Adding Drag and Drop receiver to a layout

One of our clients asked how to add a Drag & Drop area on the layout to receive file drops.

Well, the obvious answer is to copy everything from DragTest Windows.fmp12 to their solution and enjoy it for both Mac & Windows. But it's not that simple as you need to know where to look. So here a step by step solution.

Before you start, please make sure you have a recent plugin version. The Drag & Drop features for Windows were introduced in MBS FileMaker Plugin 8.1, but we got bug fixes later, so best is to use current 8.4 release.

Add placeholder rectangle

First in our DragTest Windows.fmp12 file is a picture used for the drop area. The picture is there twice. Once assigned to the rectangle in the layout and second in a global field. So add the DropLogo field to the layout and you can export the image named "Drop-Files-Here-extra.jpg".

In your solution, you may find a layout where you like people to drop stuff. This should be a layout where you would enable Drop in one trigger and than disable it when the user leaves the layout.

To the layout, we add a rectangle area in the header for the drop. Our image is 500 x 224, so our rectangle should be 250 x 112 points big to use 2x image for sharp picture on high resolution screens. In the inspector for fill option, please select picture and select the image file. For the aspect ratio, please choose "Scale to Fit" below. In the first inspector tab, please give the rectangle a name, e.g. "drop".

Please add a global container image named DropLogo. Put it on the layout and import the image here, too. After the import, you can remove the global field from the layout. We need this image later in the script. It's a good idea to actually have two images, one of drop off and one for drop on. This way you can see whether the plugin is working.

Two scripts

Lets add the first two scripts. Please add an empty script named "FreeDrop" and one named "Setup Drop". The names can be different, but you must be consistent and later use the right names in the function calls. For the layout, please edit it and go to the Layout Setup dialog. There pick the SetupDrop script for the OnLayoutEnter trigger. Connect the FreeDrop script with the OnLayoutExit trigger. For debugging you may need to run them directly without trigger.

Lets start with the **FreeDrop** script:

```
If [ Length($$dropView) > 0 ]
```

```
    Set Variable [ $result ; Value: MBS( "DragDrop.Release"; $$dropview) ]
    Set Variable [ $$dropview ; Value: "" ]
End If
```

As you see we plan to keep the reference number for the drop area in the global variable $ $dropView. If it is a number bigger than zero, we call DragDrop.Release function and free it. Than we clear the variable. With the If you can run the script several times without problems, e.g. call it before doing an action, which needs the drop area removed.

Now we add lines for the **SetupDrop** script. Don't type it, but please copy & paste from our example database. The script looks like this:

```
Set Variable [ $$dropview ; Value: MBS("DragDrop.CreateWithControl";
Get(WindowName); "drop") ]
Set Variable [ $result ; Value: MBS( "DragDrop.RegisterDropTypes"; $$dropview; "file") ]
Set Variable [ $result ; Value: MBS( "DragDrop.SetDragActionHandler"; $$dropview; Get
( FileName ); "DragAction") ]
Set Variable [ $result ; Value: MBS( "DragDrop.SetCursor"; $$dropview; 1) ]
Set Variable [ $result ; Value: MBS( "DragDrop.SetImage"; $$dropview;
DragTest::DropLogo) ]
```

We start by creating a new drop area control. We pass in the name of the window. The plugin than looks in the window list to find it, so the name should be unique. With "drop" we specify for which layout object to look for. The plugin will query coordinates from FileMaker and try to place the control as good as possible over the existing layout object.

Next line we declare what we want to receive. This is usually file and only applies to Mac where we can choose between receiving text, pictures or file and get the right cursor. With DragDrop.SetDragActionHandler we define which script to call when something arrives. We pass current file name and the script name DragAction. We set the cursor to copy (value 1) and than set the image of the container field.

You can test this already. If you run the SetupDrop script, the image fo the drop area should change a bit to show the image from the global field. If you run FreeDrop script, it should revert back.

Fields

Our example needs the following fields. First a container field for the actual file data we receive. Than a text field for the file path or name to show to the user. And for our example we also have a description field to show how we got the file, but you don't need that.

So please add the container and text field to your database table or reuse existing fields there and change the following script.

DragAction Script

Here we have now the big script to handle the drops. The script has 4 sections. First a bit of setup to get the drop view ID from the script parameter and prepare the names list as variable with just an empty entry.:

```
Set Variable [ $dropview ; Value: Get(ScriptParameter) ]
Set Variable [ $names ; Value: ¶ ]
```

Second part accepts file paths dropped on our rectangle and imports the files. If we get a file path and the file is not yet here, we wait a second to the other application write it. Anyway we read file path and put file name in the names list. This way we can avoid duplicates if a file is provides two ways on Windows. Next we create a record and add the file as container value to the container field. The loop runs til all files are processed:

```
# Check for paths in Drag & Drop
Set Variable [ $count ; Value: MBS("DragDrop.GetPathCount"; $dropview) ]
If [ $count > 0 ]
    Set Variable [ $index ; Value: 0 ]
    Loop
        Pause/Resume Script [ Duration (seconds): ,1 ]
        Set Variable [ $path ; Value: MBS("DragDrop.GetPath"; $dropview; $index) ]
        If [ Length ( $path ) > 0 ]
            If [ MBS("Files.FileExists"; $path) = 0 ]
                # File not yet written? Wait a bit
                Pause/Resume Script [ Duration (seconds): 1 ]
            End If
            Set Variable [ $name ; Value: MBS( "Path.LastPathComponent";
$path) ]
            Set Variable [ $names ; Value: $names & $name & ¶ ]
            New Record/Request
            Set Field [ DragTest::field ; MBS("Files.ReadFile"; $path; "auto") ]
            Set Field [ DragTest::Text ; $path ]
            Set Field [ DragTest::Description ; "Got via path list" ]
            Commit Records/Requests [ With dialog: Off ]
        End If
        # next
        Set Variable [ $index ; Value: $index+1 ]
        Exit Loop If [ $index = $count ]
    End Loop
End If
```

The third part of the script checks for file descriptors. The MBS plugin can receive promised file drops. For Mac, we convert them to real files for you and provide them with the path list above. For Windows we get file descriptor which describes the file. If you ask the plugin for size, path or data, the plugin will request the file and the data is transferred in memory without disk usage. The file descriptor may describe a file on disk or in memory, so we handle both in the script: Either we get a file path and read the file or we get it a container and put it in the field right away:

```
# Check for a file descriptor coming with Drag & Drop
Set Variable [ $count ; Value: MBS("DragDrop.GetFileDescriptorCount"; $dropview; "") ]
If [ $count > 0 ]
    Set Variable [ $index ; Value: 0 ]
    Loop
        Pause/Resume Script [ Duration (seconds): ,1 ]
        Set Variable [ $name ; Value: MBS("DragDrop.GetFileDescriptor"; $dropview;
$index; "name") ]
```

```
                Set Variable [ $data ; Value: MBS("DragDrop.GetFileDescriptor"; $dropview; 
$index; "data") ]
                Set Variable [ $size ; Value: MBS("DragDrop.GetFileDescriptor"; $dropview; 
$index; "size") ]
                Set Variable [ $path ; Value: MBS("DragDrop.GetFileDescriptor"; $dropview; 
$index; "path") ]
                Set Variable [ $pos ; Value: Position ( $names; ¶ & $name & ¶; 1; 1) ]
                If [ $pos = 0 ]
                    # not a duplicate
                    If [ Length ( $path ) > 0 ]
                        New Record/Request
                        Set Field [ DragTest::field ; MBS("Files.ReadFile"; $path; 
"auto") ]
                        Set Field [ DragTest::Description ; "Got via file descriptor with 
path" ]
                        Set Field [ DragTest::Text ; $path ]
                        Commit Records/Requests [ With dialog: Off ]
                    Else If [ $size > 0 ]
                        New Record/Request
                        Set Field [ DragTest::field ; $data ]
                        Set Field [ DragTest::Text ; $name ]
                        Set Field [ DragTest::Description ; "Got via file descriptor with 
data" ]
                        Commit Records/Requests [ With dialog: Off ]
                    End If
                    Set Variable [ $names ; Value: $names & $name & ¶ ]
                End If
                # next
                Set Variable [ $index ; Value: $index+1 ]
                Exit Loop If [ $index = $count ]
        End Loop
End If
```

In the forth part, we check for a text drop:

```
# Check for text coming with Drag & Drop
Set Variable [ $text ; Value: MBS("DragDrop.GetText"; $dropview) ]
If [ Length($text) > 0 ]
        New Record/Request
        Set Field [ DragTest::Text ; $text ]
        Set Field [ DragTest::Description ; "Got text" ]
        Commit Records/Requests [ With dialog: Off ]
End If
```

With that script, we are done. Please enjoy and if you have questions, please don't hesitate to contact us.