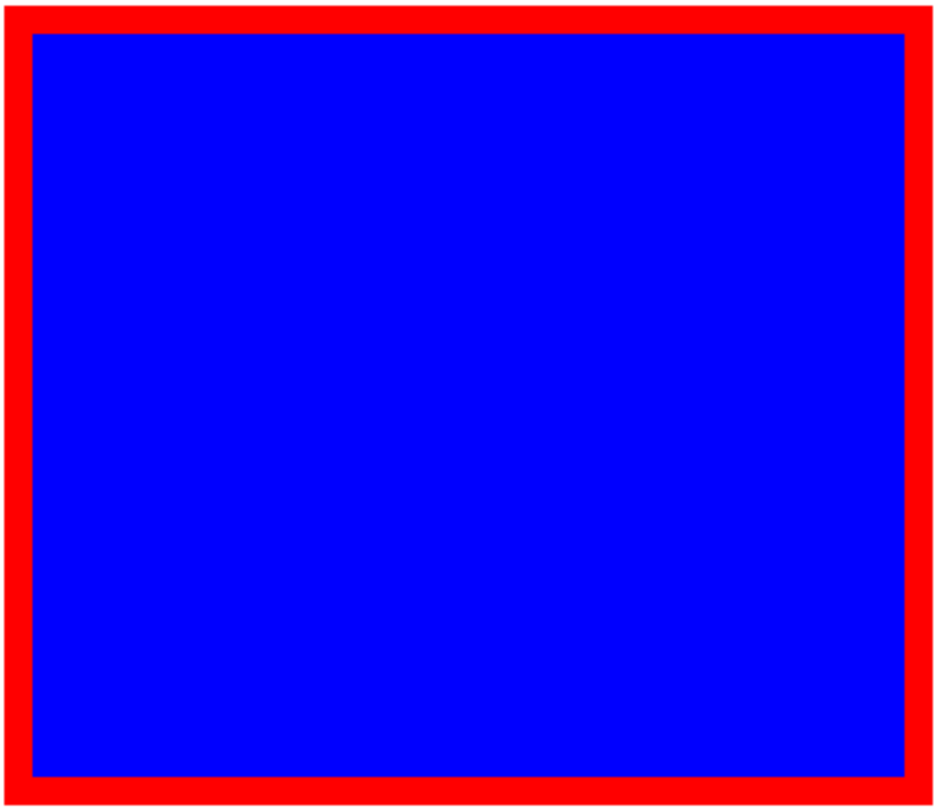


If we give the ellipse a start angle of 0 and a target angle of 180 then our ellipse looks like this



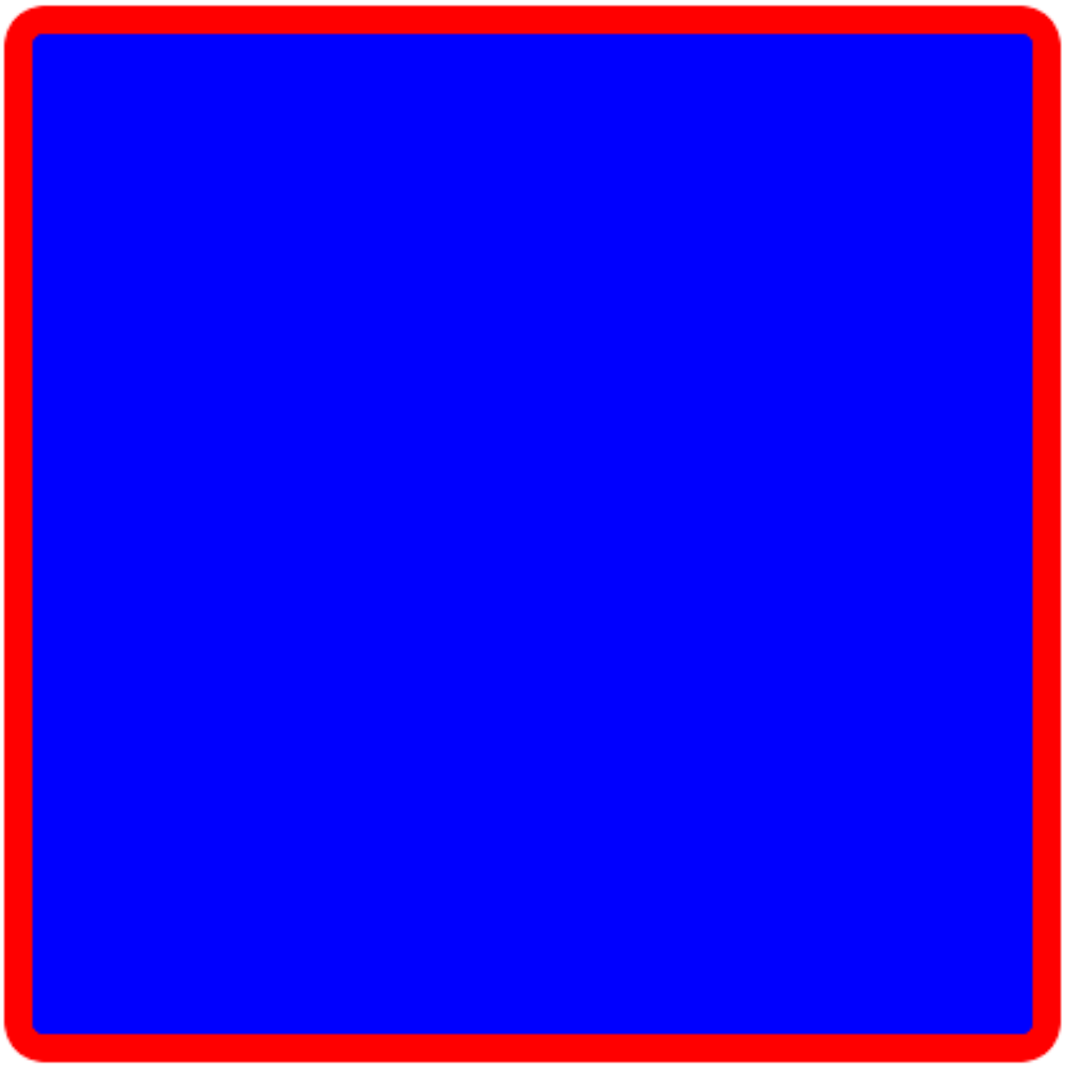
Then we have our rectangle and our rounded rectangle. The functions for these shapes are similar. Let's start with the simpler function for the rectangle. For this we use the GMImage.DrawRectangle function. We define the rectangle by first specifying the upper left corner of the rectangle (50/50) and then the lower right corner (400/350).

```
Set Variable [ $r ; Value: MBS( "GMImage.DrawRectangle";
$GM; 50; 50; 400; 350 ) ]
```



The difference between a rectangle and a rounded rectangle are the four rounded corners. We can create such a rectangle with the GMImage.DrawRoundRectangle function. The first parameters are the same as the rectangle (upper left corner and lower left corner) and the last two parameters specify the radius of the corners.

```
Set Variable [ $r ; Value: MBS("GMImage.DrawRoundRectangle";
$GM; 50; 50; 450; 450; 10; 10) ]
```



The arc is a bit more complicated. An arc is a section of an ellipse and we use the GMImage.DrawArc function to draw it. Here we first define a box in which an ellipse would fit exactly. The last two parameters describe what we see of the ellipse. To illustrate this, I have defined 4 individual arcs, each of which has been assigned a different color.

```
Set Variable [ $GM ; Value: MBS( "GMImage.New"; "500x500";
"#FFFFFF" ) ]

Set Variable [ $r ; Value: MBS("GMImage.SetStrokeColor";
$GM; "#FF0000") ]
Set Variable [ $r ; Value: MBS("GMImage.SetFillColor"; $GM;
"#0000FF") ]
Set Variable [ $r ; Value: MBS("GMImage.SetLineWidth"; $GM;
10) ]
```

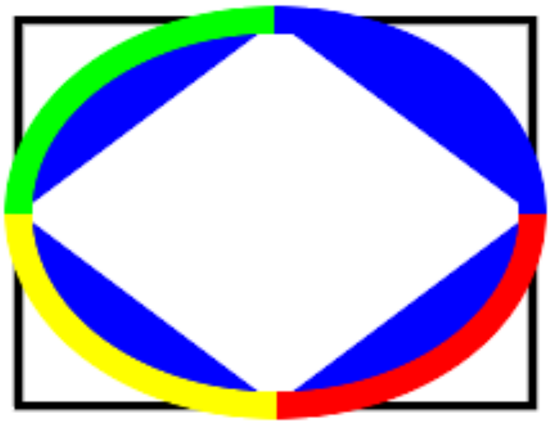
```
Set Variable [ $r ; Value: MBS("GMImage.DrawArc"; $GM; 100;
100; 300; 250; 0; 90) ]

Set Variable [ $r ; Value: MBS("GMImage.SetStrokeColor";
$GM; "#FFFF00") ]
Set Variable [ $r ; Value: MBS("GMImage.DrawArc"; $GM; 100;
100; 300; 250; 90; 180) ]

Set Variable [ $r ; Value: MBS("GMImage.SetStrokeColor";
$GM; "#00FF00") ]
Set Variable [ $r ; Value: MBS("GMImage.DrawArc"; $GM; 100;
100; 300; 250; 180; 270) ]

Set Variable [ $r ; Value: MBS("GMImage.SetStrokeColor";
$GM; "#0000FF") ]
Set Variable [ $r ; Value: MBS("GMImage.DrawArc"; $GM; 100;
100; 300; 250; 270; 360) ]
```

To clarify the box that defines the size of the ellipse, it is drawn in black.



To get our image displayed correctly we use the function GMImage.WriteToPNGContainer to save the graphic environment as PNG image.

```
Set Field [ GraphicsMagick Advent::Image ;
MBS("GMImage.WriteToPNGContainer"; $GM; "abc.png") ]
```

I hope you enjoyed it again this time and will join us tomorrow when we open the next door.

GraphicsMagick in FileMaker, part 21

21 of 24

Welcome to the 21st door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Yesterday we already looked at how we can draw with the basic shapes in GraphicsMagick. Today we would like to draw our own shapes. For this we have some functions in the plugin that we can use. We will draw our own shapes with paths. Let's imagine a pen. We tell this pen where it is and where it goes. Let's draw a triangle together. To do this, we first need to move our pen to the start position. For this we use the function GMImage.AddPathMovetoAbs in the parameters we specify the start coordinates for our point (50/50) in addition to the reference.

```
Set Variable [ $r ; Value: MBS( "GMImage.AddPathMovetoAbs";
$GM; 50; 50 ) ]
```

We have now specified an absolute position because we want to start exactly at these coordinates. But if you have a look at the documentation of GraphicsMagick you will notice that there is also a very similar function called GMImage.AddPathMovetoRel. The difference between these two functions is that we specify the absolute coordinates (the exact point where we want to place the pen) or the relative coordinates. The relative coordinates describe where the point is in relation to the old position. This means for example if we have a Relative Coordinate (1/1) we go one unit to the right and one unit down from the point where we are currently located. Most path drawing functions from the MBS FileMaker Plugin occur twice with this difference.

We want to draw a line from the point where we are now (50/50) straight down and this line should be 300 pixels long. For this we use the GMImage.AddPathLinetoRel function which expects a relative coordinate from us. So we specify that our coordinate is located 0 pixels to the right from our position and 300 pixels below our position.

```
Set Variable [ $r ; Value: MBS( "GMImage.AddPathLinetoRel";
$GM; 0; 300 ) ]
```

Our third triangle coordinate should now be at 400/350. So this time we use the function GMImage.AddPathLinetoAbs and pass it the Absolute Coordinate.

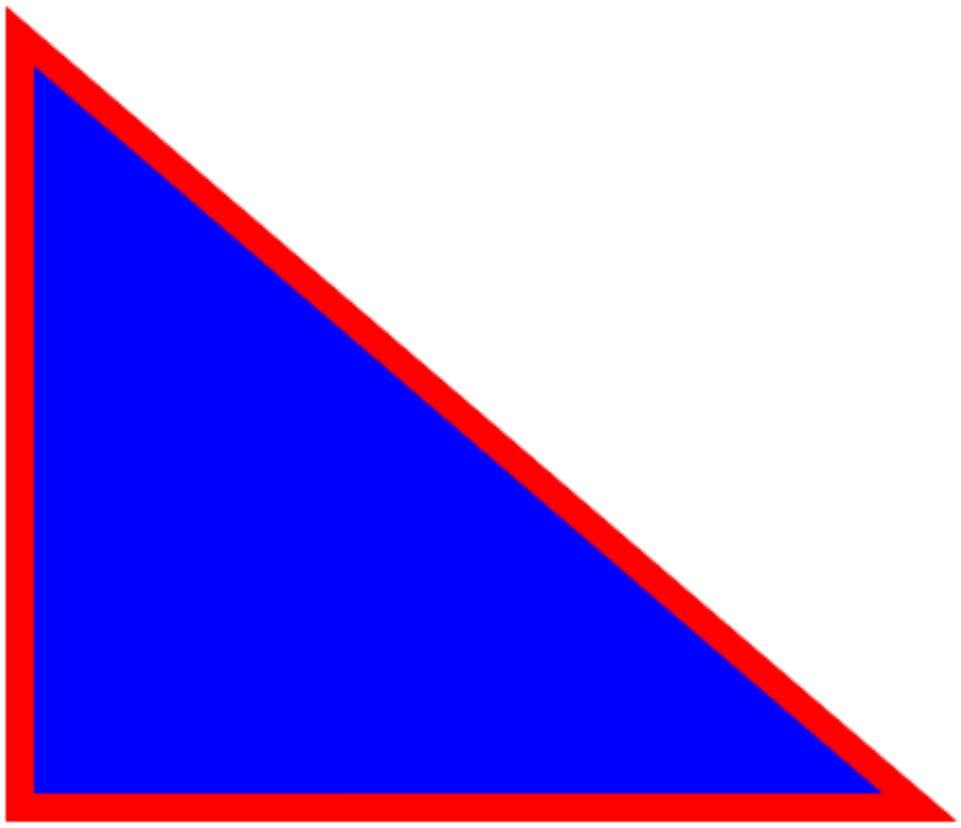
```
Set Variable [ $r ; Value: MBS( "GMImage.AddPathLinetoAbs";
$GM; 400; 350 ) ]
```

We have described our three corner coordinates and connected them with two lines. Now we only have to connect the last corner point with the starting point again. For this we can use the already known functions, or we can use the function GMImage.AddPathClosePath which connects the point where our pen is currently located with the starting point.

```
Set Variable [ $r ; Value: MBS( "GMImage.AddPathClosePath";
$GM ) ]
```

If we exported our graphics environment into the container like this, we would see that we don't see anything, because we have already prepared the path, but this path is first drawn with the function GMImage.DrawPath.

```
Set Variable [ $r ; Value: MBS( "GMImage.DrawPath"; $GM )
```

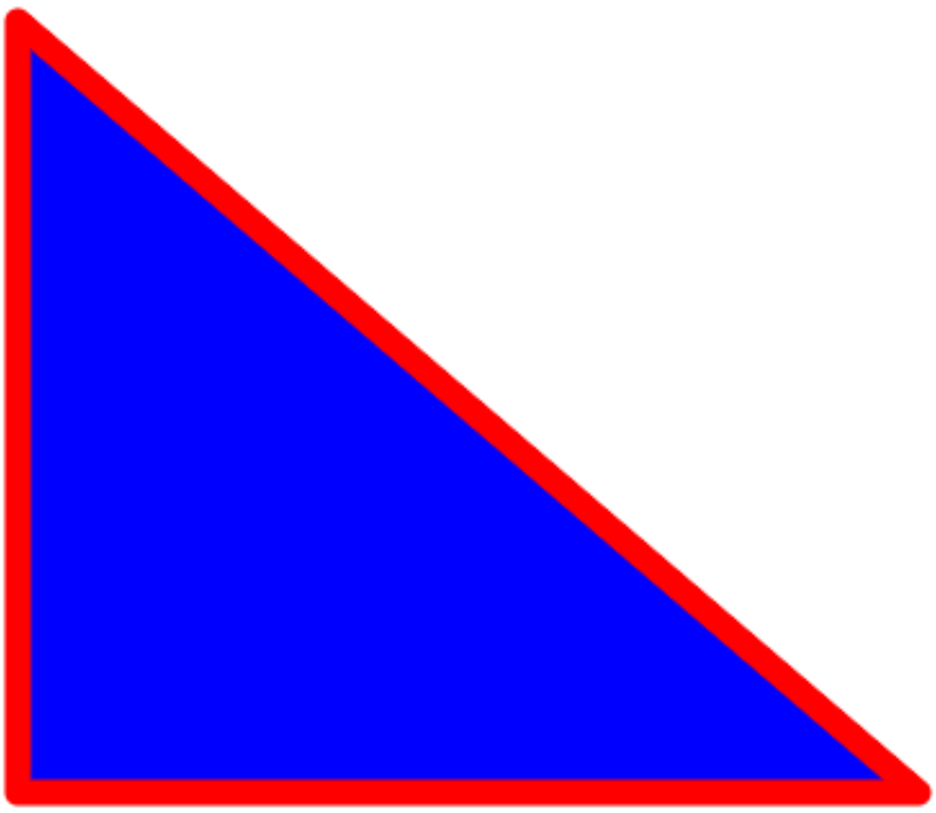


Our corners are now very sharp if we prefer to round these corners we can use the GMImage.SetStrokeLineJoin function. Here we have a selection from

- UndefinedJoin = 0
- MiterJoin = 1
- RoundJoin = 2
- BevelJoin = 3

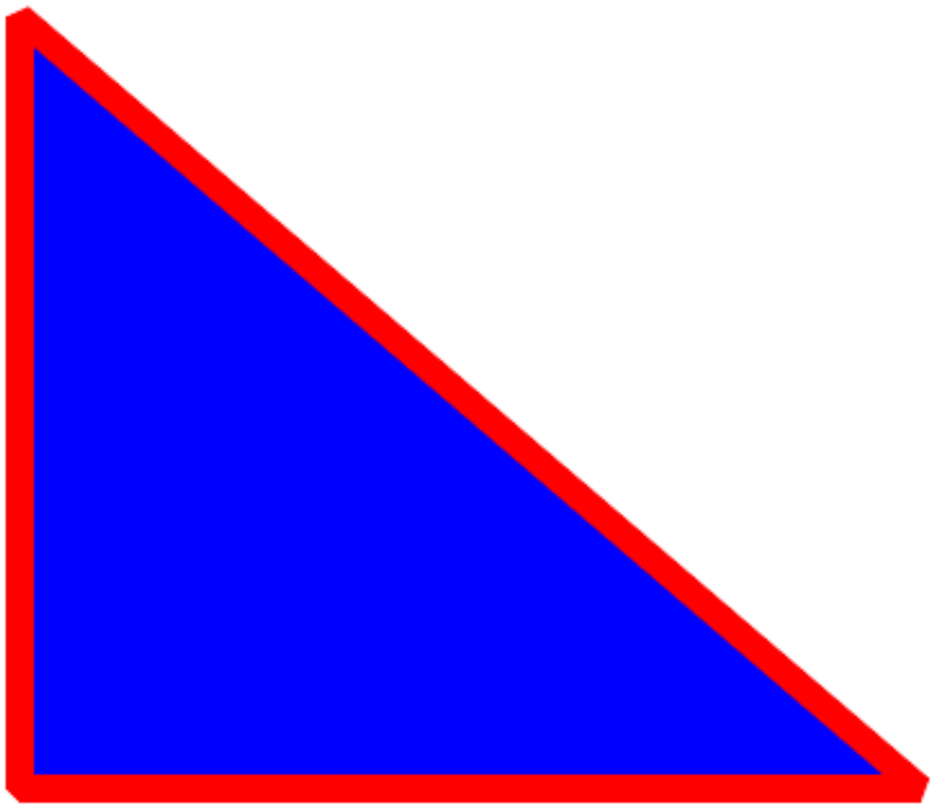
Miter Join is where the lines meet in the middle and form sharp edges as we have just seen.

Round Join rounds off the edges



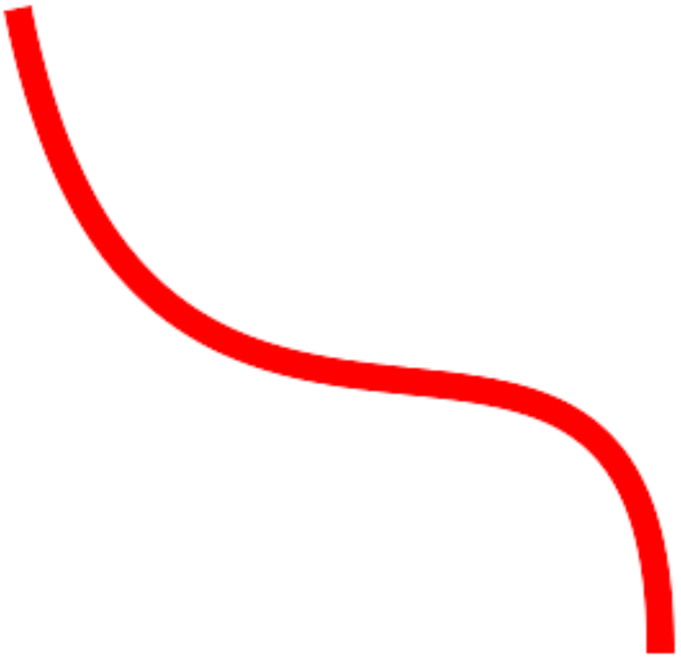
The bevel join creates an additional beveled edge

```
Set Variable [ $r ; Value: MBS( "GMImage.SetStrokeLineJoin";
$GM; 2 ) ]
```



We can draw not only straight lines but also curves. We want to draw a cubic Bezier curve and for this we use the function GMImage.AddPathCurvetoAbs. We give this function three points. The first control point (100/300), the second control point (300/100) and the end point (300/300). We don't need to specify the start point, because we already have it by the position of our pen.

```
Set Variable [ $r ; Value: MBS("GMImage.AddPathCurvetoAbs"; $GM; 100; 300; 300; 100; 300; 300 ) ]
```



I hope you enjoyed this door and see you tomorrow again for the 22th door of our Advent calendar until then I hope you have fun with your drawings.

GraphicsMagick in FileMaker, part 22

22 of 24

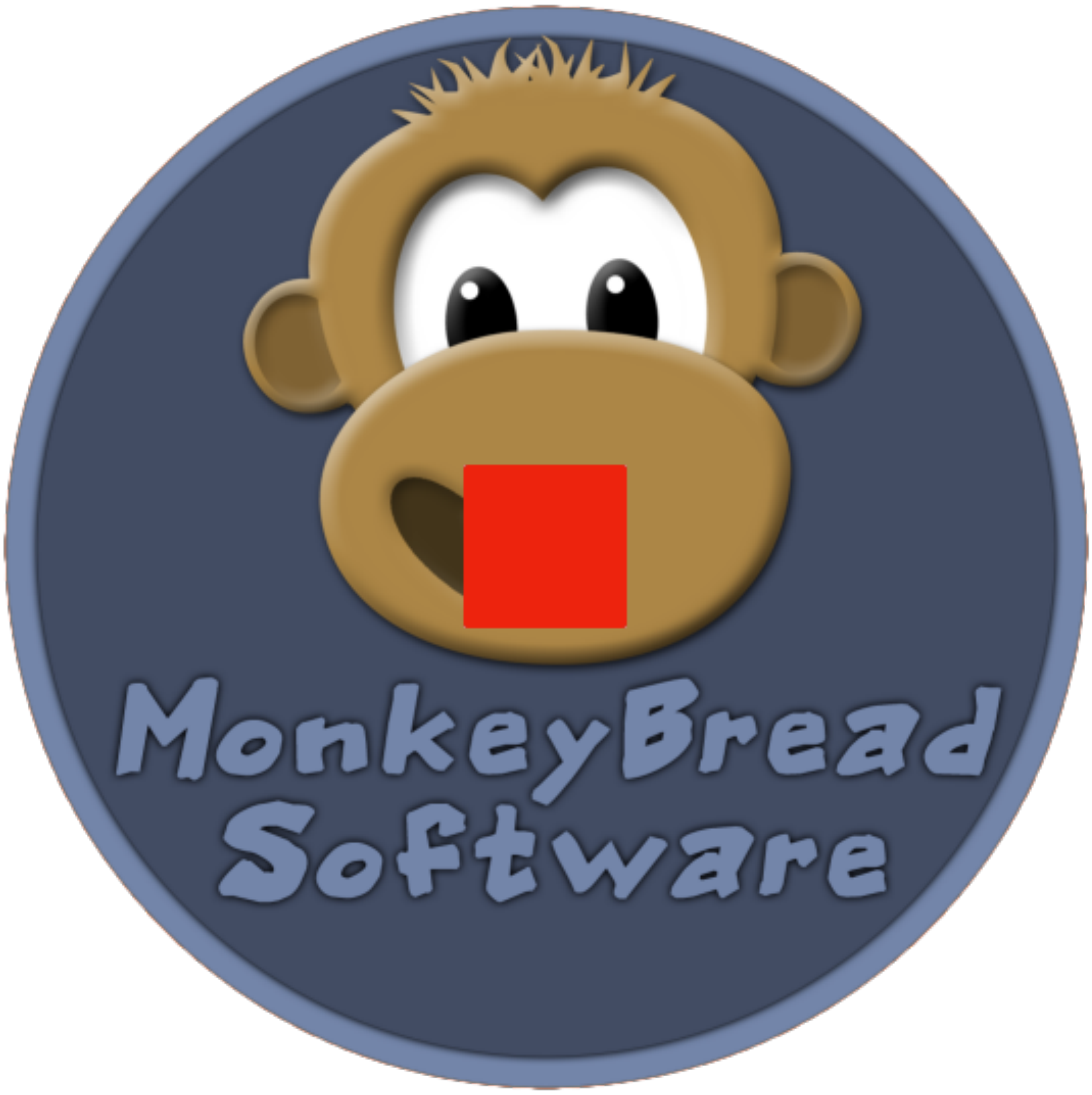
Welcome to the 22th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I show you how you can combine pictures. It's almost Christmas and our monkey is still missing the right costume. This we want to put on him today. The Christmas hat is on a separate PNG image. That means we want to put the image with the Christmas hat on top of the image with our logo. For this we can choose from two functions GMImage.Composite and GMImage.CompositeXY. I would like to introduce the GMImage.Composite function first. First we specify the reference in which we want to have the result in our case this is the reference of the logo. Next is the reference of the image or workspace we want to combine with the image, in our case an image of the cap. The next parameter refers to how the second image is arranged to the first one. Our two images are the same size, so it makes no difference for us the way the two images are positioned in relation to each other, but if the images are different in size, for example the second image is smaller than the first, then it makes a difference whether we align the image to the upper left corner or centered. The following options are available to us:

- ForgetGravity=0
- NorthWestGravity=1
- NorthGravity=2
- NorthEastGravity=3
- WestGravity=4

- CenterGravity=5
- EastGravity=6
- SouthWestGravity=7
- SouthGravity=8
- SouthEastGravity=9
- StaticGravity=10

Here we see on the small box we put over the logo how the Gravity behaves:
CenterGravity=5



NorthWestGravity=1

MonkeyBread
Software

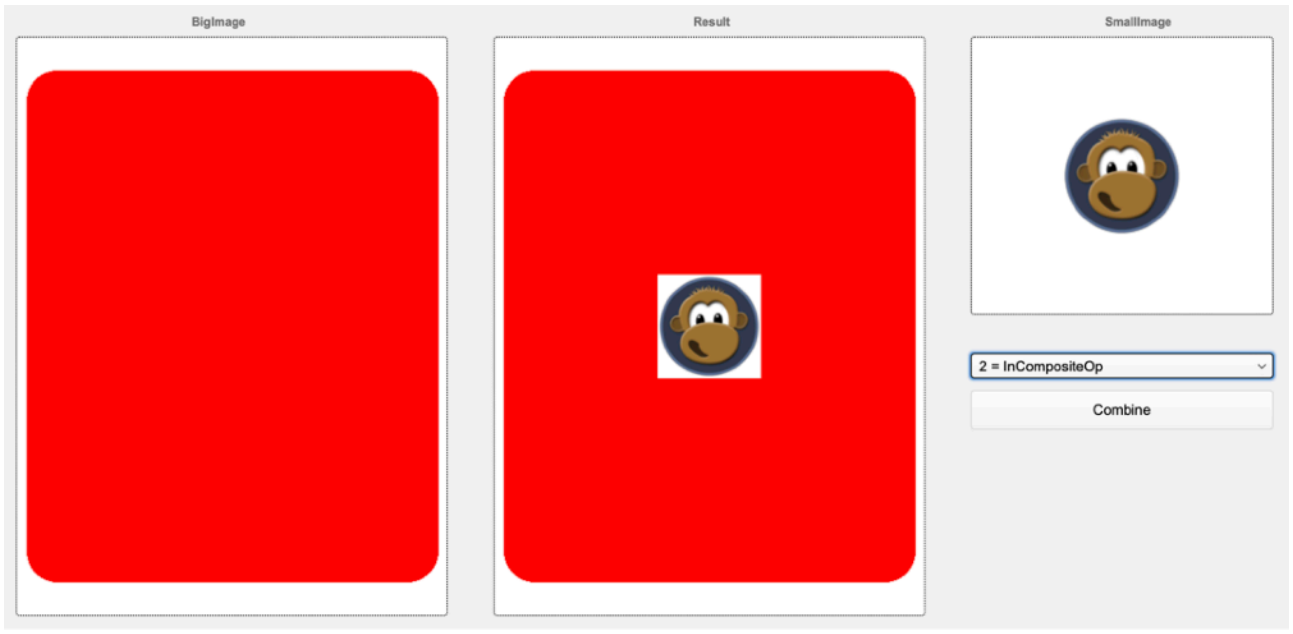
Optionally, we can now specify how the two images should be combined. We can choose from the following options:

- UndefinedCompositeOp = 0
- OverCompositeOp = 1
- InCompositeOp = 2
- OutCompositeOp = 3
- AtopCompositeOp = 4
- XorCompositeOp = 5
- PlusCompositeOp = 6
- MinusCompositeOp = 7
- AddCompositeOp = 8
- SubtractCompositeOp = 9

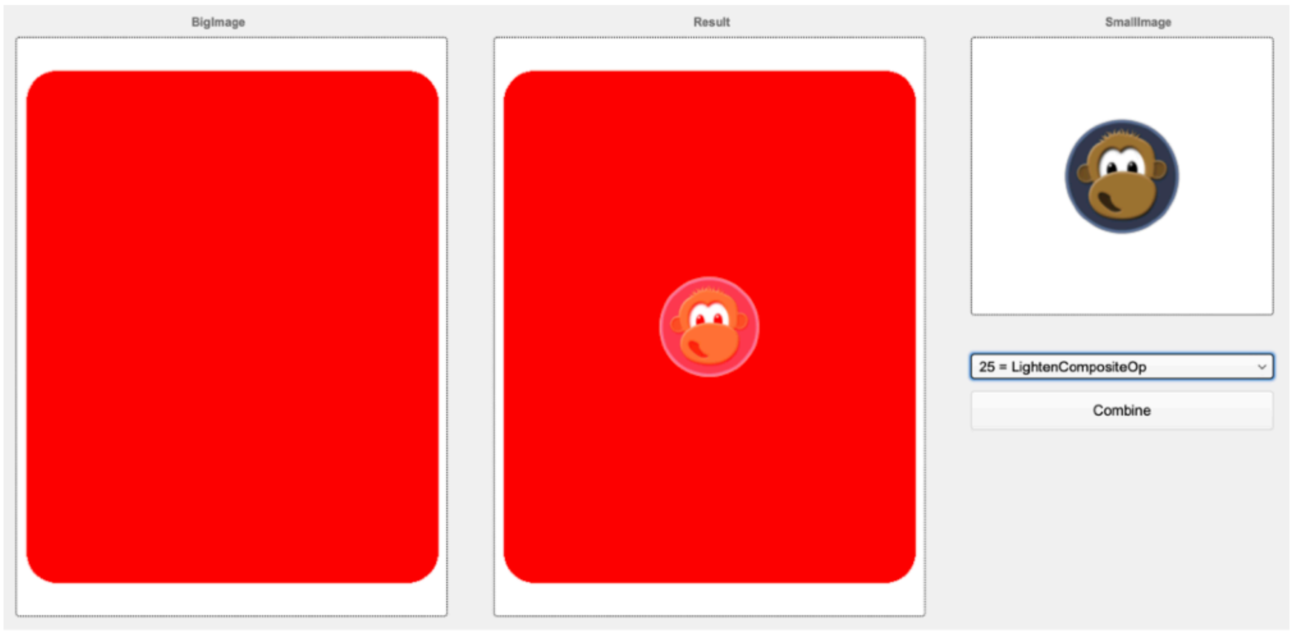
- DifferenceCompositeOp = 10
- MultiplyCompositeOp = 11
- BumpmapCompositeOp = 12
- CopyCompositeOp = 13
- CopyRedCompositeOp = 14
- CopyGreenCompositeOp = 15
- CopyBlueCompositeOp = 16
- CopyOpacityCompositeOp = 17
- ClearCompositeOp = 18
- DissolveCompositeOp = 19
- DisplaceCompositeOp = 20
- ModulateCompositeOp = 21
- ThresholdCompositeOp = 22
- NoCompositeOp = 23
- DarkenCompositeOp = 24
- LightenCompositeOp = 25
- HueCompositeOp = 26
- SaturateCompositeOp = 27
- ColorizeCompositeOp = 28
- LuminizeCompositeOp = 29
- CopyCyanCompositeOp = 32
- CopyMagentaCompositeOp = 33
- CopyYellowCompositeOp = 34
- CopyBlackCompositeOp = 35
- DivideCompositeOp = 36

We will not look at all of these options today, but if you have a general interest in how the other options work, please take a look at the Combine Pictures example. The following pictures are also taken from this example

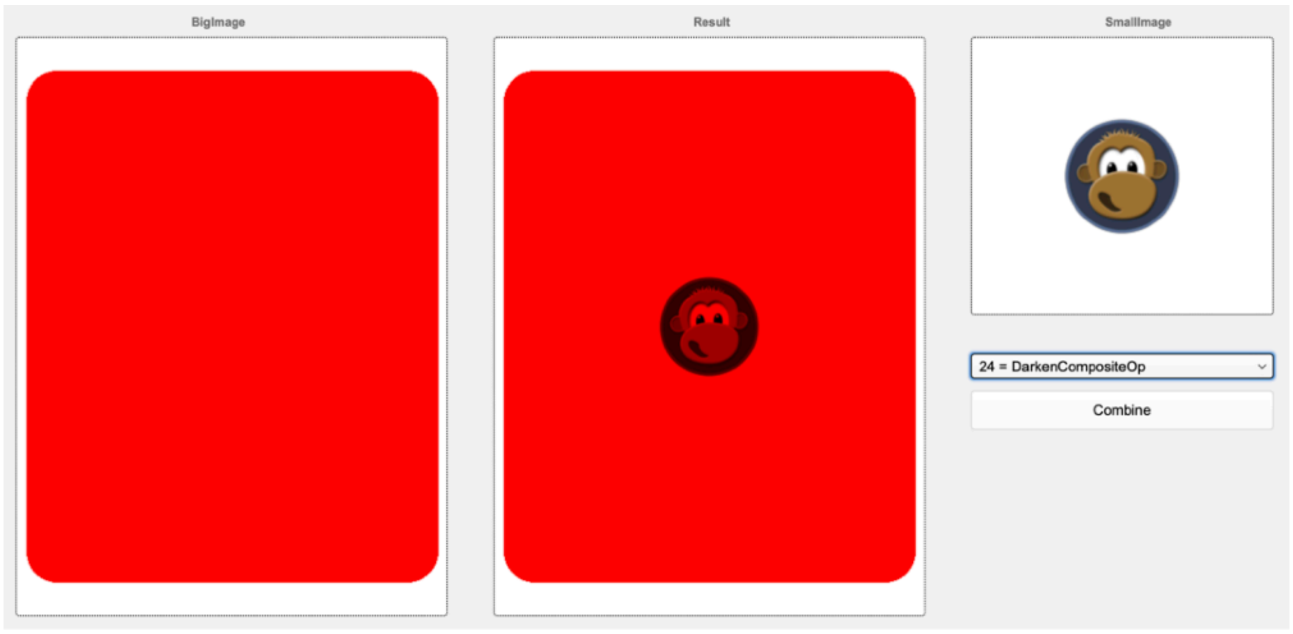
Let's start with the default value. This is InCompositeOp (2). With this composition we put the second image on top of the first one. The alpha channel information of image 2 is completely ignored and interpreted as white.



With LightenCompositeOp (27) we get the effect of a watermark. The image background becomes lighter in the places where there is color on the superimposed image, depending on the color value.



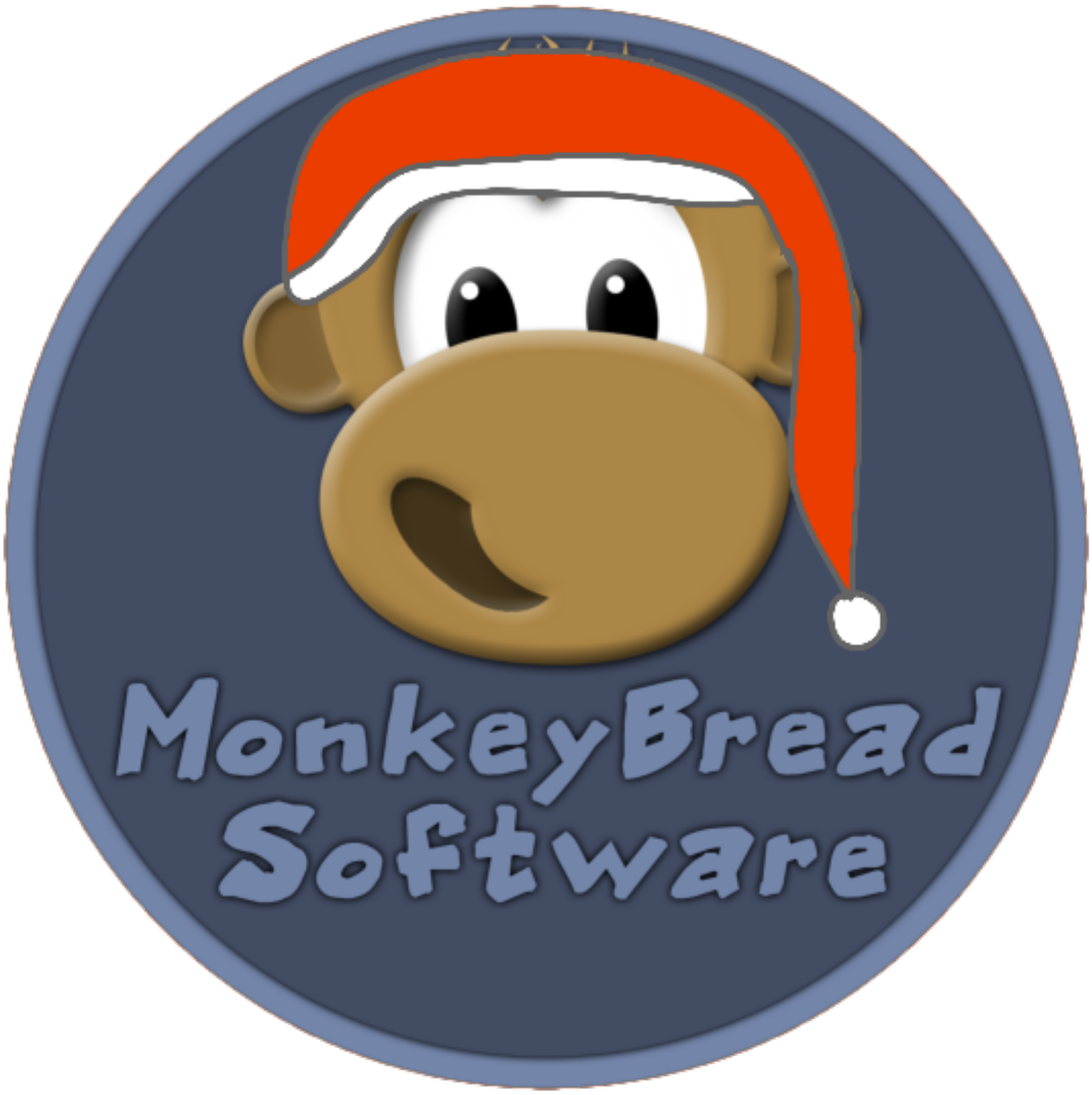
The situation is similar to DarkenCompositeOp. Here it is darkened depending on the color.



The most frequently used option is probably OverCompositeOp (1) This option applies image two to image 1, taking care of the alpha channel information. We also use this option to put the cap on our monkey. Our script looks like this:

```
Set Variable [ $GMLogo ; Value:
MBS("GMImage.NewFromContainer"; GraphicsMagick Advent::Logo)
]
Set Variable [ $GMCap ; Value:
MBS("GMImage.NewFromContainer"; GraphicsMagick
Advent::Cap) ]
Set Variable [ $r ; Value: MBS( "GMImage.Composite";$GMLogo;
$GMCap; 1; 1 ) ]
#
Set Field [ GraphicsMagick Advent::Image ;
MBS("GMImage.WriteToPNGContainer"; $GMLogo; "abc.png") ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

So now our result image looks like this:



We have just said that in addition to the GMImage.Composite function we can also use the GMImage.CompositeXY. Here, image2 will be placed on image1 with the help of an offset, specifying X and Y. So we don't need to specify the Gravity we used in the GMImage.Composite function. Instead we specify how we want to move image 2 in relation to the upper left corner of image 1. Our monkey still has some hair coming out of the top of the hat, we want to change this by moving the image in negative Y direction by 5 pixels. By the way, nothing changes in the overlay options, they are the same. Here we see the appropriate code:

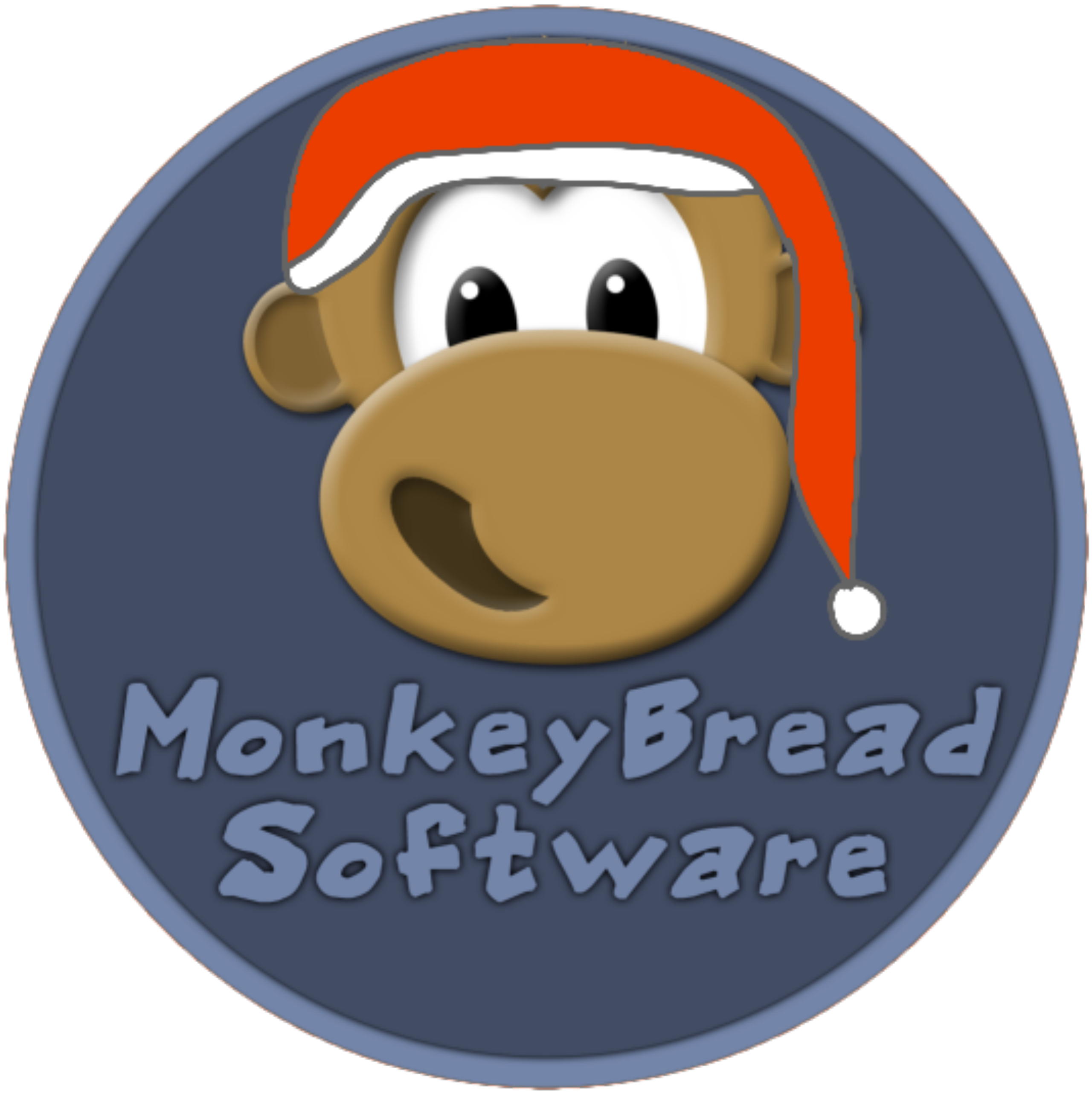
```
Set Variable [ $GMLogo ; Value:
MBS("GMImage.NewFromContainer"; GraphicsMagick Advent::Logo)
]
```

```
Set Variable [ $GMCap ; Value:
MBS("GMImage.NewFromContainer"; GraphicsMagick
Advent::Cap) ]

Set Variable [ $r ; Value: MBS( "GMImage.CompositeXY";
$GMLogo; $GMCap; 0; -5  ; 1  ) ]

Set Field [ GraphicsMagick Advent::Image ;
MBS("GMImage.WriteToPNGContainer"; $GMLogo; "abc.png") ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

Now the monkey's cap also fits perfectly.



I hope you liked the door and see you tomorrow.

GraphicsMagick in FileMaker, part 23

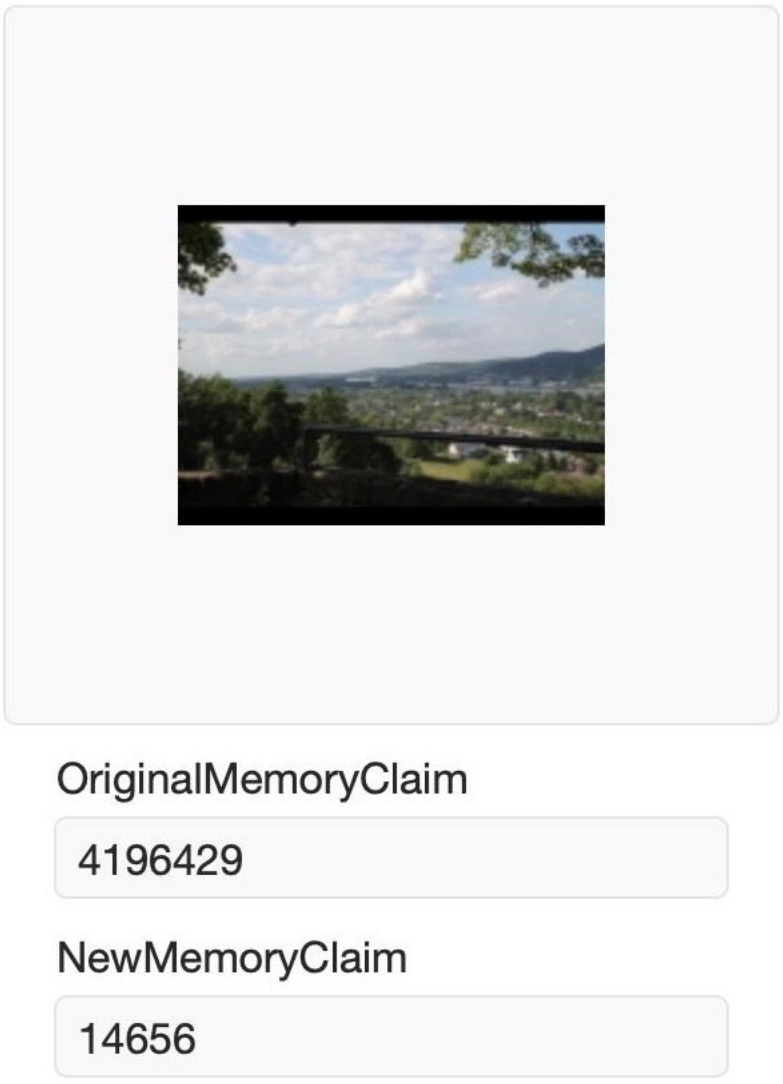
23 of 24

Welcome to the 23th door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I'm going to show you how to reduce the size of your image files. If you want to store images in your database, images often consume a lot of space. But sometimes the images don't have to be that huge, because you only want to display them in a container.

On one hand, if available, you can use the function "GMImage.ExifThumbnail" to retrieve the thumbnail and write it to the database instead of the original image. But with many images the thumbnail cannot be retrieved easily and an error occurs or we get back an empty value. So this case we should catch for sure.

```
...
Set Variable [ $IMG ; Value: MBS( "GMImage.ExifThumbnail";
$GM; "abc.jpg") ]
If [ MBS("IsError") or $IMG="" ]
    Show Custom Dialog [ "Error" ; "Thumbnail cannot be
retrieved¶" & $IMG ]
    Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
    Exit Script [ Text Result:    ]
End If
Set Field [ GraphicsMagick Advent::Image ; $IMG ]
...
```

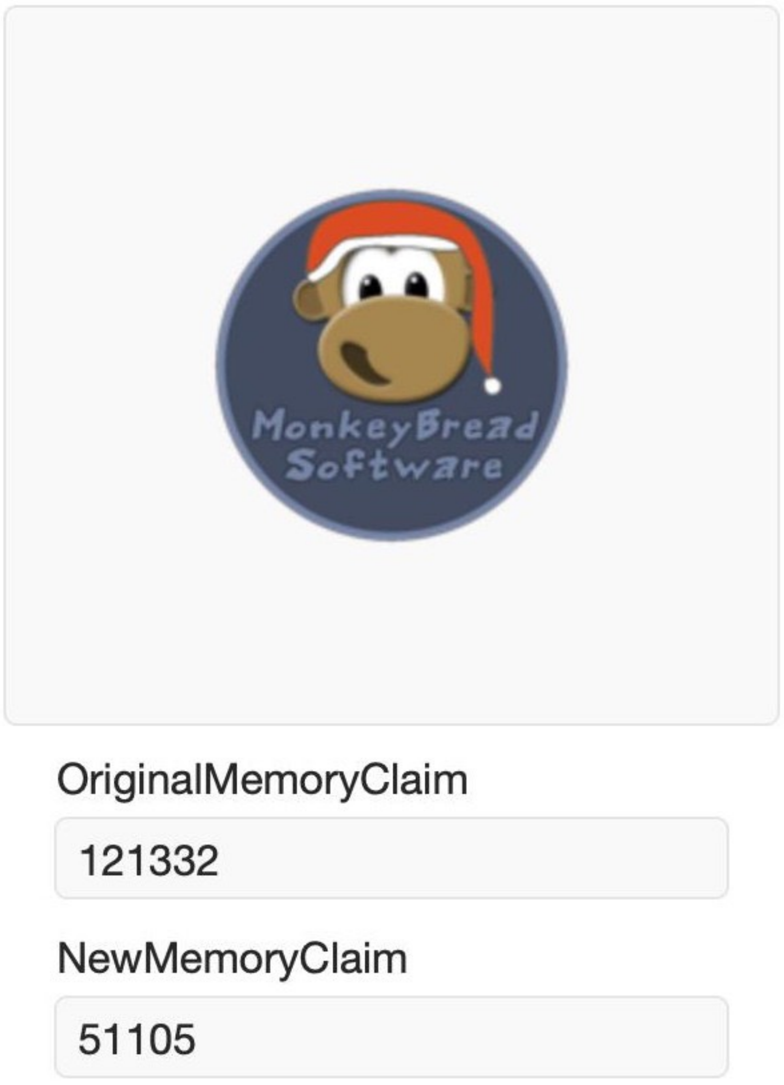

OriginalMemoryClaim

4196429

NewMemoryClaim

14656

Of course, we always have the possibility to scale our image, which means to change its physical size. How this works in detail we have seen in door 5.


OriginalMemoryClaim

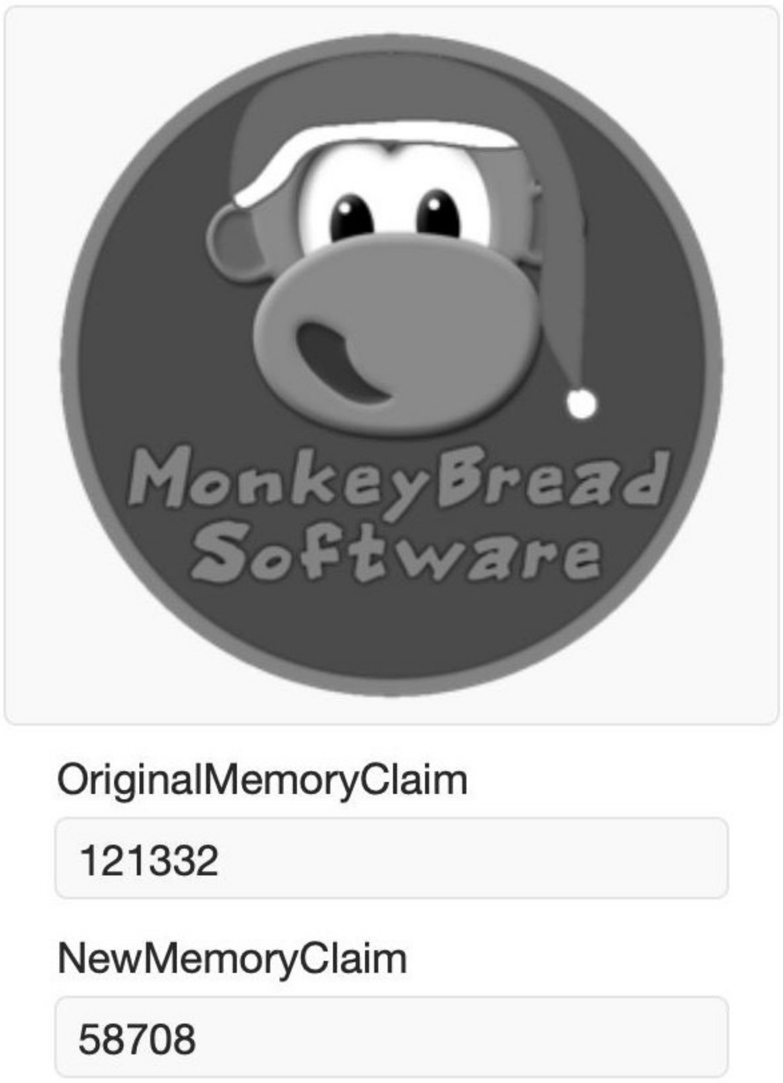
121332

NewMemoryClaim

51105

But we don't necessarily have to change the physical size to make our file smaller. For example, you can save an image as grayscale instead of RGB. This has the advantage that the color information in the grayscale is more compact. In an RGB pixel we have to store 24 bits of information (8 bit red, 8 bit green and 8 bit blue) for each pixel. In grayscale it is 8 bit. For this reason, the size is drastically reduced when we display the image in grayscale.

```
...
Set Variable [ $r ; Value: MBS( "GMImage.SetType"; $GM;
3 ) ]
...
```



OriginalMemoryClaim

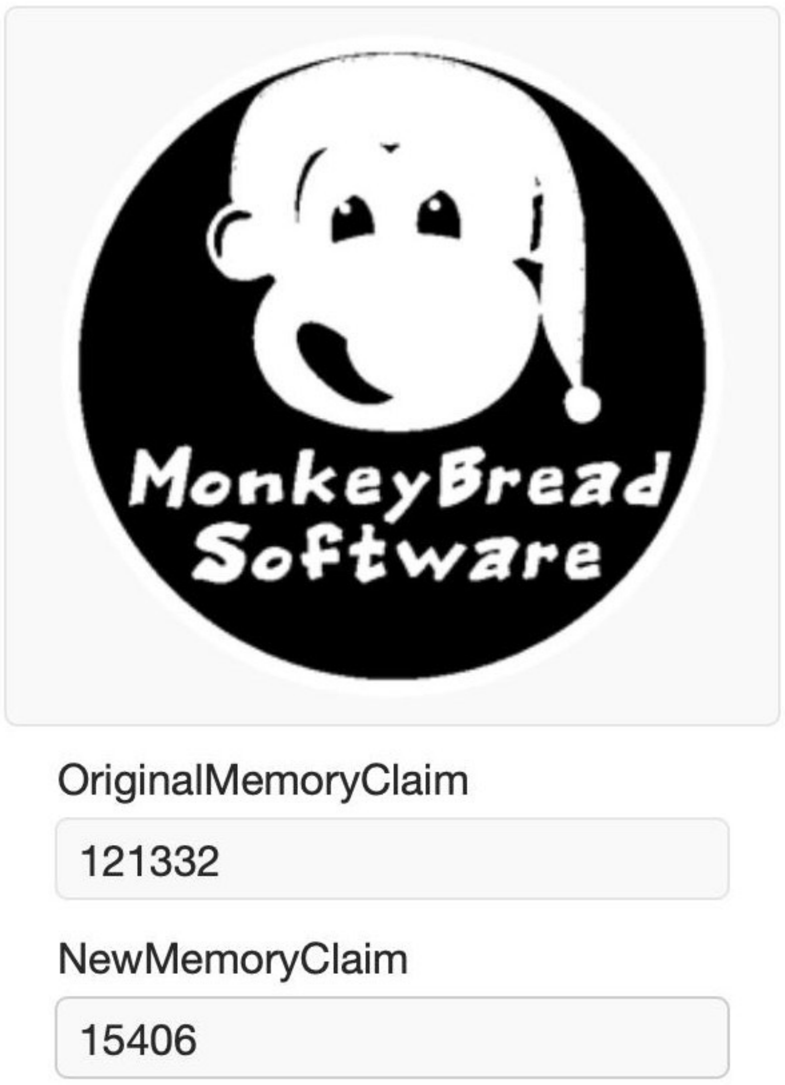
121332

NewMemoryClaim

58708

You can find more information about this function in door 8. Note that not every image that looks gray is also saved in the grayscale color space. Even in the RGB color space, an image can only have colors that are gray. So if you get an image from a scanner, don't count on the image being in grayscale format. You can also convert the image directly to black and white instead of grayscale format, which also saves memory. For this you use the same function as for grayscale, only this time you put a 1 in the parameters instead of the 3. This formats the image to black and white

```
Set Variable [ $r ; Value: MBS( "GMImage.SetType"; $GM;
1 ) ]
```



OriginalMemoryClaim

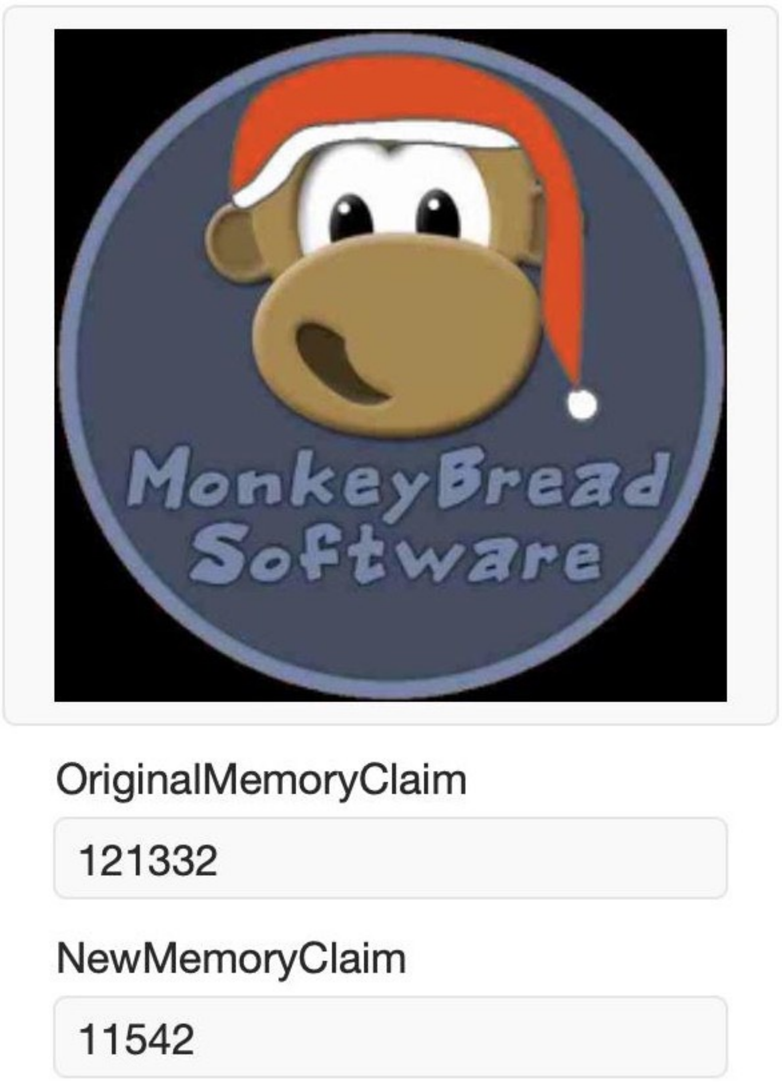
121332

NewMemoryClaim

15406

Another thing that can reduce the size of the image without having to physically reduce the size of the image and without having to sacrifice color is setting the quality of a JPEG image. A JPEG image has a built-in compression and we can now set the level of compression via the quality. For this we use GMImage.SetQuality function to set the compression level. If we don't use this function before we save the image as JPEG the default value is 75. Here you can see that we set the compression level to 50 before we write the image as JPEG back into the container.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer";
GraphicsMagick Advent::Image) ]
Set Variable [ $r ; Value: MBS( "GMImage.SetQuality"; $GM;
50 ) ]
Set Field [ GraphicsMagick Advent::Image ;
MBS( "GMImage.WriteToJPEGContainer"; $GM ) ]
Set Variable [ $r ; Value: MBS( "GMImage.Release"; $GM ) ]
```



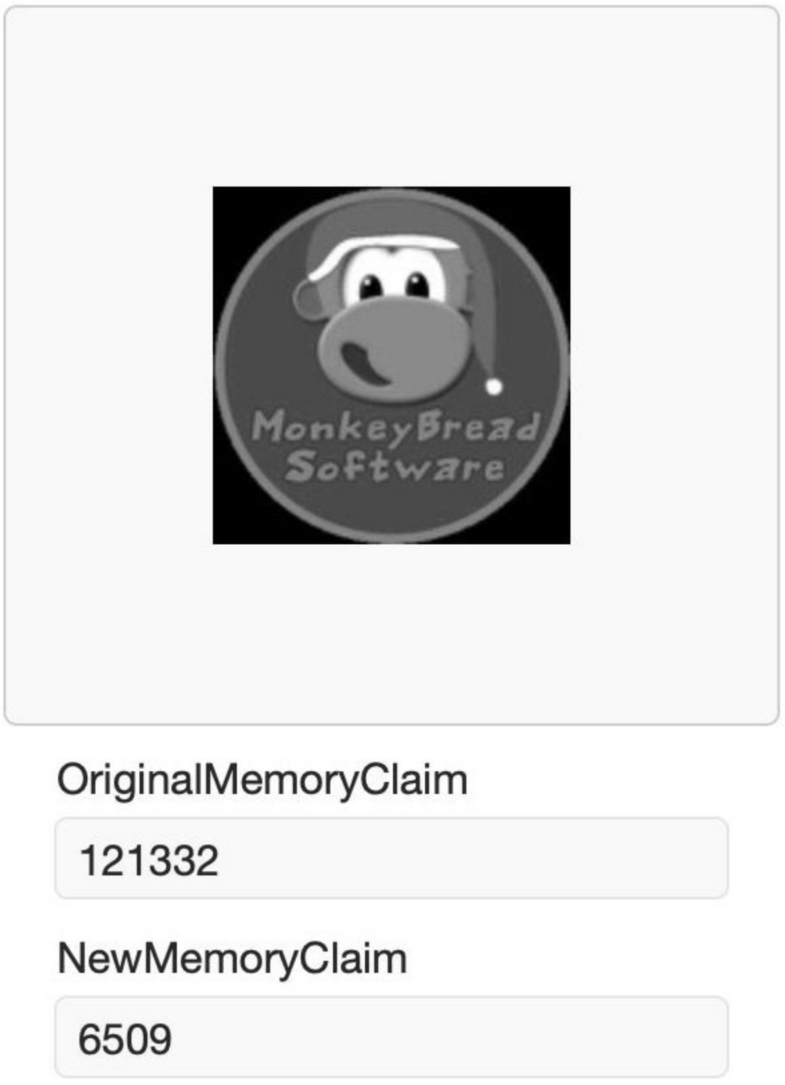
OriginalMemoryClaim

121332

NewMemoryClaim

11542

These reduction mechanisms can also be combined with each other. You can see this for example here with the image. First we reduced the physical size, then the image was converted into a grayscale image and then with a quality of 50 converted into a JPEG image.



OriginalMemoryClaim

121332

NewMemoryClaim

6509

Here you must be careful that sometimes the mechanisms do not work well together, for example, if you convert a JPEG image into a black and white image, the file size may even increase.

I hope you liked this door and it helps you. Be happy to join us tomorrow when we open the last door where the whole thing becomes round.

GraphicsMagick in FileMaker, part 24

24 of 24

Welcome to the 24th and last door of our advent calendar. In this advent calendar I would like to take you on a journey through the GraphicsMagick component in December. Every day I will introduce you to one or more functions from this component. In this component you will find functions with which you can analyze images, convert them, change them with filters, draw them and much much more. In the end, you too can take the magic of GraphicsMagick to your images. I wish you a lot of fun in the process.

Today I will show you how to turn a rectangular image into a round one. To say it at the beginning. There is no such thing as a round image. The illusion that an image is round is created by the alpha channel. This means that certain areas are transparent.

First, we load the image from the container again and determine the width and height of the image.

```
Set Variable [ $GM ; Value: MBS("GMImage.NewFromContainer"; GraphicsMagick Advent::Image) ]
Set Variable [ $Width ; Value: MBS( "GMImage.GetWidth"; $GM ) ]
Set Variable [ $Height ; Value: MBS( "GMImage.GetHeight"; $GM ) ]
```

We first need the image as a square. So we determine the size of the shorter of the two sides. For this, we distinguish our procedure for Landscape and Portrait mode. With the function "GMImage.Crop" we crop the image as we already know it from door 7.

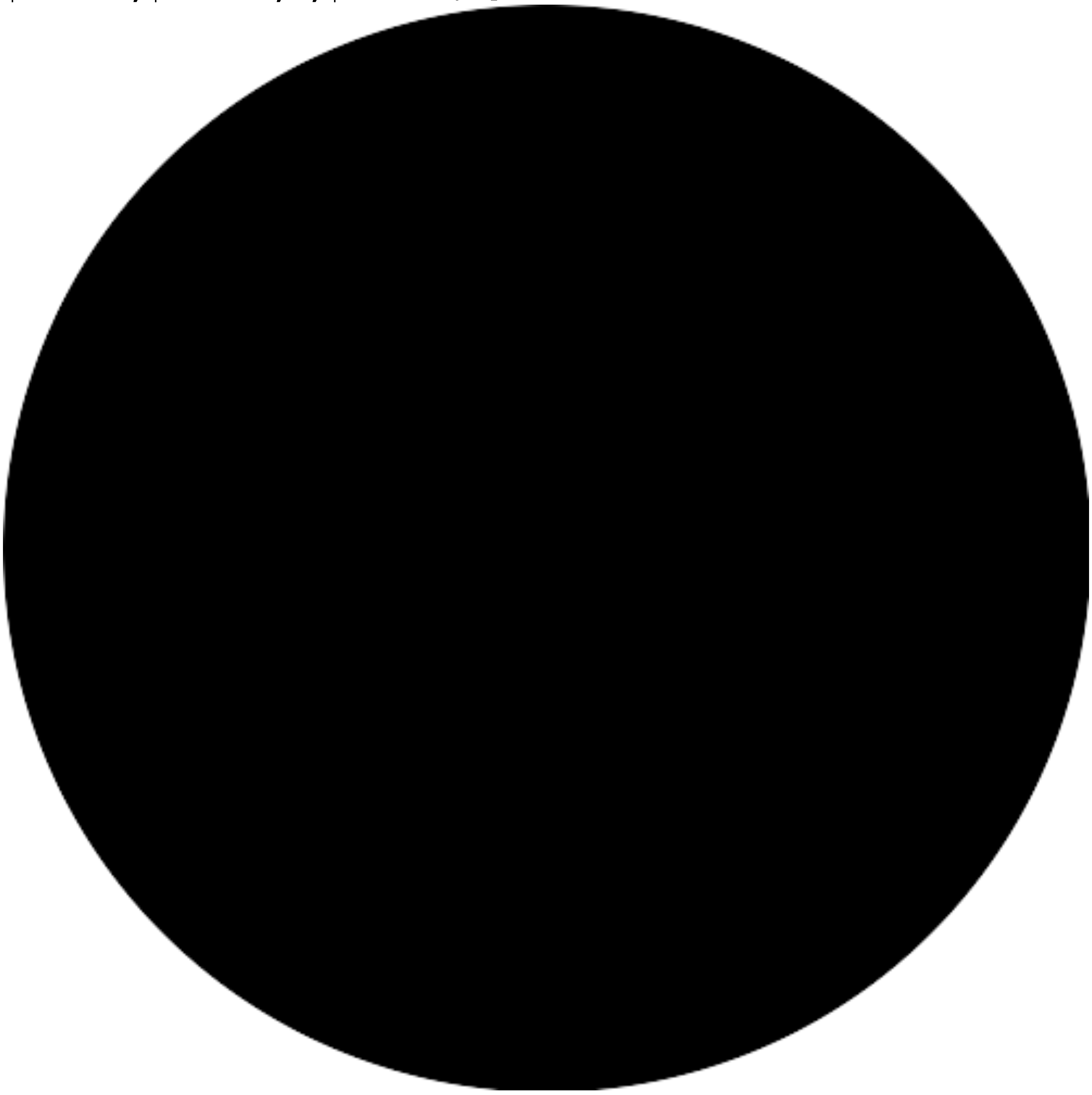
```
If [ $Width>$Height ]
    # Landscape
    Set Variable [ $SizeGeometry ; Value: $Height & "x" & $Height ]
    Set Variable [ $Size ; Value: $Height ]
    Set Variable [ $OffsetX ; Value: ($Width - $Height) / 2 ]
```

```
     Set Variable [ $OffsetY ; Value: 0 ]
Else
     # Portarait
     Set Variable [ $SizeGeometry ; Value: $Width & "x" &
$Width ]
     Set Variable [ $Size ; Value: $Width ]
     Set Variable [ $OffsetX ; Value: 0 ]
     Set Variable [ $OffsetY ; Value: ($Height - $Width ) / 2
]
End If
Set Variable [ $Geometry ; Value: $SizeGeometry & "+"
&$OffsetX& "+" &$OffsetY ]
Set Variable [ $r ; Value: MBS( "GMImage.Crop"; $GM;
$Geometry ) ]
```

This way, our image is now already cut square.

Next, we create a new GraphicsMagick workspace with a transparent background. In this environment we draw a black circle with the diameter of our square. Accordingly, the radius we have to specify for drawing is $Size/2

```
Set Variable [ $Cut ; Value: MBS("GMImage.New";$Size & "x" &
$Size; "transparent") ]
Set Variable [ $r ; Value: MBS("GMImage.SetFillColor";
$Cut;"black") ]
Set Variable [ $radius ; Value: $Size/2 ]
Set Variable [ $r ; Value: MBS("GMImage.DrawCircle";$Cut;
$radius;$radius;0;$radius) ]
```



This black circle defines the image section that we want to see later from the other image. That is why it is so important that the background of our circle is also transparent. On December 22nd we have already learned about the function GMImage.CompositeXY and we know that this function is very powerful. We use it again today. We put the image we want to see on our circle with option 2. We don't have an offset, because the images have the same size. With this combination we only see the pixels from the image where our circle is. The transparent pixels are not overwritten. Our desired result is then in the workspace $Cut.

```
Set Variable [ $r ; Value: MBS( "GMImage.CompositeXY"; $Cut; 
$GM; 0; 0  ; 2  ) ]
```

Finally we write our working environment $Cut with the function GMImage.WriteToPNGContainer into the container that shows us our round image. At the end we must not forget to release the various working environments.

```
Set Field [ GraphicsMagick Advent::Image ; 
MBS( "GMImage.WriteToPNGContainer"; $Cut ) ]
Set Variable [ $r ; Value: MBS( "GMImage.ReleaseAll" ) ]
```

Now we have reached the advent. I hope you enjoyed it and you can use one or the other of what we have shown here in your everyday work. If you have any questions about these or other features, please feel free to contact us.
Until then, we wish you a Merry Christmas and a Happy New Year.

Download sample project