

Drag and drop in combination with overlays

Today I would like to introduce you to a really cool function from the latest release of the [MBS FileMaker Plugin](#). It's about the [DragDrop.AttachToOverlay](#) function. This function combines 2 areas from our plugin. The possibilities of drag and drop and our overlay windows.

We introduced our overlays for the first time at the beginning of this year. These are areas that you can create on the screen, give them a background, which you can move freely as you wish and which you can use with several mouse events, such as click, enter or exit. At the beginning of the year, we published the blog article [Overlay windows for FileMaker](#). The second part we want to talk about today is drag and drop. This plugin part allows you to create drop zones into which you can drag files, for example, which can be processed with a script trigger. You can now also create a drop zone on an overlay.

Today I would like to show you how this works. To do this, we will create an example where each file that is dragged onto this overlay creates a new record in which the path of this file is stored.

We start by creating the overlay. We create the overlay with the MBS function [Overlay.Create](#) and receive a reference number as a return, which we can then use to work with. Now we set the size of the overlay with the [Overlay.SetFrame](#) function. In the parameters, we first specify the overlay reference to which this action should now be applied. Next, we specify the point of the upper left edge (20/20) of the overlay at which it should appear. In two further parameters we define the size (400x400). So that we can later move the layout on our screen with the mouse, we have to set this with the function [Overlay.SetMovableByWindowBackground](#). We also specify our reference here again and enter a 1 in the parameters to activate the movability. If we want to deactivate it again, we have to set this value to 0.

```
Set Variable [ $Overlay ; Value: MBS( "Overlay.Create" ) ]
Set Variable [ $r ; Value: MBS("Overlay.SetFrame"; $Overlay;
20; 20; 400; 400) ]
Set Variable [ $r ; Value:
MBS("Overlay.SetMovableByWindowBackground"; $Overlay; 1) ]
```

Now we want this area to be visible and we can achieve this by assigning it a background. We first load this background image from a file on our hard disk into a variable. To do this, we use the [Container.ReadFile](#) function and specify the path of the image in the parameters. We then use the [Overlay.SetImage](#) function to set this

image as background. The image is automatically scaled. If we run this script now, however, we won't see anything because we haven't shown the overlay yet. To do this, we use the [Overlay.SetVisible](#) function and enter 1 in the parameters.

```
Set Variable [ $img ; Value: MBS( "Container.ReadFile"; "/"  
Users/sj/Desktop/logo.png" ) ]  
Set Variable [ $r ; Value: MBS("Overlay.SetImage"; $Overlay;  
$img) ]  
Set Variable [ $r ; Value: MBS("Overlay.SetVisible";  
$Overlay; 1) ]
```

We can now see our overlay and can move it.



What is noticeable is that we do not yet have the option of hiding the overlay again. Even if we close our FileMaker file, this overlay remains. So we have to create a second button with which we can remove the overlay again. We have several options for this. On one hand, we can remember the reference in a global variable and use the function [Overlay.SetVisible](#) in another script. In the case that we want to hide the overlay, we enter 0. In this case, the overlay is hidden but still exists in the background and can be shown again later. However, we can also remove the overlay permanently and release the reference again. To do this, we use the [Overlay.Release](#) and [Overlay.ReleaseAll](#) functions. [Overlay.Release](#) releases a specific overlay that we specify using the reference. [Overlay.ReleaseAll](#), as the name suggests, releases

all references. We have used the [Overlay.ReleaseAll](#) function behind our button, because we only want to use our reference locally in a script and do not need to remember it with [Overlay.ReleaseAll](#).

But back to our original script and our drag and drop area.

Now we want to be able to drag files onto this overlay, which are then created as data records with their paths. To do this, we now create this drop zone on our overlay. To do this, we use the [DragDrop.AttachToOverlay](#) function and specify our overlay reference in the parameters. The function then provides us with a reference number for the drop zone that we can use for further work. We save this in a global variable, because we need it in another script. We describe what should happen when a file is dragged onto the overlay in the "Action" script. We assign this script to a drag action with the function [DragDrop.SetDragActionHandler](#).

```
Set Variable [ $$Drop ; Value:  
MBS("DragDrop.AttachToOverlay"; $Overlay) ]  
Set Variable [ $r ; Value:  
MBS("DragDrop.SetDragActionHandler"; $$Drop; Get(FileName);  
"Action") ]
```

When a file is dragged onto the area, the "Action" script is called. Let's take a look at this now.

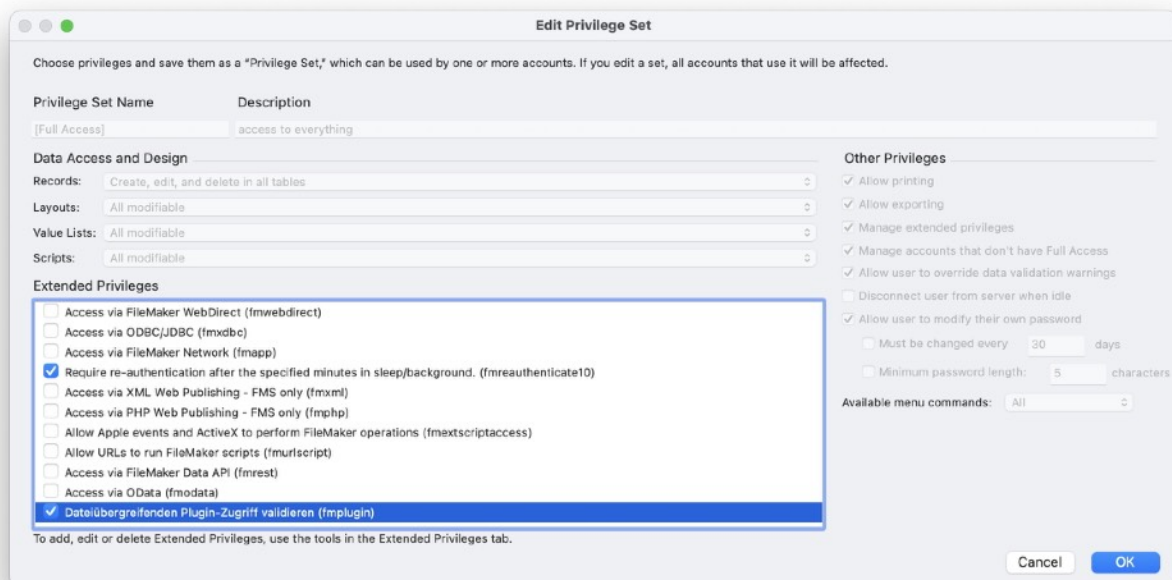
```
Set Variable [ $count ; Value: MBS("DragDrop.GetPathCount";  
$$Drop) ]  
Set Variable [ $i ; Value: 0 ]  
Loop [ Flush: Always ]  
    Exit Loop If [ $i ≥ $count ]  
    Set Variable [ $Path ; Value: MBS("DragDrop.GetPath"; $  
$Drop; $i) ]  
    If [ $Path ≠ "" ]  
        New Record/Request  
        Set Field [ DragDropOverlay::Path ; $Path ]  
    End If  
    Set Variable [ $i ; Value: $i + 1 ]  
End Loop
```

First, we determine how many elements were dragged onto the overlay during the action using the [DragDrop.GetPathCount](#) function. We run through these individual elements and use [DragDrop.GetPath](#) to determine the path for each element. If this is not empty, we create a new record and write the path in the Path field.

Time	Path	Show Overlay	Release Overlay
29.07.2024 17:05:02	/Users/sj/Desktop/Create PDF with Barcodes.pdf		
29.07.2024 17:12:40	/Users/sj/Desktop/14.3.pages		
29.07.2024 17:12:40	/Users/sj/Desktop/14.3en.pages		
29.07.2024 17:12:59	/Users/sj/Desktop/Unsere erste eigene App.mp4		



Note: If the script action is not called when you drag a file onto the overlay, please check whether you have activated the extended privilege fmplugin. You need this so that the plugin can call another script via a script trigger.



I hope you have enjoyed this excursion into the world of overlays and DragDrop. I look forward to seeing how these techniques can help you in your day-to-day work.