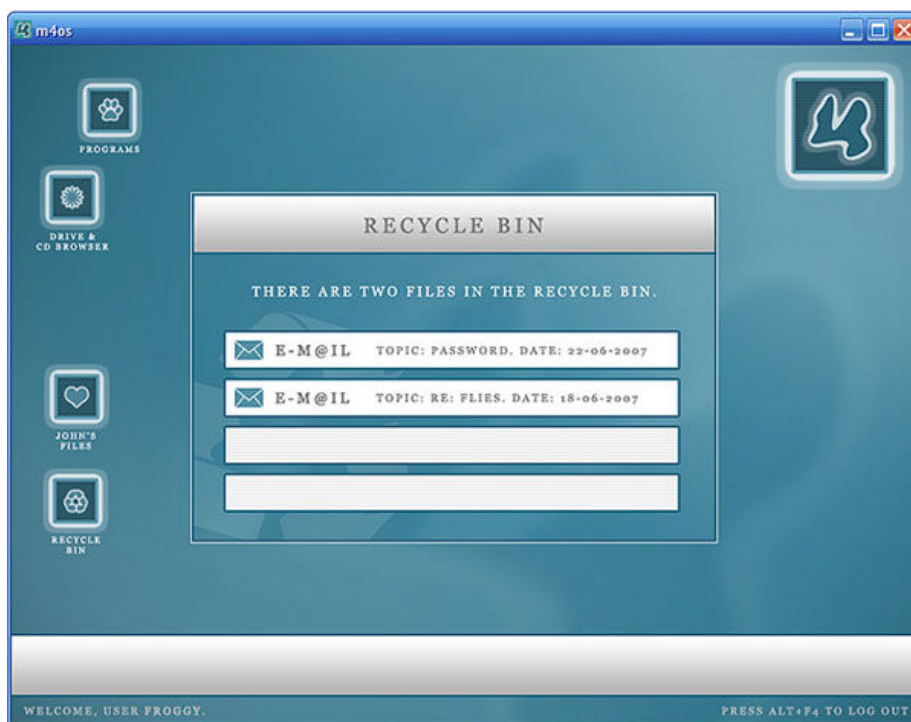


M4OS

a Multimedia fusion 2 tutorial



You may not use this tutorial for any other purpose than learning, working or having fun... In other words: You can use this PDF tutorial for anything You'd like, as long as it doesn't involve both a hammer and a squirrel.

Koobare
marchewkowy@gmail.com

Hi there, all!

Welcome to another one of Koobare's little tutorials, teaching you how to effectively and efficiently use the best multimedia authoring tool ever – [Multimedia Fusion 2](#) by Clickteam! This tutorial is going to explore a couple of MMF2 activities – such as importing alpha-blended images from a PNG file, using Edit and Sub-Application objects, making a simple drag'n'drop system inside your app, and more – in sole purpose of creating an... operating system. Huh? Sounds weird, right? Can MMF2 even do that? Well, don't be troubled, since we're not actually gonna' create a real OS, just a simple imitation, a fake one, that can be used in your projects. What for? Let me explain... And may this be a thorough explanation.

If there's a single thing that determines whether a game is successful for me or not – it's the capacious definition hidden behind the word “**immersion**”. Heck, sure, playing almost any game can be quite fun, but until they get me immersed – and I mean REALLY immersed – it's just another computer game, one of the thousands I've played in my life. “Nothing special here, bub, just finish it already and let's move on to another time-eater” – I think to myself. But... Boy, let me tell ya' what happens if a game really gets under my skin... Once the “immersion factor” kicks in – it's like eating a tasty pie with extra crème and a sugar cherry right on top of it. For the duration of those few minutes every week, I become the savior of The Swords Coast in the classic “Baldur's Gate”, the commander of the last line of global defence in “UFO: Enemy Unknown” (known also as “X-Com: UFO Defence” in some countries), the super-spy in “Splinter Cell”, the mysterious force-user from “Knights of the Old Republic”, the... Well, you got the picture. It's not just a “pleasant gaming time”. It becomes an experience. Take a great story, set it in an interesting world – and then add a couple elements that truly make the player involved, either by challenging him in a way, or making him FEEL that he's a part of what's happening on screen. As far as I'm concerned – if you've done it right, you've got an instant classic. And don't be fooled that I've only listed big-budget, big-company titles here – it's for easier recognition, so that all of you can get the picture. There's a lot less-known garage- or homemade titles on my “this game was truly a great experience” list too.

You know what was the single most memorable element from “Jagged Alliance 2” for me? Err, besides the overallly great gameplay, that is. It was **the laptop**. The fact that all the main management duties were done on a fully controllable, in-game laptop with an operating system that looked like a crossover between older versions of Windows and Mac. The fact that I could receive e-mails from Enrico Chivaldori, the man who hired my little team of mercenaries, containing either useful information or just complaints about how slow my progress is. The fact

that I had to both hire my soldiers and order guns via the internet – and wait for their arrival afterwards. All of this was pretty brilliant – in a simple way, they’ve got me involved. I was sitting there – on the tropical island of Arulco – waiting for a shipment of grenades from Bobby Ray’s Guns & Ammo, checking my e-mails and intelligence data on the finest laptop the world has ever seen. It was just that way cooler than the usual “four buttons and a slider” game interface. Heck, in “Jagged Alliance 2.5: Unfinished Business” they’ve even made the laptop a part of the scenario, since – after your chopper crashed in the mountains – you had to look for a battery to make it work again! How cool is that?

Yup, you most probably know where I’m going with this by now. Our OS – the one we’re going to create – will be a system that the player has to operate to get some important in-game data. The story goes like this... Our hero is a trouble-magnet, a small-frame reporter for a local newspaper, just trying to make a living. Think Peter Parker – but without the tights, spider-sense and any other superpowers. Accidentally, our guy – let’s say his name is Jeremy – discovers a dark plot, a mysterious organization trying to take over the world, disguised as a successful computer company, specialized in creation of multimedia authoring software. In a typical style of a classic point’n’click adventure game, Jeremy finally finds the headquarters of the evil organization – a mutant factory (yaarp!) – and tries to break in. Unfortunately, the door’s locked with a password and this time combining a crowbar with chewing gum and latex gloves won’t solve the puzzle. The player has to order Jeremy – by clicking on a nearby computer – to search for the password or a clue. And here’s where our OS – I’ve named it M4OS – kicks in. Our player has to log in as a guest and – not having any other options, since file-browsing is restricted to the administrator’s account only – check the e-mails dumped into the system’s recycle bin. In which – of course – the password to the mutant lab is ludicrously waiting to be discovered.

Sounds easy, right? Sure it does, since it’ll be easy – if you know your MMF2 basics, you should do just fine. If not – whatcha’ waiting for, download the “MMF2 Interface Guide” from Clickteam’s website and learn why Multimedia Fusion 2 is the best tool in it’s category! And hurry up, private, we don’t have all day!

To sum it all up – here are some core features of the project we’re going to complete:

- We’re going to create a good lookin’ imitation of an operating system. It’s gonna’ be blue, it’s gonna’ have some smooth fade-in’s and it’s gonna’ be called “M4OS”. The “M4” stands for... Well, nothing.
- Our OS will consist of a loading screen, login screen and the main desktop.

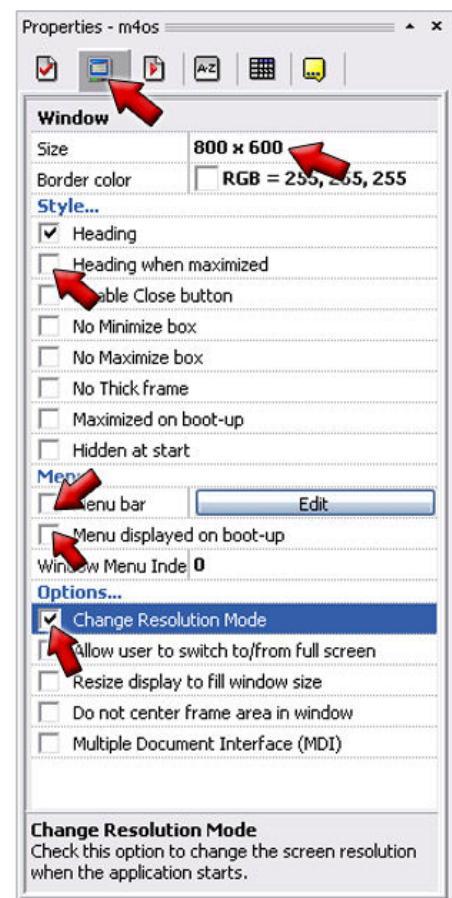
- There will be four icons on the main desktop – clicking on one of them will open the recycle bin, clicking on any other will return an error.
- All icons can be moved around the screen via the drag’n’drop method (by holding the right mouse button) – we’re also gonna’ add a funny “jiggle” effect, that’s going to shake our icons a bit, if – by any chance – any of them are overlapping.

📖 If you have any problems with this tutorial, or notice that there are some mistakes present, please, contact me and I’ll do my best to help you and replace all the errors with correct information.

Contact me at: marchewkow@gmail.com

Part I: Setting up the application.

Now, let’s get to work! Open Multimedia Fusion 2, **create a new application** and be sure to save it onto your hard drive (as I always mention: it’s a pretty good thing to have the “Autobackup” option turned on – check your “Preferences” window). Now, go to your application’s **Properties window** (if it didn’t open up by itself, right click on your application’s name in the workspace toolbar and choose “Properties” from the drop-down menu). Select the **Window** tab (second from the left, the one with the little computer screen). Set the window size to **800x600**. Higher resolutions would make our OS look a bit more realistic, but I’m pretending that the main game – the adventure game into which our system is incorporated – is made for 800x600. Now, make sure that the “*Heading when maximized*”, “*Menu bar*” and “*Menu displayed on boot-up*” options are **off**, while the “*Change Resolution Mode*” option is **on**. If you need any visual assistance, take a look at the screenshot to the right.



When that's done, it's time to create our frames. We're gonna' need seven frames in this project: one for the loading screen, one for the login screen, another one for the desktop, yet another for the error window, one for the recycle bin and two for e-mails that Jeremy will find waiting in the trash. Create them now, either by clicking on the next available frame number in the storyboard editor or choosing **Insert > New Frame** from the main menu. We're going to change the frame sizes of the last four frames later on – as for now, leave them all at 800x600.

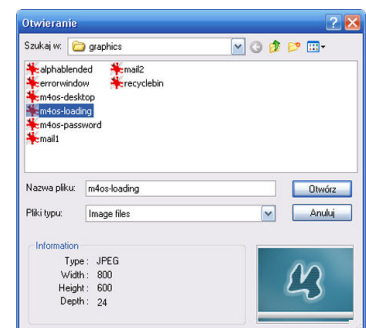
Do we really need as much as seven frames to complete this project? Nope. You could as well put it together with six, five, four, three, two, or even just a single frame. I'm using seven separate frames to make it a bit easier for all the MMF2 newcomers. Once you're done with this tutorial, feel free to explore the other possibilities, try to make all of this work exactly the same with a lower number of frames. And if you'll need any hints – don't hesitate to drop me a mail.

Part II: The loading screen.

Let's change the title of the first frame to "loading" (fastest way: right-click on that frame in the *Workspace toolbar* and select "Rename"). Once that's done – jump to the **Frame editor**.

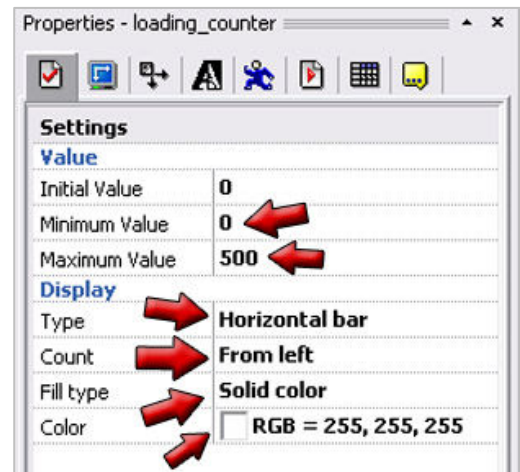
So... Here we are, gazing at a blank, white space, eight hundred pixels wide. What do we need to change this freshly created frame into a true (well... almost true) OS loading screen? Some kind of a background picture and a Counter object (acting as our loading bar). Oh, and we're also gonna' place two additional active objects without a real purpose, just for show. If it was one of my earlier tutorials, I'd just suggest to import them all from a previously prepared frame. But, heck, not this time! In this tutorial you have to create all the objects yourself.

Let's start with creating the aforementioned background picture. Create a new **Backdrop object**, double-click on it to open the image editor and then select the **Import** icon (📁), you can also press CTRL+O). Now, go to the folder into which you've unzipped the archive that contained this tutorial. Look for a directory named "**graphics**" and open it. Seek out the "**m4os-loading.jpg**" file, select it and click "OK".

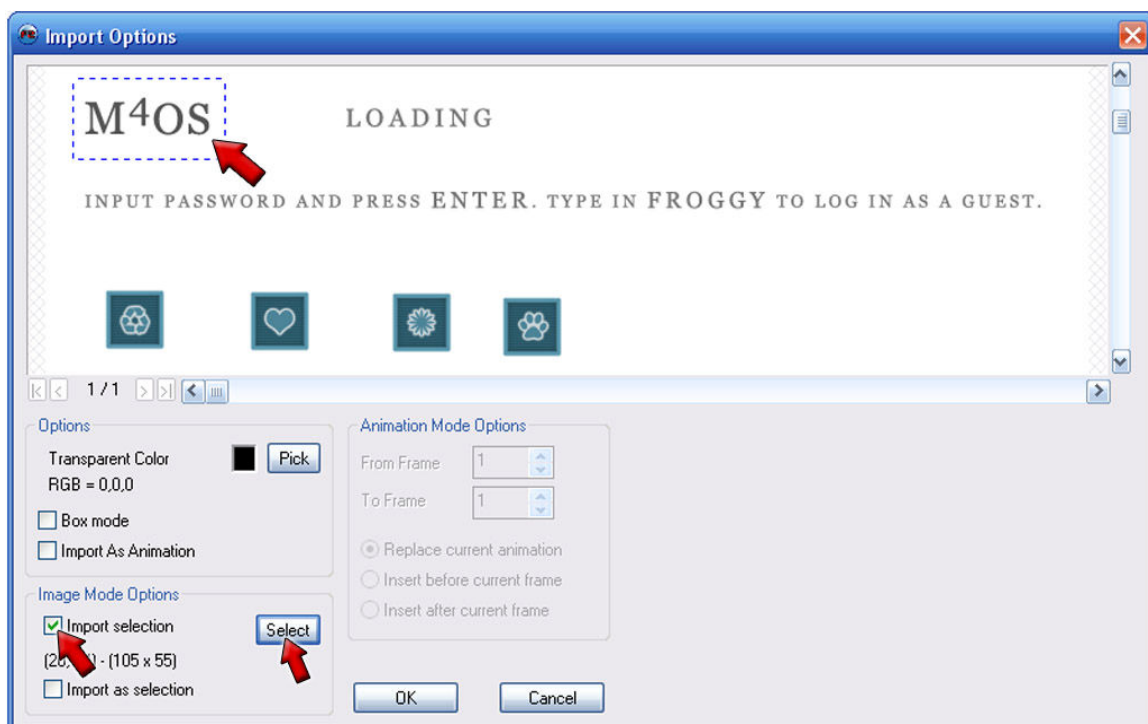


Click the "OK" button once more when MMF2 asks for import options – everything should be set correctly and you shouldn't encounter any problems. Once you're done – our background picture should be ready. Position it along the frame's borders and we're ready for the next step.

Now, let's form our loading bar – create a new **Counter** object, place it somewhere in the frame. Go to its *Properties window* double-check if the first tab to the left is open. Time to change the values a bit – set the *maximal value* to **500**, and the *minimal value* to **0**. Change the *display type* of the counter to **Horizontal bar**, with a **count beginning from left**, filled with solid color. Finally change the bar's color to **white**. If you need some assistance – check the screenshot to the right.



OK, now it's time to create those two additional actives that I've mentioned earlier. Create a new **Active** object. Click the import button and select the "**alphablended.png**" file this time. When the *Import Options* window appears, don't click "OK" yet! Select the "**Import selection**" option (don't get it confused with the "*Import as selection*" one) and then click on the "**Select**" button. Draw (by holding the left button) a rectangle around the "M4OS" little logo in the upper left part of the picture – you don't have to be too precise, just try to keep the whole logo inside your selection. Once that's done, click OK. Here, take a look at the screenshot below, if you're having any problems with keeping up:



Got it? Great! Let's move on to the image editor, then!

Firstly, either click on the cropping icon (✂️) or press Ctrl+K on your keyboard to slice away all the unneeded free space. Once that's done, notice that the "M4OS" logo has been imported onto a clean, transparent background – you didn't have to clean any white or gray pixels yourself. Also, notice that all the edges of our logo are pretty smooth – you won't find even a single ragged pixel there! Why's that?

Well, the PNG file from which we imported this little pack of pixels was an **alpha-blended image**. That means that, besides the picture, it contained an alpha mask – little bits of info telling your PC which of the pixels of our image should be semi-transparent and to what degree. The ability to use alpha-blended images in all backdrops and actives is a great new feature of MMF2 (one of the most important if you ask me) and I'm pretty sure that you'll find it useful someday too. As for now – just remember that alpha-blended images can look really great on any background and you can use them for both smooth-edged objects and stunning visual effects. Oh, and for a lot more things too, but we'll get to that much, much later, in another tutorial.

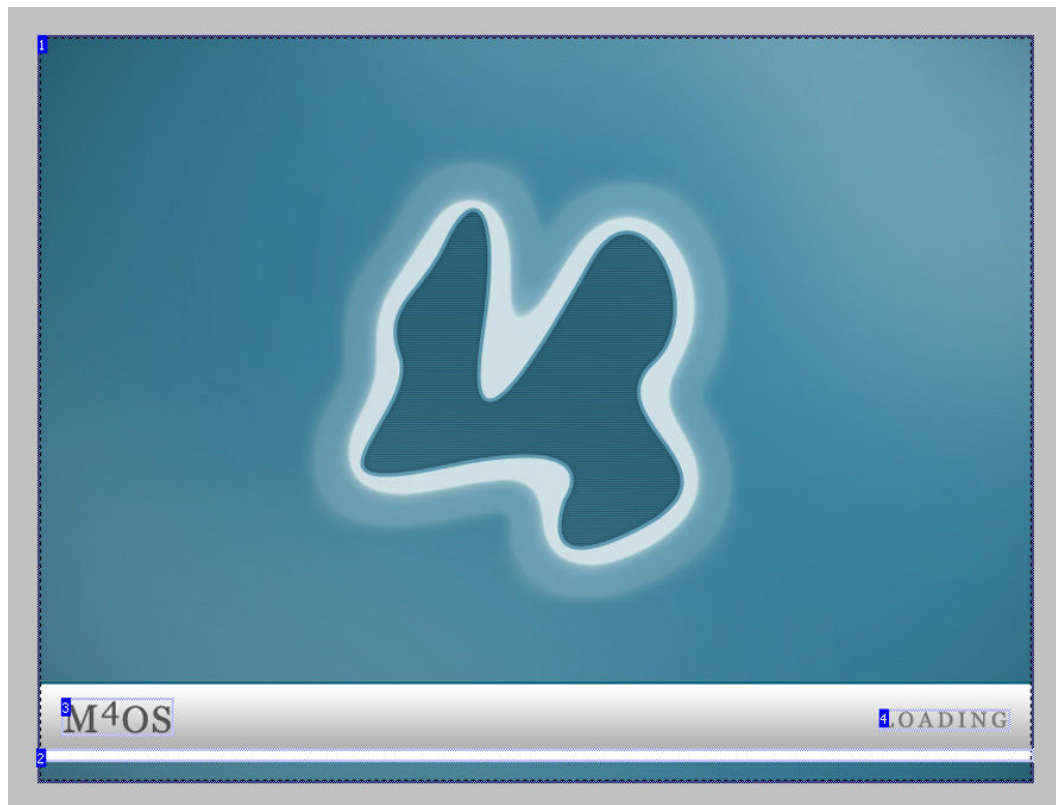
📄 **Please note: since some of the objects use alpha channels, a feature that is unavailable in Games Factory 2, TGF2 users should use basic library objects or create their own graphics instead.**

OK, when that's done, let's move on and create the second active – the "loading" sign. Do it in the same way as we created the "M4OS" logo, it's even in the same file – "**alphablended.png**". The only difference now is that you have to draw the selection rectangle around another part of the picture, bit to the right. Import the "loading" indication into a new active, click "OK", and we can move on.

Now, since we already have all the needed objects in our scene, let's finish all the remaining tasks here and wrap it all up. Select the "M4OS" active and rename it to "*little_logo*". Renaming objects with easily recognizable names is pretty important, since it can save you a lot of time. Now, go to the *Properties toolbar* and slide your view all the way to the "**Transitions**" section. Let's set up a nice, smooth Fade in transition, shall we? Open up the *Transition set-up* window (click on the "Edit" button, by the "Fade in" segment). Choose the "*Fade*" transition from the drop-down list and set it's duration to 4.74 seconds. Click "OK".

Do the same for the second active, changing it's name to "*little_loading*". But this time choose another Fade in transition, named **Bands**, and set it's duration to 3.73 seconds. I've also changed our Counter's name to "*loading_counter*" – so that I'd never forget why it's there.

Once you're done – it's time to place all these objects in correct positions. You'll also have to stretch the loading bar, so that it's as wide as our frame. Here, have a look at how I repositioned all of this... And feel free to modify this in any way in your own project:



Looks nice, doesn't it? Time to make it work!

Save your project and open the **Event Editor**. If you're new to my tutorials, let me introduce you to my event-recording system. If you know it already – just skip this big ol' frame below:

Koobare's MMF-to-paper coding system

IF (Condition): [Object for the condition] > Condition group > Condition

THEN (Action): [Object for the action] > Action group > Action

Seems pretty simple, right?

All the conditions are marked in red, while actions are written in fancy blue.

Object names are always put in [square brackets].

The final condition/action is always in *Italic*.

If we'll have a multi-condition event, then it'll be like this:

IF (Condition 1): [Object for condition 1] > Condition group 1 > Condition 1

IF (Condition 2): [Object for condition 2] > Condition group 2 > Condition 2

THEN (Action): [Object for the action] > Action group > Action

Whereas a multi-action event looks like this:

IF (Condition): [Object for condition] > Condition group > Condition

THEN (Action 1): [Object for the action 1] > Action group 1 > Action 1

THEN (Action 2): [Object for the action 2] > Action group 2 > Action 2

THEN (Action 3): [Object for the action 3] > Action group 3 > Action 3

If you'll have to input anything by keyboard, it will be indicated by coloring the text green and using < angle brackets >, like this:

< Set the Global Value A to 32 >

Additional comments, instructions and info will be put in << double angle brackets >>, using a different color:

<< Select any wave sound from the MMF2's sound library >>

From time to time I'll also use this style to throw in some extra tips about MMF2.

All you have to do is to go step-by-step through all the listed events and keep one eye on your Event Editor, and the second one on this tutorial... Not much philosophy in any of this.

Making it work...

The idea is pretty simple – every 4/100 of a second our application will add a random number (from 1 to 7) to the counter. When the counter gets to 500 – the app jumps to the next frame and we can move on.

Firstly, let's create this event:

```
IF: [The Timer Object] > Every
    << Set the timer to 4/100 of a second >>
THEN: [loading_counter] > Add to Counter
    < input: Random(6)+1 >
```

Notice that the “**Random(6)**” expression would give you a random number from 0 to 6. Adding “+1” to it's end ensures us that it'll return a value between 1 and 7. This is a thing worth remembering, since generating random numbers can be essential in a lot of projects.

Now, it's time for the second event:

```
IF: [loading_counter] > Compare the counter to a value
    << check if it's greater or equal than 500 >>
THEN: [Storyboard Controls] > Next frame
```

And that's it! Well... At least for the first frame. There's plenty more to do if we're to finish this OS imitation, so don't get lazy!

Part III: The login screen.

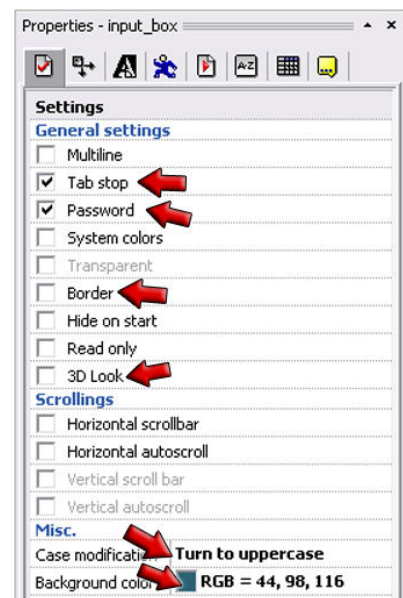
Let's move on to the second frame. We're gonna' need three objects here – one of them being the Edit object. Firstly, create a **Backdrop object**, which will act as our background image, and rename it to “login_backdrop”. Open it up, import an image from the “**m4os-password.jpg**” file and position the whole thing in the frame by dragging & dropping. Secondly, create a new **Active object**, rename it to “little_helper” and set it's Fade in animation to a **3.61 seconds Fade**. All we need to do now, is to open it up and import the line saying “*Input password and press enter. Type in FROGGY to log in as a guest*” from the “**alphablended.png**” file (it's right below the two graphical signs we imported earlier).

Got it? Great! Time to create the aforementioned Edit object. But... Wait, what exactly IS this Edit object I'm talking about? Here, let's take a look into the Multimedia Fusion 2's User Manual to help us out with this:

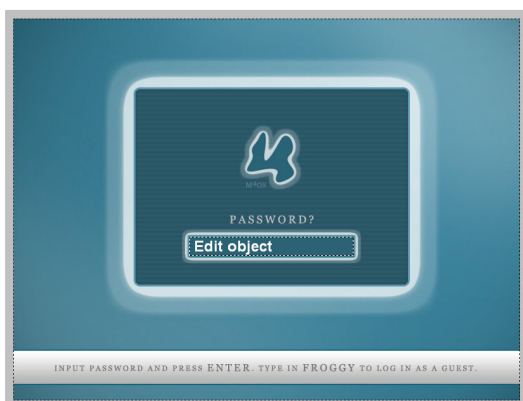
"The Edit object is a basic text editor that can load and save text files; cut, copy, and paste text; and be sized and positioned to suit your needs. You can also include scroll bars if you wish to limit the size of your Edit object, and borders to enhance its appearance. You can use Edit objects to create user entry zones and input fields, and you can define your Edit objects as read-only so that they only display text. The Edit object only handles plain text with no formatting".

Yup, you've heard the man: we can create **entry zones** and **input fields** with this object! Ain't that convenient? Sure it is.

Let's return to our work then. Create a new Edit object and rename it to "input_box". Now, let's take a look at its properties... Jeepers, there's quite a lot of them! Don't worry, though – we'll get it right in no time. Firstly, make sure that both the **Tab stop** (this will ensure that our Edit box will be selected if someone presses the TAB key) and **Password** (this will "code" all typed in letters, disguising them as dots) settings are set on. Secondly, turn **off** both the **Border** and **3D Look** boxes. Last thing: set the case modification to "**Turn to uppercase**" and try to set a background color that will be either the same or almost identical to the one that I've selected (RGB = 44, 98, 116).



When that's done, we can move on to the next tab – "**Text Options**" (third from the left). Here we can change our input field's font. Let's set it to **Arial, size 18, Bold, white**.



Got it? Great! That means we're almost done with yet another frame! Make sure that you reposition all the objects so that they look all great n' finicky. Oh, and let's not forget about resizing our "input_box" to the size suggested by the background image. You can check how I've positioned all of this by looking at the screenshot to the left. Looks nice, doesn't it?

Making it work... once again.

Once again, it's a pretty simple set of rules – and, once again, it's all playing out in just two events. The first one determines what happens if the player has entered a correct code (“FROGGY” – the password needed to log in as a guest) and then pressed ENTER. The second one determines what would happen if the ENTER key was punched while the given password was incorrect.

So, let's create the first event:

```
IF: [Keyboard & Mouse Object] > The Keyboard > Upon pressing a key
    << press ENTER on your keyboard >>
IF: [Special Object] > Compare two general values
    << check if expression Edittext$( "input_box" ) is equal "FROGGY" >>
THEN: [input_box] > Destroy
THEN: [Storyboard Controls] > Next frame
```

Yeah, I know that “*Edittext\$("input_box")*” looks scary and it won't be too easy to remember by hard... But, ya' know, you don't really have to remember it! How's that? Well, in MMF2 there's that sweet “*Retrieve data from an object*” button, which enables you to browse for the data you're interested in, retrieve it from specified objects and don't worry about any memorizing at all. All you'd have to do, is to click on the aforementioned button, right-click on the “input_box” object and select the “Get Text” command from the drop-down menu... And – voila! – you've got the same expression waiting for you!

You're most probably also wondering why the heck have I ordered our app to destroy the “input_box” object before jumping to the next frame, right? Well, we're gonna' add some frame fade-in's and fade-out's bit later on, and Edit objects don't usually go too well with them.

OK, let's cut the chit-chat and continue with our programming part... Here's the event played out when the player has entered an incorrect password:

```
IF: [Keyboard & Mouse Object] > The Keyboard > Upon pressing a key
    << press ENTER on your keyboard >>
IF: [Special Object] > Compare two general values
    << check if expression Edittext$( "input_box" ) is different than "FROGGY" >>
THEN: [Sound object] > Samples > Play sample
    << Select some kind of a short "error" sound from the MMF2's sound library >>
```

And that's it! Yup, we've just wrapped up frame number two...We're getting closer to the end with each minute! I guess that you'd just love to jump right into frame number three, right? Well then, guess again, private, since we have to go back to the Storyboard editor for a sec or two before that...

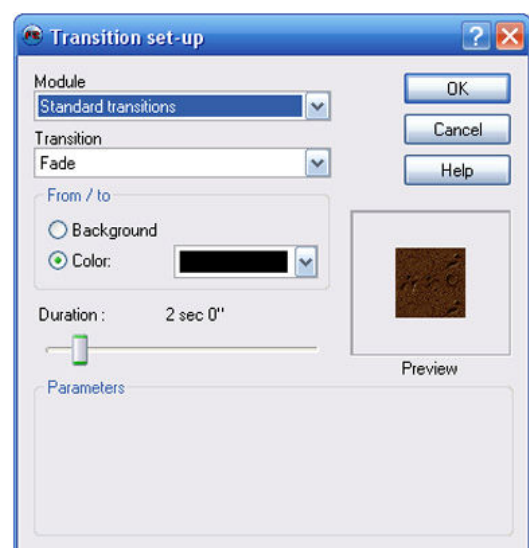
Part IV: Back to the storyboard...

Open the **Storyboard Editor**. There's not too much to accomplish here, I just wanna' add some Fade in and Fade out effects to a couple of frames, rename some of them and change the size of the last three. A true "piece of cake" situation, if you ask me.

Let's start with the easiest and least entertaining thing to do – renaming. As I said before – renaming things isn't as useless as it seems, if you'll ever wander off to the "big projects" area you'll instantly know what I'm talking about – searching through hundreds of frames without a real name can give you a real headache. Believe me – frame names like "Frame 51" and "Frame 61" will slow you down, making you gaze at the thumbnails in the Storyboard Editor, thinking "why the heck do they look so similar?!". Thus – renaming everything is a time-saver and a great habit.

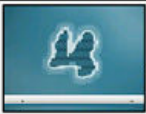


































OK, back to work. We have 7 frames in our storyboard. The first one is named "*loading*"... What about the rest? Let's rename the second one to "*login*", the third one to "*desktop*". Frame number four should be named "*error*", number five should be "*recyclebin*". Number six and seven? Change their titles to "*email1*" and "*email2*".

Got it? It's time for the Fade in's and Fade out's, then. Frames "*loading*", "*login*" and "*desktop*" should all have the **Fade** transition effect, with it's duration set to **2.0 seconds**. Oh, and we want all of them to fade to/from a solid color, **black** to be exact. To set the fades, just open the Transition set-up window, by either left- or right-clicking on the fade in/fade out buttons (🔍🔍, they are located to the left from the frame size in the Storyboard Editor's layout) and then choosing "Transition setup" from the drop-down menu.



Let's move on to the last three frames. Change the size of the frame named "error" to **400x250** pixels, then change frames "recyclebin", "email1" and "email2" to **465x310**. Once that's done, set the fade in transition effect for "email1" and "email2" to **Zoom**, from background, with the duration of **1.1 seconds**.

Got it? Great! Here, have a look at how my Storyboard Editor looks like at this point of the tutorial... If you've followed me closely enough, yours should look the same:

No.	Thumbnail	Comments
1		Fade
		Title : loading Password :
		    800 by 600
2		Fade
		Title : login Password :
		    800 by 600
3		Fade
		Title : desktop Password :
		    800 by 600
4		Fade
		Title : error Password :
		    400 by 250
5		Fade
		Title : recyclebin Password :
		    465 by 310
6		Zoom
		Title : email1 Password :
		    465 by 310
7		Zoom
		Title : email2 Password :
		    465 by 310
8	More...	

As you can see – we already have something there, but there's still quite a lot to do. To sum it up a bit: we need to create all the popping out windows (the error window, the recycle bin and two e-mails) and the main desktop, with all it's drag'n'droppable icons and stuff like that. We're gonna' leave the desktop for later, and move on to frames 4 to 7 right now.

Part V: Errors, recycle bins and e-mails.

It's all about the greater good.

This one is gonna' be real easy. Open the “error” frame. Create a new **Backdrop** object, rename it to “error_backdrop”. Open it up, import an image from the “**errorwindow.jpg**” file and position the object by dragging & dropping. Got it? Well, that means you've just finished this frame. Wasn't that quick? Sure it was.

To be totally frank with you – we don't even really need a frame for this. This error window could be easily created by using an Active object inside the “desktop” frame. So why use a whole frame for this? Just for learning purposes – it'll give us an occasion to play with the Sub-Application object a bit more. Yup, it's all about the “greater good”. And if you've seen Edgar Wright's “Hot Fuzz” you surely know that the “greater good” is an important thing, right?



The recycle bin.

Now, moving on to the next frame – time to set up our own recycle bin... Once again, create a new Backdrop object and import an image into it – this time from the “**recyclebin.jpg**” file. Once that's done, open up the **Event editor**. We'll need just two events here (what's with that “just two events” already? Geez, everywhere I go, all I need are just two events...):

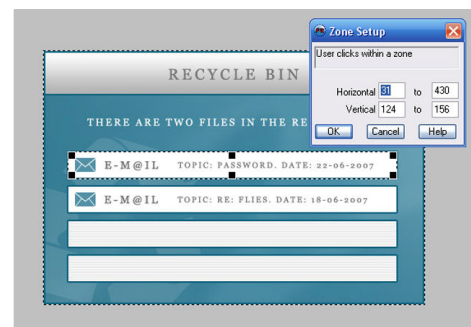
IF: [Mouse & Keyboard Object] > The Mouse > User clicks within a zone

<< Left button, single click >>

<< Select a zone around the first white button from the top – the one with the “password” e-mail >>

THEN: [Storyboard Controls] > Next frame

If you have any problems with understanding which “button” do I mean – take a look at the little screenshot to the right. My zone's size & position was “(31,124) to (430, 156)”, yours may vary a bit.



And here's the second event:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks within a zone
<< Left button, single click >>
<< Select a zone around the second white button from the top – the one with the “RE: flies” e-mails >>
THEN: [Storyboard Controls] > Jump to frame
<< Click on the “Use Calculation” button >>
< input: 7 >

Done? Time to move on to the “email1” frame.

The first e-mail.

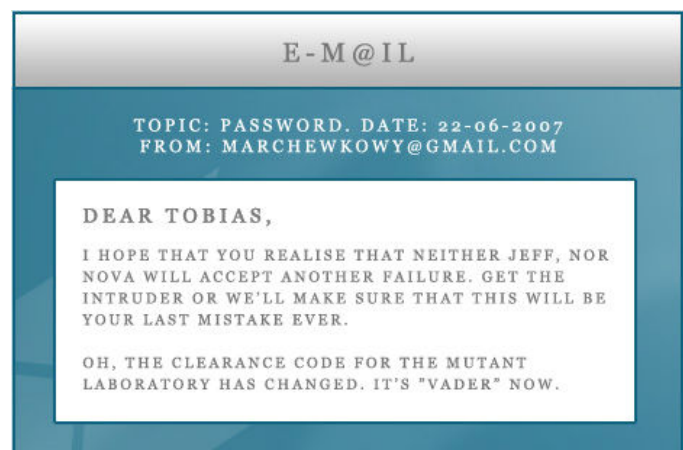
You know the drill – create a new Backdrop object, import an image from the “mail1.jpg” file, then go the Event editor. We want this e-mail to close and return to the recycle bin if the player clicks with the left mouse button:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Left button, single click >>
THEN: [Storyboard Controls] > Previous frame

And here's the same for the right button...

IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Right button, single click >>
THEN: [Storyboard Controls] > Previous frame

Don't be too troubled if the whole “previous frame”, “next frame” business becomes confusing, it can be a bit tricky to follow. It'll soon become pretty clear to you, though, once we move on to the Sub-Applications... You'll see that all of this has a lot of sense when introduced in due time. Now, let's finish up the frame for the second e-mail – pretty much the same as this one.



The second e-mail.

Import the “mail2.jpg” file into another Backdrop object. “The flies have mutated into something weird, I’m not totally sure if this is what we wanted” – it says. Well, tough luck, happens if you experiment with gamma rays and mutation... Don’t do that at home, kids! Never know when you’re gonna’ transform into Hulk or – and this would really suck – a flying banana. Anyways, let’s jump to the Event editor and create these two little thingies:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Left button, single click >>
THEN: [Storyboard Controls] > Jump to frame
<< Click on the “Use Calculation” button >>
< input: 5 >

And here’s the second one...

IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Right button, single click >>
THEN: [Storyboard Controls] > Jump to frame
<< Click on the “Use Calculation” button >>
< input: 5 >

Pretty easy, right? Oh, too easy, even? Not exactly the thrill you’ve been looking for? No sweat, we still have to create the whole “desktop” frame, remember? And this one’s gonna’ be a bit tougher to beat.

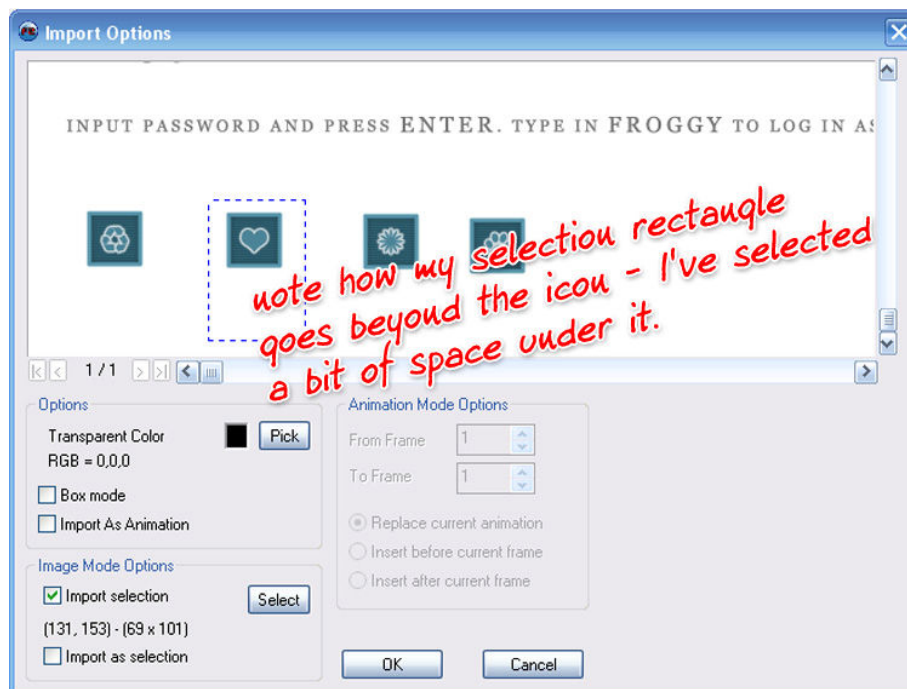
Scenario idea:

Jeremy got to Tobias’ workroom – in which the computer with the password was located – by a ventilation shaft. When he was jumping out of it, he accidentally tripped on something and heard a massive “bdong!” sound. Once he puts the lights on, he realizes what he has done – he crashed the whole computer, smashing it to pieces! Now the player has to gather up all the misplaced elements, look for some spare parts in one of the nearby cabins, and then join them together to make the workstation work again, allowing him to retrieve the password.

Part VI: The main desktop.

Open up the “desktop” frame, and let’s start importing! Firstly, as usual, we’ll need a nice good ol’ **Backdrop object** to cover the whole frame. Create one, rename it to “main_backdrop” and import “m4os-desktop.jpg” into it. Once that’s done, let’s create four **Active objects** and transform them into our icons.

Let’s start with the first one – create an Active object and rename it to “icon_files”. Load an image into it – choose the “**alphablended.png**” file, then draw a selection rectangle around the second icon from the left (the one with the heart symbol), making sure that you select not only the icon, but also a bit of the white space under it. Like this:



So, why the heck are we selecting so much blank space at the bottom of the icon...? ‘Cause, dear Watson, it isn’t actually blank! All the icons have their names beneath them (in this case: “John’s files”) – we just can’t see them yet (that’s because I’ve used a white font, while MMF2 shows PNG transparency as white). Just click “OK”, then crop the unused space (Ctrl+K) and you’ll see what I mean.

Once you’re done with the first icon, position it somewhere in the frame (I’ve placed it at x=57, y=319) and get to the next one... Create a new active, change it’s name to “icon_bin”, load the first icon to the left from the “alphablended.png” file (the one with the recycle symbol), drop it somewhere and make sure that it does not overlap the “icon_files” object.

Next on the list is “icon_drives” object – with the third icon from the left (the one that looks like a flower), followed by “icon_programs”, an icon with a dog’s paw print on it. After that’s done – we’ll have to add all these objects to a shared group – **assign a shared qualifier** – so go to the *Properties toolbar*, open up the *Events* tab (it’s the second one from right), and add all our icon objects to the “**Bonus**” qualifier group.

Once you’re done with the icons, it’s time for...

...Sub-Applications.

OK, so what are these thingies exactly? Sub-Application objects enable you to display another frame or app in your main application, giving you enormous power over what happens in your game and how it all works out. You can, for example, easily create a drag’n’droppable inventory that is displayed inside your game, covering – let’s say – the left side of the screen, while you can still play on the right side – similar to the inventory window in Blizzard’s “Diablo”. As it is said in Multimedia Fusion’s 2 manual:

“The Sub-application object allows you to insert a Multimedia Fusion 2 application (.MFA or .CCN) into a frame in another Multimedia Fusion 2 application. It can be also a frame of the same application. The inserted application can function independently of the main application or can share data and be controlled by the main application.”

Seems easy, right? Let’s see it in action, then. Create a new **Sub-Application object** and place it in your frame. Go to it’s Properties toolbar, change the “source” to “**frame from this**



application” and select **frame number 4**. Now, rename this object to “error_subapp” and place it outside of your frame (I’ve placed it at these coordinates: x=150, y=-260). As I said before – we didn’t actually need a Sub-Application for the error window, since it would perfectly work even if we used a just single active for this – but I’ve decided to use it nonetheless, just for the sake of learning & practice (and remember: “practice makes perfect”... or something like that).

Now, create another Sub-Application object and rename it to “bin_subapp”. Once again we’ll have to move it out of our frame (I’ve placed it at x=560, y=-320, but it doesn’t matter where

you put it, as long as it's outside of our workframe). Change the "source" option of this object to to **"frame from this application"** and select **frame number 5**. Note that you can both use frames from your main app and other applications when playing with a Sub-app – this may come in handy someday, when you'll be working on your own big project. It's also worth to keep in mind that you can use as many Sub-apps as you'd like, it's not just "one per frame" – as you can see, we're using two in this tutorial – which means that Sub-apps can be really handy when designing your user interface.

Now, once we're done with all that, it's time to move on to the Event editor and make all the gears work together.

Time for some coding!

At last! Finally there's gonna' be a little more action here, not just the tedious "two events to make it work", that's been haunting me since last weekend. Anyways, let's get to it, soldier! Create all the events, one after another, and we'll have our M4OS working in no time!

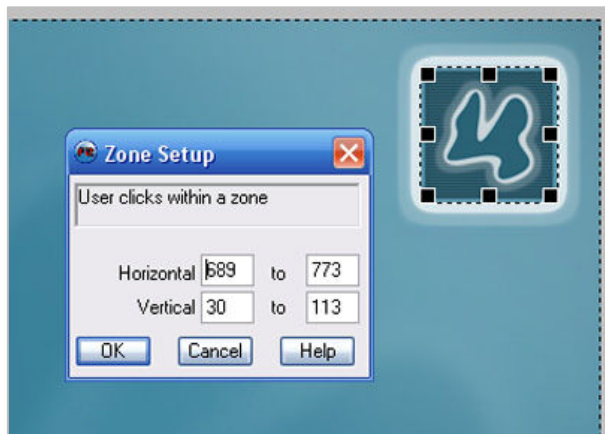
1) Let's start with an event that will become a basis for our simple (yet, a bit imperfect, to be honest) drag'n'drop system. This is gonna' be a really undemanding thingy, you could achieve a better effect by using Alterable values and changing them from 0 to 1 when player starts to drag a single icon... But this would be a bit more complicated system, demanding more time and explanations. Anyways, let's do this the simple way:

```
IF: [Mouse & Keyboard Object] > The Mouse > Check for mouse pointer over an object
<< Select the "Group.Bonus" qualifier >>
IF: [Mouse & Keyboard Object] > The Mouse > Repeat while mouse-key is pressed
<< Right mouse button >>
THEN: [Group.Bonus] > Position > Set X coordinate
< input: XMouse >
THEN: [Group.Bonus] > Position > Set Y coordinate
< input: YMouse >
THEN: [Group.Bonus] > Visibility > Change ink effect
<< Choose: Monochrome >>
```

It's a good thing to memorize the "XMouse" and "YMouse" expressions – they're both pretty simple, yet very useful, giving you the possibility to check and compare the current X and Y positions of the mouse pointer. Also, note that all icons change their ink effect to *monochrome* while they are being dragged.

2) Here's another event that will help us with our drag'n'drop system – this one is checked true and executed when the player does not hold the right mouse button. If you're not sure how to negate an event – right-click on it and choose “negate” from the drop-down menu:

IF: (NEGATE) [Mouse & Keyboard Object] > The Mouse > Repeat while mouse-key is pressed
<< Right mouse button >>
THEN: [Group.Bonus] > Visibility > Change ink effect
<< Choose: None >>



3) Here comes number three: if our player clicks on the big “M4OS” logo located in the upper-right side of the screen, all four icons are automatically arranged in a row:

IF: [Mouse & Keyboard Object]
> The Mouse > User clicks within a zone
<< Left mouse button, single click >>
<< Select a zone around the “M4OS” logo –
See the screenshot to the left >>

THEN: [icon_files] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=253, y=57 >>
THEN: [icon_bin] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=353, y=57 >>
THEN: [icon_drives] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=153, y=57 >>
THEN: [icon_programs] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=53, y=52 >>

4) Our fourth event will add a small “jiggle” effect to this whole thing. Our icons will shake a bit if – by any chance – any of them are overlapping. This is a lot simpler than it sounds:

IF: (NEGATE) [Mouse & Keyboard Object] > The Mouse > Repeat while mouse-key is pressed
<< Right mouse button >>
IF: [Group.Bonus] > Collisions > Overlapping another object

```

<< Choose: Group.Bonus >>
THEN: [Group.Bonus] > Position > Set X coordinate
< input: X( "Group.Bonus" )+Random(25)-Random(25) >
THEN: [Group.Bonus] > Position > Set Y coordinate
< input: Y( "Group.Bonus" )+Random(25)-Random(25) >

```

5) Here's a small appendix to the previous event – if one of our icons is out of the play area (if it "jiggled out" to far away), it is sent to the middle of the screen:

```

IF: [Group.Bonus] > Position > Test position of "Group.Bonus"
  << Select "Is the object outside?" – the button in the lower-left side of the window >>
THEN: [Group.Bonus] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=396, y=280 >>

```

And that's it for our drag'n'drop system! Time to get those Sub-apps working...

6) Now, let's create the event I usually start with – the traditional "Start of frame" event, which will reposition our Sub-apps in the middle of the display and make them hidden. We could as well get our Sub-apps to their positions in the Frame editor, but I've decided not to, just to show you two things... That: a) Sub-apps can be easily repositioned on runtime, which makes them even more practical; b) You can move them out of your frame in the Frame editor and then easily get them back in the middle of the screen with just a single event, which makes it easier to edit levels and operate on objects that would be otherwise not accessible, hidden behind your menu / error / inventory etc. windows.



















```

IF: [Storyboard Controls] > Start of frame
THEN: [error_subapp] > Visibility > Hide
THEN: [bin_subapp] > Visibility > Hide
THEN: [error_subapp] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=188, y=163 >>
THEN: [bin_subapp] > Position > Set Position
<< At actual X, Y coordinates >>
<< Set coordinates: x=156, y=125 >>

```

Once your done, let's make this event the first one on the list – just grab it's "number box" and drag it upwards, from the sixth position to numero uno.

Here's what we should have by now (note that it doesn't have to be *identical*...):

All the events All the objects															
1	• Start of Frame													✓	✓
2	• Mouse pointer is over  + Repeat while right mouse-key is pressed												✓		
3	• ✗ Repeat while right mouse-key is pressed												✓		
4	• User clicks with left button within zone (688,29) to (773,114)							✓	✓	✓	✓				
5	• ✗ Repeat while right mouse-key is pressed +  is overlapping 												✓		
6	•  is out of the play area												✓		

7) Time for the lucky number seven: this event will control what will happen if our player clicks on the “icon_files” object – the “error_subapp” object will be displayed, and the “bin_subapp” object will be hidden:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks on an object

<< Left mouse button, single click >>

<< Choose: *icon_files* >>

THEN: [error_subapp] > Visibility > Show

THEN: [bin_subapp] > Visibility > Hide

8) Here we have the same two actions, the only difference is that this happens when “icon_drives” is clicked, not “icon_files”:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks on an object

<< Left mouse button, single click >>

<< Choose: *icon_drives* >>

THEN: [error_subapp] > Visibility > Show

THEN: [bin_subapp] > Visibility > Hide

9) ...And here goes the same for “icon_programs”:

IF: [Mouse & Keyboard Object] > The Mouse > User clicks on an object

<< Left mouse button, single click >>

<< Choose: *icon_programs* >>

THEN: [error_subapp] > Visibility > Show

THEN: [bin_subapp] > Visibility > Hide

10) Here's something similar, but a tad different – we've got a quick switcheroo when it comes to actions, since this time it's the "error_subapp" that plays hide & seek:

```
IF: [Mouse & Keyboard Object] > The Mouse > User clicks on an object
<< Left mouse button, single click >>
<< Choose: icon_bin >>
THEN: [error_subapp] > Visibility > Hide
THEN: [bin_subapp] > Visibility > Show
```

11) And what will happen if our player clicks somewhere in the frame when either the error window or the recycle bin is visible? Yup, they just disappear! Mysterious, huh? Create these two events, one after another:

```
IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Left mouse button, single click >>
THEN: [error_subapp] > Visibility > Hide
THEN: [bin_subapp] > Visibility > Hide
```

```
IF: [Mouse & Keyboard Object] > The Mouse > User clicks
<< Right mouse button, single click >>
THEN: [error_subapp] > Visibility > Hide
THEN: [bin_subapp] > Visibility > Hide
```

12) We're almost there, private, so keep it steady and stay focused! Oh, and guess what happens when you press Escape on your keyboard...?

```
IF: [Mouse & Keyboard Object] > The Keyboard > Upon pressing a key
<< Press: ESCAPE >>
THEN: [error_subapp] > Visibility > Hide
THEN: [bin_subapp] > Visibility > Hide
THEN: [bin_subapp] > Jump to frame
<< Click on the "Use Calculation" button >>
< input: 4 >
```

13) And here's the last event – just a helper, so that the final fade out can go perfectly...

```
IF: [Storyboard Controls] > End of application
THEN: [error_subapp] > Destroy
THEN: [bin_subapp] > Destroy
```

Oh, I've almost forgot... Here's the final shot of my Event editor, just for comparison:

Thanks for your time and see you again soon!

Koobare
marchewkowy@gmail.com

Page 25/26

You have been reading...

M4OS

a Multimedia fusion 2 tutorial



Brought to you by

KOOBARE

ClickTeam's funkiest

~~mercenary~~

mutant factory worker

Created for Multimedia Fusion 2 & Multimedia Fusion 2: Developer

Always be sure to have your MMF2 up-to-date!