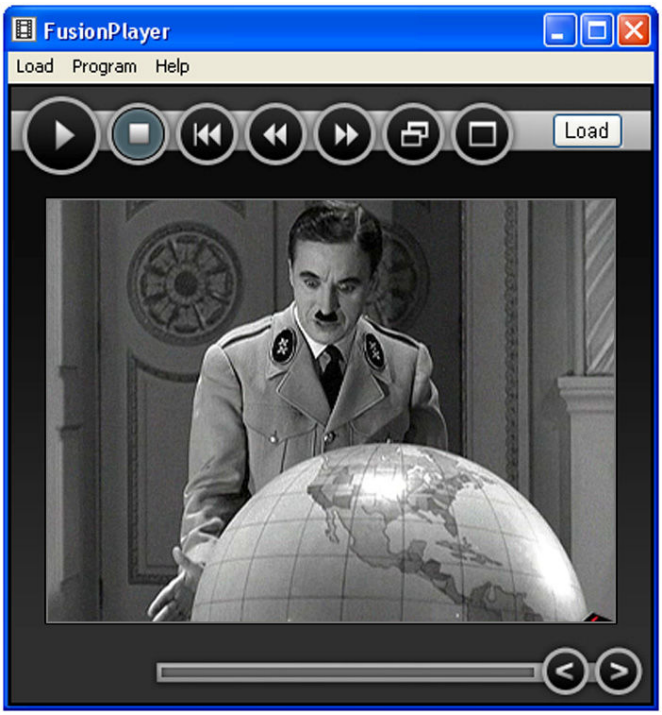
the FUSION PLAYER tutorial



**Sceenshot: Charlie Chaplin's classic: "The Dictator"*

You may not use this tutorial for any other purpose than learning, working and having fun... In other words: You can use this 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 Koobare's third little tutorial, this time focusing on how to create the "Fusion Player" - a simple, yet fully functional, video player application - using the best multimedia authoring tool ever: Multimedia Fusion 2 by Clickteam! The main purpose of this tutorial is to teach you how to work with global strings & values, global events, custom menus, groups, comments and objects such as the DirectShow object, Common Dialog object and the Window Control, to establish a well-coded multimedia player. That's a lot of interesting features, isn't it?

Remember that it's always good to have some sort of a knowledge basis – if you feel like a total newbie that doesn't even know how to create events in the Event editor, perhaps you should start with my previous tutorials (the "SmellyClaw" and "Glob Wars" tutorials), or the "Catch the fruit" tutorial by Andos. If you feel OK about discovering more advanced features at the start – don't let me discourage you, most probably you'll do perfectly fine and you'll have your own "Fusion Player" app in no time.

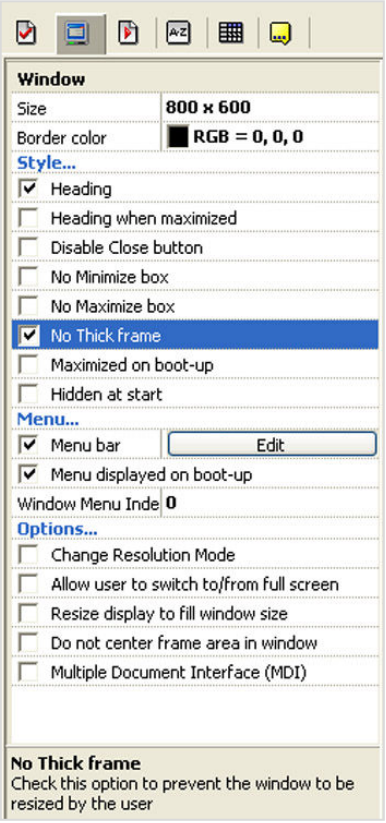
Here are some basic features of the application we're going to create:

- "Fusion Player" will be a fully functional video player that loads and plays video files with .AVI, .MPG and .MPEG extensions.
- All videos will be shown in the TV-standard: about 4:3 ratio.
- Three objects will be crucial for this project: the DirectShow object, the Window Control object and the Common Dialog object by 3EE. You'll learn more about them and their usage later on.
- We will create a custom application menu and use Global Events to control it over frames.
- We will use Global Strings and Global Values to keep track of the position and path directory of the currently played movie.
- Our "Fusion Player" will display videos in three different modes: standard mode (using a 400x385 application window), windowed mode (displaying 640x480 video size in a 800x600 screen mode) and full-screen mode (using 800x600 full-screen resolution).
- While in the standard mode, users will have the possibility to rewind and forward the playing movie, or even skip it's larger parts – by 1/6 of it's duration.
- A simple horizontal bar counter will be used as a "progress bar", showing how much of the movie has been already played.

Please note that you must have the first **Bonus Pack** installed in order to use the Common Dialog object by 3EE! If you haven't downloaded it yet, just visit www.clickteam.com, or get it from here: ***http://www.clickteam.com/eng/downloadcenter.php?i=176***

Part I: Setting up the application

It's time for some fusioning! Open Multimedia Fusion 2, **create a new application** and save it onto your hard drive (hint: turning the "autobackup" option ON in MMF2's preferences and – additionally - saving some backups on your own can really save you some time and effort in case of a system failure). Go to the application properties screen (if it didn't open by itself, right click on your application's name and select "*Properties*" from the drop down menu).



"Window" properties tab

Firstly, make sure that the **graphic mode** is set to 16 million colors (in the "Settings" tab). Then, go to the "Window" options tab (take a look at the screenshot to the right if you need some visual guidance here – it's pretty straightforward, but a little visual help never hurt anyone) and set the window's size to **800x600**. Woah! – some of you certainly shouted – wasn't the "standard mode" supposed to use a 400x385 window? Well - thanks for bringing that up – it surely was. But we're going to set this to 800x600 anyway. Why, shall you ask? Take a look at our "features list" once again. You can find two additional modes in our little app, beside the standard one. They both use a 800x600 screen resolution. And believe me – it will be a lot easier for us to set the window to a higher size in the preferences, and then just size it down a bit with the Window Control object, than doing it the other way around. It is possible, of course, but it can be a bit harder. And I don't know about you, but I'm rather a person that doesn't have to necessarily go down the harder path – I usually pick the more efficient one. Trust me – set the window's size to 800x600 and everything will be just fine.

Don't leave the "Window" tab yet, partner! There's still plenty to do here. Set the "**border color**" to black. It's purely aesthetical, but "aesthetic" is a big word in my dictionary, y'know? Uncheck the "**Heading when maximized**" option, but leave the "Heading" one on. Check the "**No thick frame**" on. Ready? Great! It's time to create our own **custom menu bar**…

Menu bars are cool. Utterly cool. You can do almost everything with them – from restarting your app to saving a game. Bah! If you'll bear in mind that you can use the "Has an option been selected?" condition in the Event editor, you'll receive a powerful new toy to play with. Just imagine: you can even create a custom movement that's based solely on the menu bar's buttons! Well… If you know how to create a custom menu bar, that is.

If you don't – don't worry. I'll teach you that in seconds, it's really quite easy.

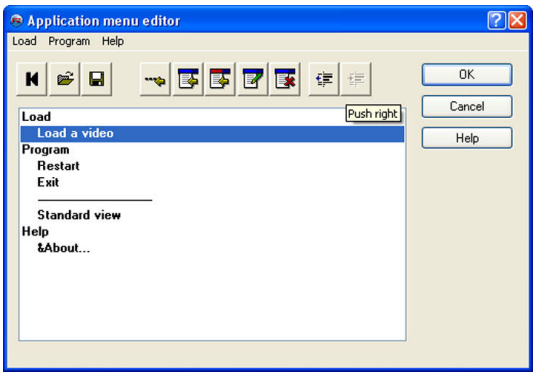
You're still at the "Window" preferences tab, are you? Great. Can you see that Edit button right by the checked "Menu bar" option? Click on it. A new dialog window will appear. Take a look at it – it's not too complicated, right?

You can save a menu bar, load a menu bar or reset all menu bar settings with those three buttons to the left (look at the screenshot to the right). If you'll look to the right, you'll find some even more interesting buttons, enabling you to insert a separator, insert an item, insert an item from the default menu (such as "Password", "Players", "Quit" etc.), edit the currently selected item, delete an item or "push" it either to the left, or to the right (creating sub-menus). Now… Enough wandering around with your cursor. Time for some action!

Delete all the positions in your menu. Then, input new ones – using the "insert an item" button - one after another, in this order (use the texts in these brackets as texts of your menu options):

["Load"]
["Load a video"]
["Program"]
["Restart"]
["Exit"]
[Insert a separator here]
["Standard view"]
["Help"]
[Insert an item from the default menu – select the *Help* > *About* option]

Got it? Well that was pretty easy, wasn't it? The only problem is that our menu looks a bit awkward right now. You can take a look at it at the top of this dialog window – notice how all the options are positioned. Time to fix that. Select the "Load a video" option, and then press the "Push right" button. Now, do the same for the "Restart", "Exit", "Standard view" and "&About" options, and then for the separator too. You should have something like this by now:



["Load"]

["Load a video"]

["Program"]

["Restart"]

["Exit"]

[--- separator ---]

["Standard view"]

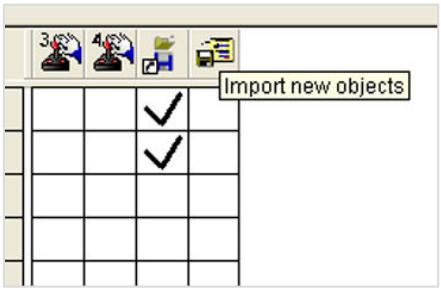
["Help"]

[&About]

That really seems to work, doesn't it? Well, perhaps this menu bar looks fine and steady, but beside the "About" button (which is a button selected from the standard menu options) it really does not work at all. We'll make it work soon enough, though. And we'll use another one of MMF2's powerful features to do that. We'll use Global Events!

Part II: Time for some theory - Introduction to Global Events

What are those mysterious Global Events, you ask? Well, they're not that mysterious really. Global Events are basically the same as normal Events, with the addition that they work… Globally. In other words: if you'll create a normal event in a single frame – it will be triggered only in that specified frame. But if you'll use the same conditions and actions to create a Global Event – it will be triggered in every frame, no matter which level is currently running, where has the player wandered off to, etc. Just think of it for a moment: you're given a tool that enables you to change the WHOLE game with a few single clicks, just by adding a few Global Events here and there – no matter how many levels have you built already, no matter how many hundreds of frames have you produced – you can easily add an event that – for example - restarts the game if one of the Global Values is lower than 0. Can you smell it? *sniff*, *sniff*. Yes, my Sith apprentice, that is the smell of power!



Now, let's put aside our lightsabers (Losties are reminded to stop pushing that darn button already!) and return to our application's properties screen, shall we? Select the "Events" options tab (second from the right) and click on the Edit button. The Global Event editor will open, very similar to the standard Event editor. Notice that you can find a new object here, the one most to

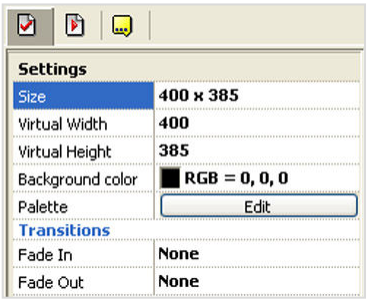
the right. It's called "Import new objects" and used for… Du-uh! Importing new objects! Well, not exactly "new"... And not exactly "importing" either. :P It's main purpose is to create SHORTCUTS to objects that exist already in your game. Such shortcuts will enable you to control the specified object from the Global Event editor's level. Difficult to understand? Well, maybe this example will shed a little more light…

Imagine that you want to have all the data from a specific array saved every time player presses the "X" key, no matter which frame he is currently in. How to do this? Well, for once, you could add a single event in all of your frames, one by one. But what if you'll suddenly decide to replace the "X" key in the event's condition with a "T" key? Well, then you're going to have one heck of a night, replacing a single event in each and every single frame you've produced by now. Not the best way to spend time, actually. I guess that you'd be happier to spend the evening with your beloved one, or at least watch a good zombie-flick, instead of messing with that single line of fusion-code, wouldn't you? Sure you would. That's why we have Global Events. They can save you a lot of time, believe me, and they're pretty good at speeding up the app-creation process too. All you have to do, to solve our little array-saving riddle, is to create a shortcut for the previously mentioned array in the Global Event editor, and then just add a single event there: when "X" key is pressed, then: save array to a file. And that's all! It doesn't really matter if the player is crossing the flaming river in frame 11 or fighting a blood thirsty dragon in frame 47 - he will be able to save all game data stored in the array by simply pressing the "X" key.

Easy, isn't it? We're going to use this knowledge a bit later on, in the coding section of this tutorial. Well, I guess that we could even script all those Global Events now, but let's just set up the frame and import all the needed objects firstly.

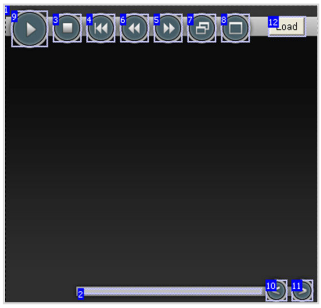
Move on!

Part III: Setting up the frames. Importing, creating and setting up objects

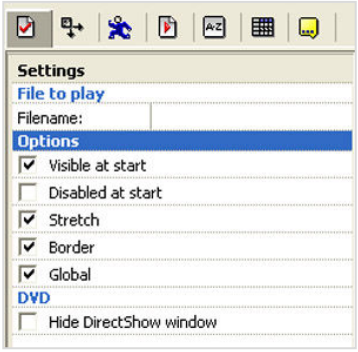


We'll need three frames for our application (create them in your Storyboard Editor, by clicking on the number right by the "More…" text). Set the size of the first frame to **400x385** (see screenshot to the left)**.** The second one should be set to **640x480**, and the third one – to **800x600**. Set the "**background color**" of all three frames to **black** (you can find the right option in the frame properties screen, "Settings" tab).

It's time to import (or create) all the needed objects. Firstly, make sure that you have the "snap to grid" option turned off (check the "view" menu and search for such a button: Snap to Grid). Then, find the "**playerlibrary.mfa**" file, which was packed into the same archive as this .PDF tutorial, and open it. Select all the objects (hint: press CTRL+A), copy them into the first frame of your application (Note: some of the objects use **alpha channels**, a feature that is unavailable for TGF2 users. TGF2 users should use basic library objects or create their own graphics instead) and place them in a way, that will make them perfectly fit the frame (if you remembered to check the "snap to grid" option off, there shouldn't be any problems with this). Notice that all the controller buttons are made as active objects – only the "load" button is really a button object. Why? No particular reason – I'm just used to using actives as buttons, since that gives me a bit (but only a bit) more control over them. It's up to you whether to use the button object or actives in your future applications.



Having all the objects imported from our library (your frame should now look in a similar way to the screenshot on the right), you've probably noticed that something is missing here. Yep, you're right, there's neither a DirectShow object, Window Control object or the Common Dialog object, although I've mentioned them earlier. Let's create them, shall we?

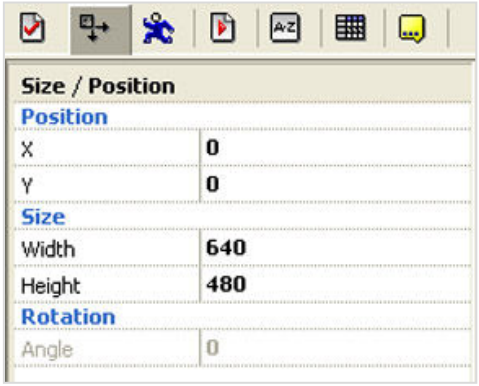


The DirectShow object. Double-click with your left mouse button on an unused space in or out of your frame in the Frame editor (or you can just select the "New object" command from the "Insert" menu) and select the DirectShow object. Notice that it will ask you to specify a starting video as soon as it is created – just click "cancel". Enter object's properties. In the first tab (take a look at the screenshot to the right if you require some visual guidance), check the "**stretch**" and "**global**" options (and leave the "**visible at start**" and "**border**" options ON), in the second one – set the object's width to 358, and it's height to 267. Then, in the same tab, set it's X position to 21, and Y position to 69. Ok, we've got the DirectShow object set up! Time for the next thing on our list…

The Window Control object. Create the Window Control object, move it above your playframe and… That's all. :) Nothing more to set up here, we'll control this object with the Event editor only.

The Common Dialog object. Create the Common Dialog object. A dialog window will appear, asking you to input starting preferences. Just click "OK". If you wish, you can play a bit with those preferences, setting the initial directory to "C:\" or changing the dialog title – but that's really not necessary.

It's time to make sure that the other two frames are ready as well. Firstly, select all three newly created objects (by clicking on them while holding the "SHIFT" button) and press "CTRL+C" (or select the "copy" command from the "Edit" menu), to copy them to your clipboard. Then, when that's ready, go to the second frame, the 640x480 one. Press "CTRL+V" (or select the "paste" command from the "Edit" menu) and click somewhere in the frame to copy all the three crucial objects into this frame. Now, select the DirectShow object and enter it's preferences. In the "Size / Position" tab (second from the left – you can find a screenshot to the right) set the object's X and Y positions both to 0, it's width to 640 and it's height to 480. It should cover up the whole frame. Now, save your work and enter the next frame (the 800x600 one). Do the same here: paste the previously copied objects, enter DirectShow object's preferences, set the position to (0,0) and the size of the object to 800x600, so that it can perfectly fit the frame.



Note that there are alternate methods of achieving similar results – by using the event-driven resizing and positioning of the DirectShow object, with some additional help from the Window Control object, you could create a video player with all the similar functions, utilizing a single frame, instead of three. But let's keep this as simple as possible, shall we? It's always best to start with a bit simpler methods, even if they're not as effective as their more advanced counterparts.

Part IV: Some more theory – Global Values, Global Strings, Qualifiers

It's time for some more theory, true believers! To make our application work in the way that it's supposed to, we'll need to know a thing or two about **Global Values**, **Global Strings** and **Qualifiers**. They sound impressive, don't they? But don't be worried – using them is pretty easy.

Firstly, let's take a look at Global Values. What are they, why are they useful? And why did the chicken cross the street in the first place? And… Err. Anyways, Global Values – or "Globals", as some of MMF users, including myself, like to call them – are values stored globally in your app, accessible from any frame. Let's just illustrate this… Let's say that we have a game in which the player has to collect various items in various frames to use them in other frames and thus perform the needed action to acquire another item or just move along. Think of traditional point'n'click adventure games (minus the logic, dialogues and a clever plot :)) or the "Dizzy" series. Our player has just stumbled upon another bring-and-use-this-and-that-on-me kind of riddle. He has to unscrew the entrance to the ventilation shaft

(which is located in frame 12) with a screwdriver (from frame 16), just to pick up an apple that was left inside, and then pass it on to the guard (in frame 19), so that he opens the gate (from frame 20) and our little brave hero can continue with his quest to find the golden tomato, or whatever. Achieving this will be easy. All we have to do, is to create some Global Values (you could either split the whole thing to various GV's, or use a single one – I'll illustrate this by using a single GV, named "Game progress"). At the start, our global is set to 0. That means that the ventilation shaft is locked, player doesn't have the screwdriver in his inventory, nor the apple, and the guard keeps the gate closed. When the player wanders off to frame number 12, he sees the ventilation shaft – but can't unscrew it, since he doesn't have the screwdriver yet ("Game progress"=0). Then, the player goes to frame 16, get's the screwdriver (change the "Game progress" to "1"). If he'll try to go through the gate in frame 20, he will find that it is locked (because "Game progress" < 4). If he'll try to talk to the guard, he will just be ignored. The player has to go back to frame 12, press the action key while standing right by the ventilation shaft, and – the game checks if the "Game progress" value is equal to "1" – voila! The entrance to the ventilation system is unlocked ("Game progress" is changed to "2") and player can get the apple (when he does – "Game progress" get's changed to "3"). Now, all the player has to do is to give the apple to the guard in frame 19 ("Game progress" is then changed to "4"), and go through the unlocked gate in frame 20 (if "Game progress" = 4, then: unlock the gate and let the player go through). Easy? Sure it's easy. And it's a good idea to use the Global Values here and there, since they're the fastest data storing method in MMF2.

All right, we already know what GV's are and what you can use them for. Well, the same basic idea was used to create Global Strings – the only difference is that you can store strings (letters, text, etc.) inside them, instead of values. If you're creating a game where each player has to create a named profile for his settings, or where the players are enabled to give a name to their characters – Global Strings will come in handy.

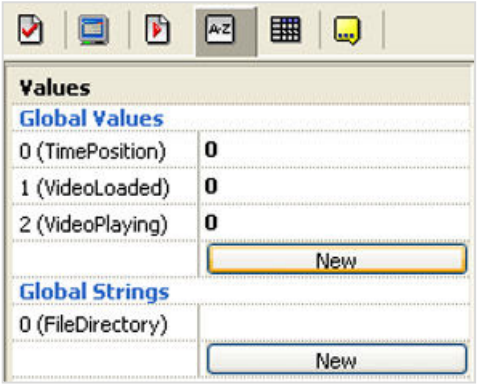
OK, Global Values and Global Strings are out of our "to-learn" list. It's time to have a little chat about Qualifiers. Firstly, let's see what MMF2's built-in help has to say about them:

> "*A qualifier is a small icon and a name, that can be applied to several objects. For example, the "Bad" qualifier can be applied to all of your enemies in a game, even if they are different objects. They all can have the "Bad" qualifier. Under the event editor, the qualifier will appear in the list of objects. You can use the qualifier as you would do for any object. You now understand that it simplifies a lot programming the same event for all the baddies: instead of entering the action or condition for each of the bad guys, you simple use the qualifier "Bad" and everything is done in one line*".

Quite simple, eh? Let's illustrate this with an example… Let's say that you have a game in which there are a lot of collectibles, all doing the same: adding 100 points to the player's score. And when I say a

lot, I mean A LOT: apricots, bananas, acorns, strawberries, grapes, apples, pineapples, nuts, fish, french fries, tomatoes, pumpkins, carrots, cucumbers, broccoli, coins, bullets, first aid kits, high-end gadgets and a lot of different toys. Well, sure, you could add a line of code (if: collision between an [specified object] and the player, then: destroy the object, add 100 points to player's score) for each and every one of those collectibles, but what for? It's a lot easier (and eventually more efficient when it comes to system resources) to apply a single qualifier (let's say: the "bonus" qualifier) to all of them, and then just use a single event (if: collision between [group "bonus"] and the player, then… etc.) – and presto! We'll have it done in a jiffy!

See, I've told you that all this theory stuff will be TOO easy for ya'. It would be great to see this theory in practice – and you soon will. But before we'll get to that – let's set it all up in the way it's meant to be. Go to your application's properties and open the "**Values**" tab (third from the right – look at the screen to the left for visual guidance). **Create three new Global Values and name them "TimePosition", "VideoLoaded" and "VideoPlaying"** (they will help us control the DirectShow object in-between frames, when the player will change one mode to another). Then, when that's done, create a single Global String and name it **"FileDirectory"** (it will store the directory in which the currently loaded video is in). Ready? Great! Now, go to the Frame editor of the first frame (the standard mode one). Select one of the actives that we use as buttons ("button_play", "button_stop", "button_back", etc.), open its properties and go to the "Events" tab. You'll see that I've added all those buttons into one **qualifier** group – "Bonus" (the one with the yellow star icon). Keep that in mind, it will come in handy a bit later.



Part V: Setting up the Global Events

OK, so here we are, at the place where we left off at the end of part II. In other words: it's time to create some **Global Events**. Go to your application's properties and open the "**Events**" tab (second from right). Click on the Edit button. The Global Event editor will open. Click on the "**Import new objects**" icon (it's the last object in the horizontal objects list) and import the **Common Dialog object** (select it from the list). Ready? Great! It's now time for some programming.

If you have read my "Glob Wars" tutorial, you'll be familiar with most of this (having said that, I encourage you to read through this again, since I've introduced some changes – this system is currently a 1.1 beta ;)) – but if not, let me introduce you to my MMF2-to-paper event-recording system:

IF (Condition): [Object for the condition] > Condition group > *Condition*
THEN (Action): [Object for the action] > Action group > *Action*

All the conditions are marked in red color, while actions are written in 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 we'll have:

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*

If you'll have to input anything by keyboard (for example: a value to set the counter to, or a text that is going to be displayed with the alterable string option – or other things that you use the expression editor for) it will be indicated by coloring the selected text in green and using < angle brackets >, like in this example (note that sometimes the given text will be set *Italic* for easier recognition):

< Set the Global Value A to 50 >

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

<< Select any music from the MMF2 midi library >>

Pretty simple, right? Let's see it in action, then.

1) Remember what did we want to achieve with those Global Events? We wanted to control our custom menu, globally. Let's do this, starting with the "Load a video" option from the menu we've created. Create a new event:

IF: [Special Object] > Application menu > *Has an option been selected?*
<< Click on the "click here" button and select the "Load a video" option from the "Load" menu >>
THEN: [Common Dialog Object] > *Show Open Dialog*

2) Second on the list, we have the "Restart" menu option:

IF: [Special Object] > Application menu > *Has an option been selected?*
<< Click on the "click here" button and select the "Restart" option from the "Program" menu >>
THEN: [Special Object] > Change a global value > *Set*
< Set *VideoPlaying* to 0 >

THEN: [Storyboard Controls] > *Restart the application*

Two down, three to go.

3) And another one (the "Exit" option – alternatively, you could use the default MMF2 "quit" menu option, there's not really a reason to do this on your own… I'm just a bit whacky):

IF: [Special Object] > Application menu > *Has an option been selected?*
<< Click on the "click here" button and select the "Exit" option from the "Program" menu >>
THEN: [Storyboard Controls] > *End the application*

4) Here's the "Standard view" option – when selected it will return the player to frame 1 and play the movie from the right position:

IF: [Special Object] > Application menu > *Has an option been selected?*
<< Select the "Standard view" option from the "Program" menu >>
THEN: [Storyboard Controls] > *Jump to frame*
<< Click on the "use a calculation" button >>
< input: *1* >

5) And – finally – we've got to the last global event. This one is used to determine what filters will be applied to the "load file" dialog window, and – furthermore – how will this window be titled.

IF: [Storyboard Controls] > *Start of frame*
THEN: [Common Dialog Object] > *Set Dialog Title*
< input: *" Choose a video file to load "* >
THEN: [Common Dialog Object] > Filters > *Add filter*
< input (1): *" AVI Files "* >
< input (2): *" *.avi "* >
THEN: [Common Dialog Object] > Filters > *Add filter*
< input (1): *" MPEG Files "* >
< input (2): *" *.mpeg "* >
THEN: [Common Dialog Object] > Filters > *Add filter*
< input (1): *" MPG Files "* >
< input (2): *" *.mpg "* >
THEN: [Common Dialog Object] > Filters > *Add filter*
< input (1): *" All files "* >
< input (2): *" *.* "* >
THEN: [Window Control Object] > Resize > *X size*
< input: *410* >

THEN: **[Window Control Object] > Resize > *Y size***

< input: *441* >

That's done! We've just prepared all the Global Events that we will need – the rest of our code will be scripted directly in the normal Event editor, frame by frame. If you wish, you can take a look at the screenshot below and compare it with your Global Event editor (note that it should look similar, but it doesn't have to be identical – you could have created your events in a different sequence).

	All the events All the objects												
1	• Start of Frame											✓	
2	• Menu option "Load a video" selected											✓	
3	• Menu option "Restart" selected	✓		✓									
4	• Menu option "Exit" selected			✓									
5	• Menu option "Standard view" selected			✓									
6	• New condition												

Ready? Exit the Global Events editor, save your work and proceed to part VI of this tutorial!

Part VI: Coding the events – the standard mode

Finally, it's time to sit down a bit and do the usual MMF2-business: code the events, combine all those little gears and other parts to make the whole machinery come alive. Let's do this, so we can have a fully operational "Fusion Player" minutes away. Open the first frame, go to the **event editor**. Create all the events, one after one:

1) Firstly, let's start with the "**Start of frame**" event:

IF: **[Storyboard Controls] > *Start of frame***

THEN: **[Special Object] > Application menu > *Disable***

<< Select the "Standard view" option from the "Program" menu >>

THEN: **[Storyboard Controls] > Screen > *Windowed mode***

THEN: **[Group.Bonus] > Visibility > *Make object invisible***

THEN: **[DirectShow Object] > General > *Stop***

2) Here's an event that makes our little horizontal bar counter a more useful thing – it will show the playing movie's progress:

IF: [Special Object] > *Always*
THEN: [Counter] > *Set Counter*
<< Click the "retrieve data from object" button >>
<< Select [DirectShow] > General > *Get current position* >>

3) A simple event that shows the specified button from the "Bonus" group when they are highlighted by the mouse:

IF: [Keyboard & Mouse] > The mouse > *Check for mouse pointer over an object*
<< Select [Group.Bonus] from the list >>
THEN: [Group.Bonus] > Visibility > *Make Object Reappear*

4) Here's the negated version of the event above. To negate, just create the given condition, right click on it and select the "negate" option from the menu.

IF: [negated] [Keyboard & Mouse] > The mouse > *Check for mouse pointer over an object*
<< Select [Group.Bonus] from the list >>
THEN: [Group.Bonus] > Visibility > *Make Object Invisible*

5) This event will make sure that all is displayed correctly and starts from the right position when the player will return to the standard mode from either the full screen or maximized modes.

IF: [Timer Object] > *Is the timer equal to a certain value?*
<< select: 0,5 second >>
IF: [Special Object] > *Compare to a global value*
<< VideoPlaying is equal 1 >>
THEN: [DirectShow Object] > General > *Load*
<< Press the "Expression" button to open the Expression editor >>
< input: *FileDirectory* (note: no quotation!) OR press the "retrieve data from an object" button, and then go to *Special Object / Retrieve a global value / FileDirectory* >
THEN: [DirectShow Object] > General > *Set position*
<< Press the "Expression" button to open the Expression editor >>
< input: *TimePosition* OR find the TimePosition global value with the "retrieve data…" browser >
THEN: [DirectShow Object] > General > *Play*

At this point of our little coding adventure, we should have established something looking like this:

	All the events All the objects																	
1	• Start of Frame	✓		✓						✓				✓		✓		
2	• Always												✓					
3	• Mouse pointer is over															✓		
4	• ✕ Mouse pointer is over															✓		
5	• Start of Frame + VideoPlaying = 1									✓			✓					

6) Create a new **group of events** (right click on an event number and select *Insert* > *A group of events*) – name it "Loading a new file". This is not necessary – our program will work in the same way, whether these groups are inserted in, or not. It's just a little habit that I'd like to inject into your mind, as both creating groups and comments can really make your life easier, as they make your code better organized. If you wish, you can create a comment here as well (for example: input "These events will enable us to load our files"). After you're done with playing with your groups, just create this simple event (inside the "Loading a file" group):

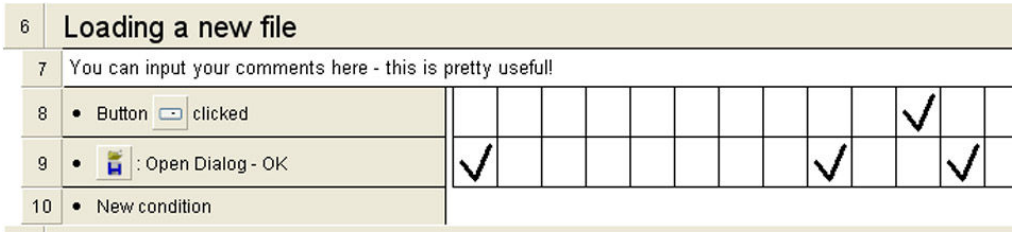
IF: [Button_LOAD] > *Button clicked?*
THEN: [Common Dialog Object] > Show Open Dialog

7) Create another event, in the same group as the one above. This one will control all the actions that are made when the user chooses the video he wants to load and presses the "OK" button:

IF: [Common Dialog Object] > *After Open Dialog*
THEN: [Special Object] > Change a global value > *Set*
< Set VideoLoaded to 1 >
THEN: [Special Object] > *Set global string*
<< Select the "FileDirectory" Global String >>
< Either input this: *GetFilename$("Common Dialog object")* OR click on the "retrieve data from an object button, select the Common Dialog object, and then the "Get Filename" option >
THEN: [DirectShow Object] > General > *Load*
<< Press the "Expression" button to open the Expression editor >>
< input: *FileDirectory* (note: no quotation!) OR press the "retrieve data from an object" button, and then go to *Special Object / Retrieve a global value / FileDirectory* >
THEN: [DirectShow Object] > General > *Play*
THEN: [Counter] > *Set minimum value*
< input: *0* >
THEN: [Counter] > *Set maximum value*

< either input this: *GetDuration("Direct Show")* OR click on the "retrieve data from an object button, select the Direct Show object, "General" category and then the "Get Duration" option >

Here's a little visual aid (just the "Loading a new file" group) that you might be interested in. Remember that your Event editor **does not** have to look identical:



8) Create another event group – name it "Buttons". It will contain all the events used to control those actives that act as buttons. Create an event inside – this one will control the "Stop" button, stopping the video when it's pressed:

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

<< Left button, single click >>

<< Select the [button_stop] object >>

THEN: [DirectShow Object] > General > *Stop*

9) Another one – this time it's the "back" button, which sets the video's position to 0 (returns to the first frame of the loaded video):

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

<< Left button, single click >>

<< Select the [button_back] object >>

THEN: [DirectShow Object] > General > *Set position*

< input: *1* >

10) And what do we have here? Ahh, what a surprise! More buttons… This one enables the "forwarding" option – as long as the user holds the left mouse button pressed and the pointer above this button, it adds 6 seconds to the video's time:

IF: [Mouse & Keyboard] > The mouse > *Repeat while mouse key is pressed*

<< Left button >>

IF: [Mouse & Keyboard] > The mouse > *Check for mouse pointer over an object*

<< Select the [button_forward] object >>

THEN: **[DirectShow Object] > General > *Set position***

< input: *GetPosition("Direct Show")+6000* >

11) Very similar to the event above, this one subtracts 6 seconds from the current video's position, enabling the user to smoothly rewind his video:

IF: **[Mouse & Keyboard] > The mouse > *Repeat while mouse key is pressed***

<< Left button >>

IF: **[Mouse & Keyboard] > The mouse > *Check for mouse pointer over an object***

<< Select the [button_rewind] object >>

THEN: **[DirectShow Object] > General > *Set position***

< input: *GetPosition("Direct Show")-6000* >

12) This one enables the user to enter the full screen mode – it saves the current video's position in the "TimePosition" global value and then jumps to frame number 3:

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

<< Left button, single click >>

<< Select the [button_fullscreen] object >>

THEN: **[Special Object] > Change a global value > *Set***

< set TimePosition to *GetPosition("Direct Show")* >

THEN: **[Storyboard Controls] > *Jump to frame***

<< Click on the "use a calculation" button >>

< input: *3* >

13) This event controls the button that enables the user to enter the maximized mode (frame number 2, the 640x480 screen mode):

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

<< Left button, single click >>

<< Select the [button_maximise] object >>

THEN: **[Special Object] > Change a global value > *Set***

< set TimePosition to *GetPosition("Direct Show")* >

THEN: **[Storyboard Controls] > *Jump to frame***

<< Click on the "use a calculation" button >>

< input: *2* >

14) This event makes our little app a bit cleverer – when the program sees that no video has been loaded, it asks the user to open one via the file selector, before going into the full screen mode:

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

<< Left button, single click >>

<< Select the [button_fullscreen] object >>

IF: [Special Object] > *Compare to a global value*

<< VideoLoaded is equal 0 >>

THEN: [Common Dialog Object] > Show Open Dialog

15) The same as above, only this time it's true when the maximized mode is going to be initiated:

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

<< Left button, single click >>

<< Select the [button_maximise] object >>

IF: [Special Object] > *Compare to a global value*

<< VideoLoaded is equal 0 >>

THEN: [Common Dialog Object] > Show Open Dialog

16) Woo-wee, we've got quite a speed here… Another event that does practically the same as the one above, only this time covering up the "play button":

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

<< Left button, single click >>

<< Select the [button_play] object >>

IF: [Special Object] > *Compare to a global value*

<< VideoLoaded is equal 0 >>

THEN: [Common Dialog Object] > Show Open Dialog

17) Finally! Something a bit different! This event controls the "Play" button and… Errr… Covers the whole screen in Smurfs, running around with machine guns. Well, maybe not exactly, but close enough - it just plays the whole video:

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

<< Left button, single click >>

<< Select the [button_play] object >>

IF: [Special Object] > *Compare to a global value*

<< VideoLoaded is equal 1 >>

THEN: [Direct Show Object] > General > *Play*

18) This one is pretty clever too. It skips 1/6 of the movie (no matter if the movie is a 3 hours long action-flick, or a 3 minutes long commercial), enabling the user to travel through the whole video in light

speed. You could easily do the expression used here yourself – just use the “retrieve data from an object” and play with the Direct Show object a little.

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

<< Left button, single click >>

<< Select the [button_fastforward] object >>

THEN: **[DirectShow Object] > General > *Set position***

< input: *GetPosition("Direct Show") + (GetDuration("Direct Show")/6)* >

19) The same as above – but working backwards:

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

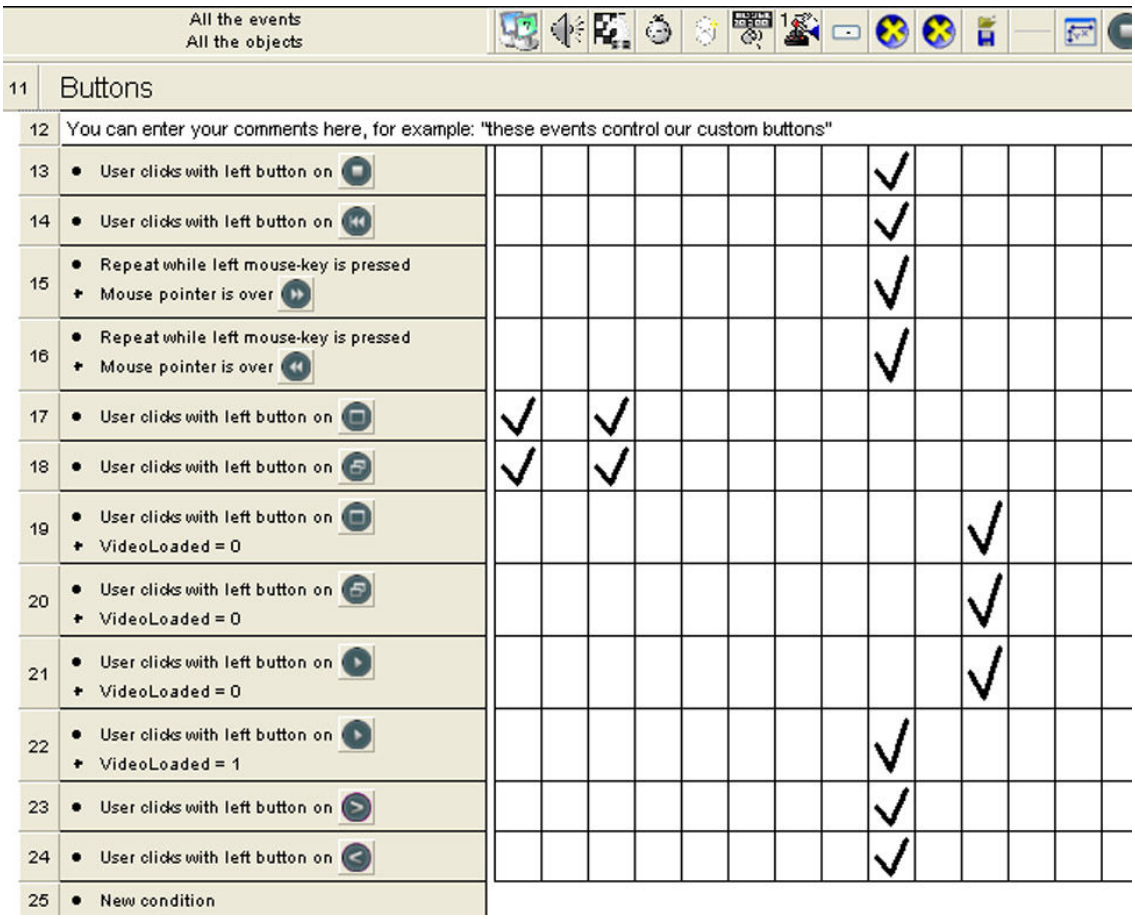
<< Left button, single click >>

<< Select the [button_fastrewind] object >>

THEN: **[DirectShow Object] > General > *Set position***

< input: *GetPosition("Direct Show") - (GetDuration("Direct Show")/6)* >

If you wish, you can compare your events to the ones shown below:



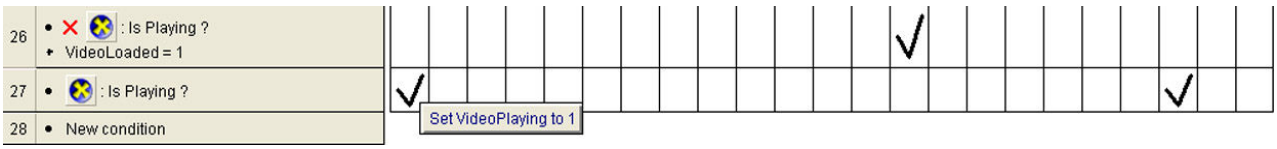
20) Here's something that I would put out of the "Buttons" group, so… No more "Buttons" group here, people. Of course, don't be worried if you didn't put all the above events to that group – it really doesn't matter, it's just for keeping our code clean and understandable. Anyways, create such an event:

IF: [negated] [Direct Show Object] > General > *Is Playing?*
IF: [Special Object] > *Compare to a global value*
<< VideoLoaded is equal 1 >>
THEN: [button_stop] > Visibility > *Make Object Reappear*

21) And here's the last event in our first frame!

IF: [Direct Show Object] > General > *Is Playing?*
THEN: [Special Object] > Change a global value > *Set*
< set Videoplaying to *1* >
THEN: [button_stop] > Visibility > *Make Object Reappear*

Here are the last two events for comparison:



We've finished creating the "Fusion Player's" standard mode! We're almost at the finish line! Hooray!

Part VII: Coding the events – full screen and maximized modes

When it comes to the two other frames – it's really a piece of cake. The only problem is that SOMETIMES (not always, and not on all system configurations) the DirectShow object seems to have a bit of a problem with resizing while the screen resolution is changed. I'm not sure what's causing this, but it's hardly reproductive, and rather a minor problem… There's an easy way around this, though, and we're going to use it here – all we have to do is to destroy the DirectShow object at the frame's start and create another one after a small amount of time, to replace the previous one.

1) Here's the "Start of frame" event. As I said earlier – we're going to destroy the DirectShow object + we will enable the option from the menu that was inactive while the user was in standard mode.

IF: **[Storyboard Controls] > *Start of frame***

THEN: **[Special Object] > Application menu > *Enable***

<< select the "Standard view" option from the "Program" menu >>

THEN: **[Direct Show Object] > *Destroy***

2) Time to change the resolution and create the new DirectShow object as well!

IF: **[Timer Object] > *Is the timer equal to a certain value?***

<< select: 0,5 second >>

THEN: **[Create New Objects] > *Create Object***

<< Select the "DirectShow" object >>

<< Set the coordinates to x=0, y=0 >>

THEN: **[Storyboard Controls] > Screen > *Full Screen Mode***

3) After another 0,5 seconds the video will be loaded and played at the saved position:

IF: **[Timer Object] > *Is the timer equal to a certain value?***

<< select: 1 second >>

THEN: **[DirectShow Object] > General > *Load***

<< Press the "Expression" button to open the Expression editor >>

< input: *FileDirectory* (note: no quotation!) OR press the "retrieve data from an object" button, and then go to *Special Object / Retrieve a global value / FileDirectory* >

THEN: **[DirectShow Object] > General > *Set position***

<< Press the "Expression" button to open the Expression editor >>

< input: *TimePosition* OR find the TimePosition global value with the "retrieve data…" browser >

THEN: **[DirectShow Object] > General > *Play***

4) If the user exits this mode, the video's position will be saved to the TimePosition global value:

IF: **[Storyboard Controls] > *End of frame***

THEN: **[Special Object] > Change a global value > *Set***

< set TimePosition to *GetPosition("Direct Show")* >

5) And here's the last event that we're going to script today…

IF: **[Common Dialog Object] > *After Open Dialog***

THEN: **[Special Object] > Change a global value > *Set***

< set VideoLoaded to 1 >

THEN: **[Special Object] > *Set global string***

<< Select the "FileDirectory" Global String >>

< either input this: *GetFilename$("Common Dialog object")* (note: quotation as given!) OR click on the "retrieve data from an object button, select the Common Dialog object, and then the "Get Filename" option >

THEN: [DirectShow Object] > General > *Load*

<< Press the "Expression" button to open the Expression editor >>

< input: *FileDirectory* (note: no quotation!) OR press the "retrieve data from an object" button, and then go to *Special Object / Retrieve a global value / FileDirectory* >

THEN: [DirectShow Object] > General > *Play*

And here's the usual "compare if you wish" screenshot:

	All the events All the objects										
1	• Start of Frame	✓							✓		
2	• Timer equals 00"-50			✓		✓					
3	• Timer equals 01"-00								✓		
4	• End of Frame	✓									
5	• : Open Dialog - OK	✓							✓		
6	• New condition										

6) Now… Want to hear a really funny thing? The third frame (the Full Screen mode) needs EXACTLY the same events to work as the maximized mode! You can either code them again, just for practice, but this time in the third frame – or you can do as I did: just copy all the events from frame two and paste them into the blank event editor from frame three… Either way: just get the same events into the third frame, and we're done!

That's it!

Take a look at your application – save it, build it as a stand-alone app (it should work better as a stand-alone, having a bit faster loading times), run it and use it to watch some nice movie or something! If you'd like to – play with the "Fusion Player" a little, add new functions, add a playlist (just use a list or multiple Global Strings), or MP3 support (actually – it HAS MP3 support right now… All you have to do is to load an MP3 file or an JPEG image – they all should work well)… Or something else. Unleash your imagination! It's all up to you! Have fun!

Cheers!

Koobare

marchewkowy@gmail.com