Multimedia Builder Help

Title page 1

Use this page to introduce the product

by Mediachance

This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)

Multimedia Builder Help

Printed: November 2007

Publisher

© 2007 Mediachance

Team

Roman Voska (Oscar)
Peter Misik (Orol)
Pavel Kudrys (Odklizec)

Special thanks to:

Thanks to all of you who sent me any suggestions or just nice words.

Special Thanks to:
Edd Twilbeck - for his ideas and encouragement and of course for the logo.
Derrick Yohn - for making MMB web space reality not just my dream.
Artur Svarc - for his beta testing.
Ronnie Toon - for sending me pages and pages of great suggestions.
Pierre Labelle - for Helping me with the Help File.
Mandryka, Gotlib and Sconi for their Bird and Worm Flash samples.

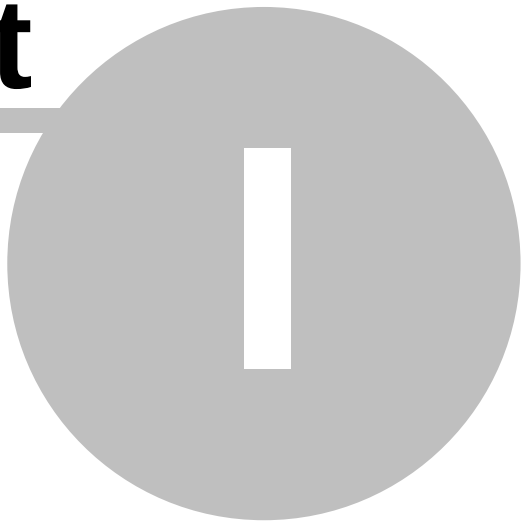
My great wife Zuzana - for her endless patience (she really hates computers now!) and many other people...

Table of Contents

Top Level Intro

This page is printed before a new top-level chapter starts

Part



1 Introducing Multimedia Builder

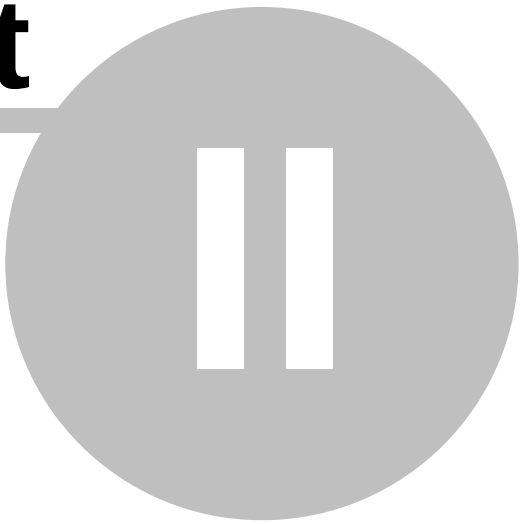
Multimedia Builder is a Windows-based multimedia authoring system that allows you to create autorun CD menus, Multimedia Applications on CD-ROM, Demos, Presentations, Audio players and much more. The software is great for beginners as well as for advanced users.

MMB creates small stand-alone Windows exe applications and has all the bells & whistles you will ever need. You will love this easy-to-use, intuitive software.

Top Level Intro

This page is printed before a new
top-level chapter starts

Part



2 General Overview

Easy to learn, Non-expensive Software

Now you can develop multimedia apps, autorun menus or front-ends for your CD's without having to spend months learning any programming language.

Make an autorun for your CD-ROM

If you are already developing software on CD-ROM, creating your own CD-Audio or making your audio CD you will love this easy-to-use, intuitive software.

Multimedia applications in minutes !

Create multimedia applications with graphic, text, sounds, audio ,Video, supporting CD Audio or Mixed-mode CD's, executing applications and much more...
Apply many **cool effects** to your images.

MMB has many great features!

MMB creates small stand-alone exe applications and has all the bells & whistles you ever need.
Create a cool app and send it to all your friends for kudos.

With MMB you can create:

- Autorun CD browsers (menus) for corporate CD-ROM's.
- Tutorials
- Cue Cards
- Kiosks
- CD Audio and Mixed-mode CD Audio Players
- Audio Players
- Front-end for your corporate or personal CD's
- File launchers and toolbar
- Computer based training

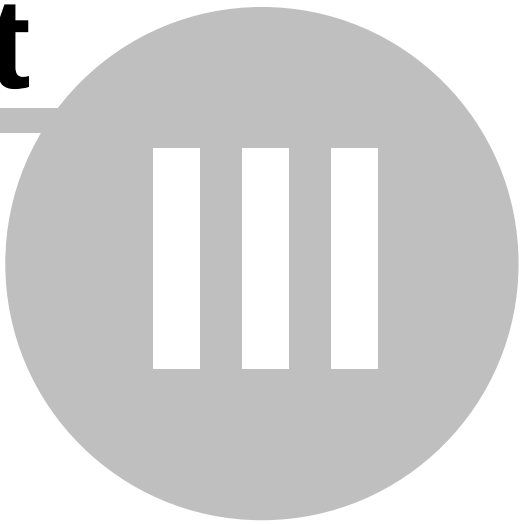
Note:

In this help, we skip all the basic GUI descriptions and go directly to the bone - describing the functions, objects and how to use the software in general. We are sure you will find the interface very logical and easy to use.

Top Level Intro

This page is printed before a new
top-level chapter starts

Part



3 What's new?

Updated 4.9.8 (4.9.8.13)

[ADDITIONS]

+ Added an option to enable window Maximize and Resize (by dragging the application border). These new options can be found in Project Settings.

+ Added Maximized and IsMaximized() scripting function - to detect if the window is in maximized state.

+ Added an option to run MMB scripts from HTML page (loaded in Browser object).

[FIXES]

- Fixed problem with Flash fscommand.

- PluginGet/PluginSet now supports correct notation of string variable arrays - str$[idx]

- Fixed the size of rectangle object placed above the windowed objects (like HTML, Flash or ListBox).

- Fixed the primitive object border color bug (the border color field is enabled, even though is selected none, sunken or windows).

- Fixed font bug in Text object (the Text object created from script inherits the font, which was previously set to another object - designer related problem).

- Fixed crash in StrDel if count index > number of characters in passed string (designer related crash).

Initial 4.9.8 (4.9.8.7)

[ADDITIONS]

+ Added new Script function RunScriptCode("parameter1","parameter2") for running external scripts.
parameter1: string or string variable holding the script source code
parameter2: 0=quiet parsing, 1=loud parsing (1 = runtime errors are displayed)

+ Added an option to run MMB scripts from 3rd party plugins (see MMBPlugInSDK.h and CommandLine.dll sample)

+ Added 1000 new ScriptTimers - should be enough for all ;)

- The syntax of new timers is following: ScriptTimer("Timer1=name",100") to ScriptTimer("Timer1000=name","100").
- Original Timers (TimerA,TimerB,TimerC) remains also in 4.9.8, but it's recommended not to use them in new projects. TimerA=Timer1, TimerB=Timer2,

TimerC=Timer3

+ Added an option to Delete objects in a runtime using the DeleteObject("name$") function - good for freeing memory.

+ Added an option to create objects in a runtime (via Script):

  - Text Button - CreateTextButton("inlabel","outlabel$,x,y,w,h,text")
  - Text Label - CreateText("inlabel","outlabel$,x,y,text")
  - Paragraph - CreateParagraph("inlabel","outlabel$,x,y,w,h,text")
  - Circle - CreateCircle("inlabel","outlabel$,x,y,w,h,r,g,b")
  - Line (relative positioning) - CreateLine("inlabel","outlabel$,x,y,w,h,r,g,b")
  - Line (absolute positioning) - CreateLineAB("inlabel","outlabel$,x1,y1,x2,y2,r,g,b")
  - Rectangle - CreateRectangle("inlabel","outlabel$,x,y,w,h,r,g,b")
  - HotSpot - CreateHotSpot("inlabel","outlabel$,x,y,w,h")
  - Script object - CreateScript("inlabel","outlabel$")

+ Extended SetObjectParam function to allow changing the object's script:

  - Added new parameters in SetObjectParam (parameter:subparameter=value):

    - For Circle, HotSpot, Rectangle, Text, Text Button, Bitmap and Bitmap Button:
      parameter: MOUSEDOWNSCRIPT/MOUSEUPSCRIPT
      subparameter: 0 - quiet parsing or 1 - loud parsing
      value: source code/variable$

      example1:
      ** this adds (replaces the actual object's script with..) "Message" function to "obj" object (with loud parsing)
      SetObjectParam("obj","MOUSEUPSCRIPT:1=Message("Hello there!","")")

      example2:
      ** the same as above now with using the string variable (quite parsing)
      code$='Message("","a")'
      SetObjectParam("obj","MOUSEDOWNSCRIPT:0=code$")

    - For Script object:
      parameter: SCRIPT
      subparameter: 0 - quiet parsing or 1 - loud parsing
      value: source code/variable$

      example1:
      SetObjectParam("obj","SCRIPT:1=Message("Hello there!","")")

      example2:
      code$='Message("","a")'
      SetObjectParam("obj","SCRIPT:0=code$")

+ Added an option to change the size of selected object from keyboard (in designer) -

Ctrl/Shift + Left/Up/Right/Down

+ Added an option to change the Line orientation and size using the - (minus) value in the Dimensions panel
+ Added an option to use zero or negative width/height in object creation scripts (good mainly for Line creation).
+ Added CBK_AppFileName, which returns the filename of actual project (in format "Name"."Ext").
+ Added an option to change/delete the content of "Comments" field in runtime player File Version info (the one displaying "Created with Multimedia Builder, version x.x.x"). This requires purchasing of special comment unlock code. After entering the correct code into MMB >> About box there will be displayed new "Comments" field in "Compile" dialog.

[FIXES]

- Fixed MS VISTA compatibility problems. So MMB is now MS Vista Ready :)
- Fixed incorrect FONTSTYLE and FONTEFFECT behavior.
- Fixed WinXP manifest in designer/player resources.
- Fixed 4.9.7 bug that caused MediaFX plugin to stop working.
- Fixed 'ya' cyrillic character in EditBox.
- Fixed Script Editor background color. The color is now obtained from system settings.
- Fixed minor bug in script text color highlighting.
- Fixed the keyboard input to allow to send <Left> and <Right> keys to windowed objects (e.g. HTML object).
- Fixed recursive calling of scripts in Page Start/End script (and so the crash caused by recursive calls).
- Fixed bug with decimal number entries in EditBox object. Previously, the decimal numbers were always converted to Integer, even if the EditBox was set to Float.
- Fixed double click selection of a long text in scrollable EditBox.
- Fixed the "Windows Desktop Component" (menu Project >> Project Settings) allowing to run the application on a Desktop.
- Fixed overwriting of the first line in text file using the function StrToLine.
- Fixed crash caused by "Vertical Flip" effect, applied on a resized Bitmap object.
- Fixed reading of large files using the StrFromFile. Sometimes it read were read two lines instead of just one.
- Fixed sending of command line parameters to an already running application with "Allow only one instance" switch. Previously, the parameter sent to one selected

application was sent to all running MMB apps;)

- Fixed displaying of Horizontal Scrollbar if the ListBox is hidden, filled and shown via script.
- Fixed "Cover Windows Taskbar" switch if selected "Full Screen Background". It now works correctly and the full screen background is now automatically updated after resizing the taskbar or changing the screen resolution.
- Fixed displaying text of specific length in the Paragraph object.
- Fixed reading of single line file created by StrToFile and its reading by StrFromFile.
- Fixed PrintPage crash caused by resizing of the application window using the SysCommand function.
- Fixed disappearing of the FullScreen background after maximizing the application window and pressing the <ESC> key (instead of exiting the project).
- Fixed displaying of CBK_ReturnVal in debug window.
- Fixed "Minimize" function and Full Screen background if the Minimize is used from Page Start script.

[CHANGES]

+ Modified Line settings dialog:
  - Values x1,y1 are now coordinates of first point in Line object, while the values x2,y2 are coordinates of second (end) point in Line. Previously, these values were the coordinates of left/top - bottom/right corner of the Line bounding box.
  - Allowing to use negative Line coordinate values.
+ Changed version of mbd file. The files saved with 4.9.8 will no longer openable in 4.9.7!

Top Level Intro

This page is printed before a new top-level chapter starts

Part

4 History

What's new in 4.9.7?

ADDITIONS:

- Added fully featured and configurable Message box! The function is called **MessageEx**.
- Message and MessageEx now uses WinXP visual styles. Sorry, no support for the rest MMB objects yet ;)
- Added **SetObjectParam**("object","parameters") for setting common aspects of some basic MMB objects.
- Added **FontPicker**() function, which will open the standard Font dialog (similar to ColorPicker). In this dialog you can set the various font parameters and the result is a CBK_FONT containing all font parameters in a string array.
- Added **SetProjectParam**("","") for changing the project parameters in a runtime. For now, it will allow you to set/change the page/master page/project background image/color.
- The workspace area is now scrollable with size 3072x2304. We think it should be enough..at least for few next versions ;)
- Added **StrToLine** function: StrToLine(filename$,string$,toline,overwrite) Simply, it's a function to add/replace a line in a text file.
  fileaname$ = path to text file (not binary files!)
  string$ = string to add to line
  toline = number of line to add
  overwrite = TRUE/FALSE - if true, 'toline' line is replaced otherwise moved down
- Added new **CurrentObject**() which returns the name (label) of the current object in which was this function called.
- Added new **ImageOpacity**("ImageObject","opacity") which allows you to set the image (Bitmap object) opacity.
  opacity = 0 object is fully transparent, opacity = 100 object is fully opaque
- Added **GetImageOpacity**(Bitmap) function. With this function you can get the current opacity of the given Bitmap object.
- Added function for object reordering in a runtime **ReorderObject**("Object $","FRONT/BACK/FORWARD/BACKWARD")

FRONT - move object to Front

BACK - move object to Back

FORWARD - move object one step up

BACKWARD - move object one step back

- Added Cut/Copy/Paste to right click menu to **EditBox** (in a runtime).
- Added "Enable scrolling" option in EditBox. This option allows scrolling text when typing inside the Editbox, in case the text is longer than a defined width of EditBox. The "Fixed Width" must be enabled.
- Added text selection inside EditBox via Shift + Left/Right arrow key.
- Added Password mode to EditBox object! EditBox in password mode displays * characters instead of typed characters. This mode can be set both via EditBox properties or by SetObjectParam("object","PASSWORD=TRUE/FALSE") function.
- Added "Enable menu" option to EditBox properties, which enable/disable rclick menu (and shortcuts) in EditBox.
- Added comment block to Script window. With this sequence of characters /* */ you can comment multiple lines of code at once..

  /* code

  code

  code

  code */
- PluginSet/Get/Run can now use variables both in first and second parameter.

FIXES:

- Fixed rounding when incrementing small numbers (like 0.01).
- Fixed the missing application button in task bar.
- Fixed the problem with fullscreen background and Minimize/Restore the application window to tray bar(via plugins).
- Fixed IsMinimized() problem (incorrect return value) in case of enabled full screen background.
- Fixed GetArrayItem and hexa params data$=GetArrayItem(string$,'0x2C',1)
- Fixed bug in GetArrayNum if used string array - n1=GetArrayNum(list$[1],)
- StrToFile now accept append a linefeed as a variable.
- Fixed refresh of CBK_VTotal and CBK_VTime text
- Fixed custom appearance of custom cursor if custom cursor is moved above the object with no custom cursor defined.

- Fixed bug in SearchForFiles, which returned also the files with incomplete file extension
- Fixed an inability to start anything after RunMBD in CBK_EXIT
- Fixed a designer freeze when Copy&Paste and the label of the copied object is just a number.
- Fixed - NumPad keys return numbers and not characters as previously.
- The size of script accepted per single Script window is now about 120kb.
- Fixed some cursor hot-spots.
- A lot of other internal fixes and tweaks ;)

KNOWN PROBLEMS & LIMITATIONS:

- The comment block /* */ can sometimes incorrectly highlight/dehighlight the code. Make sure the /* characters are at the start of line (as the very first characters on the script line) and the terminating */ at the end of line.
- The Font charset cannot be set on ListBox object.

4.9.6a

FIXES:

- Fixed a serious syntax checker bug.

4.9.6

ADDITIONS:

- Added a complete **Command line** support for MMB compiled apps. With this you can send an unlimited number of parameters to your applications (even already running apps with a single instance enabled!). With this new feature you can create windows screen savers including settings dialog!
- From this version you can place any rectangular object over the **Video object**! Video still plays OnTop, but it's now "masked" with the objects placed over the video rectangle.
- Completely rewritten **Run** command with a focus on BAT files. Now the Run command should work the same way with WAIT/TOPMOST as without these parameters. Several new Run command starting options were added too.
- Added a new predefined function **GetVideoParam**(object, videoparam), which returns the current state (0 or 1) of a given video parameter.

- Added **VideoClose**() command to Close (unload) and Hide video object.
- VideoLoad now create still image from the first frame of loaded video file. Previously it just drop the still image from previously loaded Video.
- Added new MPEGAccurate option to **ListBox Properties** and **ListBoxParam** ("ListBox","parameters") function. This will allow you to load VBR (variable bitrate) audio files into ListBox with correct time.
- Added new **ListBoxSortItems** option - sorting by VALUE.
- Added an option to dynamically change the range of **For..Next** loop counter. Thanks to this you can prematurely terminate the For..Next loop under a certain conditions.
- Added a **Break**() function to jump from **For..Next** loop. With this function you can jump from For..Next loop, but you can still continue with the main script. Previously, there was only **Return**() function, which skips entire script.
- **SongListLoad** now loads the lists in a given format. For example, TXT file can be read as M3L songlist format or vice versa. This is useful for text files editing. Also, if you want to use TXT file as a songlist storage, you have to always read it as M3L file, because it uses some songlist reading improvements.
- Batch files (*.bat) can now be run with **Run** with WAIT parameter.
- Added "Web Archive (*.mht file) to **HTML object** and **Browser** function.
- Added new **Flash**("Flash","MINMENU/FULLMENU") which switch the Flash right click menu to minimized or full mode.
- Added some new properties to **FlashGetProp** function.
- **ScriptTimer** started and looped on Master Page/Layer will not be terminated after jump from one page to another (as it was previously).
- Added a **Restore**() function to restore minimized project window to its original size and position (before **Minimize**()).
- Added a NEW_WINDOW parameter to **RunMBD** command. As name of parameter says, it will open mbd project in the new window.
- Added some additional parameters to **Run** command: MINIMIZE, MAXIMIZE, HIDE.
  MAXIMIZE - Start application in maximized mode.
  MINIMIZE - Start application in minimized mode.
  HIDE - Start application with hidden window (useful for command prompt or BAT files).

FIXES:

- SysCommand("MoveWindow","x,y") can be again used in Page Start script ;)
- Fixed bug in OpenFile function, which prevents to change the <File> macro.
- Fixed list of available files in Video OpenFile dialog (missing WMV file).
- Fixed bug in ListBoxAddItem function, which prevents adding the strings separated by comma character.
- ObjectWidth/Height now returns correct size of multiline Text object.
- ObjectWidth/Height now returns correct size of Video object (size of still image).
- Fixed Up/Down ScrollBar buttons in Paragraph object. Press and hold Up/Down arrow buttons keep scrolling text up or down.
- Fixed crash caused by lost focus in ListBox (minimized application window and ListBoxSelectItem/ListBoxDeselectItem).
- Fixed crash if <List> macro is assigned to string/integer variable like number=VAL (<List>) or string$=<List>. Both examples are useless, but they shouldn't crash ;)
- Fixed some bad screen redraws mainly visible in Page transition effects.
- Fixed SongListSave problem with # character.
- Fixed (we hope) the click through the Video in FullScreen mode.
- Fixed ListBoxSortItems (by TIME) - longer files were not correctly sorted.
- Fixed ListBox columns width calculation.
- Fixed If..Then formatting problem (more 'End' than 'If' or more 'If' than 'End').
- Fixed bad redraw of transparent parts of overlaid GIF objects.
- Fixed ListBoxAddItem problem with strings containing comma character.
- Fixed ListBox scrollbar bug - after resizing ListBox with MoveObject function.
- Fixed bug with parsing multiple path macros in Run command.
- Fixed running BAT (batch) files in Run command.
- SongListSave in second parameter now opens (in script wizard) "Save As" instead of "Open File" dialog.
- SongListSave no longer adds an empty line at the end of saved file.
- Fixed bug in StrFromFile that caused a loading of strange characters.
- Fixed Win9x/ME crash caused by Flash object and "Hide Menu" option.
- Fixed some memory problems and crashes on certain computers (actually, we don't know what we exactly did, but it no longer crashes;)
- Added missing I cursor in EditBox object (in a runtime).

- Fixed missing objects in Path Replace tool.
- Fixed OnFinish Video crash.
- Fixed a bug in script syntax checker.
- Fixed bug in NOL function.
- Fixed bug in CDPause regarding CBK_Time and CBK_TimeSec.
- AV object can now be copied.
- Fixed a possible dangerous part of code, which might cause a crash on some computers.
- Many other fixes and code optimizations.

CHANGES:

- VideoParam("VideoBox","FULLSCREEN") to VideoParam ("VideoBox","FULLSCREEN=ON/OFF")
- VideoParam("VideoBox","MODE=FRAME/TIME") to VideoParam ("VideoBox","FRAME/TIME") "MODE=" is no longer required.
- Flash("Flash","SHOWMENU/HIDEMENU") now enable/disable the Flash right click menu (previously it switched the menu between minimized or full mode).

KNOWN PROBLEMS:

- ASF/WMA file formats don't work with CBK_AudioEOF. Unfortunately, these MS file formats don't support the EOF detection. However, there is a way (even a little tricky) how to detect end of file. Check **this** example file.

KNOWN LIMITATIONS:

Q: Some objects placed over the "windowed" objects (like a HTML, ListBox, Binding, Video or Flash) does have an ugly white rectangle around them.
A:
It's because MMB4.9 allows only the rectangle objects to be placed over these special objects. Then if you place for example a circle over the HTML, it will still appear as a circle inside the ugly white rectangle. We will try to do something with this limitation in any next version of MMB.

Q: WMA, ASF audio formats don't fill the CBK variables.
A:
At the moment only the OGG files have the complete support for CBK variables.

What to do if you experience any problem? Read the **Support** chapter in this help file.

4.9.5

MMB Video object:

MMB **Video object** is now completely rewritten! The days of an obsolete and not reliable MCI video playback are counted! The MCI object still remains there and unchanged (for technical experts), but the Video object is now a fully featured and easy to use DirectShow player. Of course, there are some dependencies on the installed video codecs; however, if you have installed Windows Media Player 7 or higher, you should have all the usual codecs installed. Now it is possible to run multiple video objects on a page at the same time!

Bitmap object enhancements:

Added a lot of new Image related scripting commands and designer enhancement. Now you will be able to preserve the Image size, or stretch image to fit window, resize/rotate the image and preserve its aspect ratio and lots of other things. Look at the detailed**Bitmap Object description**.

Polygonal Hot-spot:

Added a **polygonal hot-spot** object. Irregular hot spots are now possible!

ZoomTool:

Added **Zoom Tool** for magnifying the selected part of MMB workspace.

MMB Script:

A lot of new scripting commands were added to work with strings, date/time, video, image/hotspot/polygon object resize and rotation, and many others. BTW, complete list of all MMB functions, CBK variables and predefined functions is now accessible over the right-click menu in MMB scripting dialog (both basic and enhanced windows) and scripting functions are no longer case sensitive. Check the **Scripting Unleashed** chapter. This chapter is really important to all of you who want to understand MMB Scripting. We really recommend you to check this chapter (thanks Bokzy! :)

OTHER ENHANCEMENTS:

- Added tooltips and ability to change the default cursor on some next objects (Bitmap, Polygon objects, Matrix, Text and Text Edit).
- Added the ability to change the color of HTML status bar and progress bar (designer).
- Added the option to run just one instance designer/compiled application.
- Added a button to delete all guide lines (menu Edit>>> Edit Guides).
- Added buttons and shortcuts to Page Manager for inserting (INS) and deletion (DEL) the publication pages.
- Added the ability to change the colors of AudioVisualization object - Equalizer (both from designer and script).

- Added a Minimize button to the standard project window.
- Added the ability to completely disable the Flash right click menu.
- Added the ability to load Flash file in its original size and maintain its aspect ratio.
- MMB4.9 files can now be saved in 4.8 compatible file format. All 4.9 related things will be removed and scripts will be amended.
- Each change in a Dimension tool can be confirmed by Enter and not only by "Apply" button as previously.
- FMOD audio library updated to version 3.6.2.
- Restored the ability to change the Player.exe icon (Compile dialog).
- The main Object toolbar is now redesigned to popups (according common features). However, if you don't like the popups, you can switch back to the previous long toolbar (View>>>Toolbars>>>Object Toolbar (simple)).
- Video Object now supports audio playback (ogg, mi, mi, midi, rmi, wav, snd, au, aif, wma, asf), but without the ID tags!
- Added the option to enable/disable multiselection, item numbering and audio time in a ListBox object.
- **LoadVideo** and **ReplaceImage** functions now supports OpenFile button in the "Path" parameter. You will no longer need to Copy and Paste the absolute paths from somewhere else (i.e. windows explorer).
- ListBox object now supports Drag&Drop from outside of MMB application (e.g. from windows explorer).
- Added the option to load the audio and video files as a parameter of a compiled MMB application. Of course, if you want to run a video from command line a video object must be available in the running application.
- Added On Load, On Start and On Stop/Finish events to the Video object (check the Run Script option in Video properties).
- Polygon object can now be resized.
- Already opened and changed projects are marked with * mark in the MMB title bar (behind the project name).
- Added the option to compile final application without the FMOD audio library (NoFMOD option in Compile dialog).
- MMB internal playlist (m3l) now supports Video files storage.
- All windowed objects (Flash, HTML Browser, Binding, ListBox or AV) can now be added on Master Page and Master Top Layer and controlled via script! This allows

you to use single Flash object on the Master Page as a project menu or single ListBox for entire project.

- Significantly improved "Snap To Grid" feature. Snap To Grid now works with all objects as expected (including creating objects and dragging the object corners! ;)
- ZoomTool floating can be disabled by toggling ZoomTool button OFF/ON (and not only by Ctrl+Space shortcut as previously).
- Improved speed of Enhanced Script window.
- Multiple embedded sounds can be now played at the same time.
- Message box caption is now set according the project window title (Project>>Project Settings menu).

SCRIPTING ENHANCEMENTS:

- Added commands to resize/rotate Bitmaps, Hotspots (w/o image) and filled polygon objects.
- **SearchForSong** now supports file extension filtering... **SearchForSongs** ("C:\",”ogg,wav").
- Added a lot of new **CBK_Time** and **CBK_Video** related commands.
- Added <Temp >, <Windows >, <System > predefined variables.
- All <> variables (like a <SrcDir >, <CD >, <System >) can now be expanded to a string variable.
- **RunScript** and **ScriptTimer** commands can now be called from the Master Page or Master Top Layer - **RunScript**("Master Page::Script")
- Added some new **predefined functions** (WinVer , ScreenCol , PubWidth , PubHeight , UsingWinNT , ...)
- Added new CBK variable for obtaining the current page name - CKB_PageName .
- WMA and ASF now returns correct CBK time variables.
- Audio CBK now supports reading OGG tags.
- Added two new CBK variables for obtaining the total time of songs in the internal song list - CBK_TotalList and CBK_TotalSecList .
- Added many new **string handling functions** .
- Added PageExit event.
- FMOD and Flash Error/Warning messages can be disabled (Compile page) and then handled by CBK_Error variable.
- Predefined functions ObjectX , ObjectY , ObjectWidth , ObjectHeight and IsVisible

now supports string variables as a parameter i.e. ObjectX(object$).

- Added new **ListBoxSort** ("object","param") function. Possible parameters: NAME, TIME, REVERSE and RANDOMIZE.
- Added new **ListBoxMoveItem**("ListBox1","PositionNum").
- Added **FMODConfig** function for changing the configuration of FMOD audio library (DX, SW or None + MPEG accurate playback) at runtime.
- Added three new predefined functions for checking the actual mouse state - MouseLButton(), MouseMButton() and MouseRButton() Returns 1 if mouse is pressed and 0 if released.
- If..End conditional loop finally supports Else ;)
- Added WAIT parameter to Run command. This will cause pausing the MMB application until external application is running. Usage: **Run**("C:\WINNT\notepad.exe","WAIT") or more advanced example **Run**("C:\WINNT\notepad.exe","TOPMOST,WAIT c:\your.txt")
- Transparent parts of the bitmap objects will not detect the mouse clicks.
- Scripts can now be saved/loaded to/from external files (Designer option...sorry guys, but the runtime loading of external script files is not yet available ;)
- Added **BackgroundPlay** , **BackgroundPause** and **BackgroundStop** commands for controlling the Background music at runtime.
- Added a syntax completion to MMB script. However, because this feature doesn't need to satisfy everyone, you can turn it OFF by right click menu in script window.
- Function **SongListSave** now allows you to save the internal songlist to file with any extension you want to use. But the file format of the saved list will be still plain txt (or MMB internal m3l format if you want) **SongListSave**("SongList","c:\test.aaa").
- Added CBK_URLPath for obtaining the actual path from HTML object.
- Predefined String functions can now be merged within the single line of code e.g. s$= CHR(55)+ CHR(60)
- **SongListLoad** now supports loading of m3l, m3u, pls and txt file formats.
- Object arrays (e.g. TextBTN[n]) now works with every function, which supports Object label.
- "Master Page::", "Master Layer::" and "Page::"prepositions now works with every function, which supports Object label. **LoadText**("Page 2::TextBTN","kuk...")
- Functions **Hide** /**Show** /**Invert** and **MoveObject** /**MoveTo** now works with

stopped as well as running video.

- Predefined functions now supports Object/String/Integer arrays. ColW= ObjectWidth(Bitmap[i])
- Page Exit now works with **RunMBD** command.
- Speed improvements regarding the Image object and MoveObject/MoveTo command.
- MMB now installs (and embed) Macromedia Flash Player 7.
- Pressing Cancel button in **ColorPicker**() dialog sets CBK_SelColor = 0.
- **ListBoxParam** () function to setup some of ListBox properties in a runtime (like a Background and Text color, enable/disable Drag&Drop, etc.).
- **ColorPicker** () function with CBK_SelColor to selecting and storing colors.
- Added Drag&Drop event tab to the ListBox script dialog.
- Possibility to change the colors of a Menu Item button style (in a design time).
- **Run** command now returns the return values from the running applications (of course if return codes were implemented in these applications). The return codes are displayed in CBK_ReturnVal.

CHANGES:

- Predefined functions are no longer case sensitive. You can type " objectwidth() " or " ObJecTWidth() " and it will be automatically converted to ObjectWidth() .
- Function **ListBoxAddItem** no longer supports RANDOMIZE parameter. Use new **ListBoxSort** ("object"," RANDOMIZE ") function instead of previous **ListBoxAddItem** with RANDOMIZE parameter .
- Predefined functions ScreenWidth and ScreenHeight are no longer filled after the publication start-up. If you want to obtain the actual screen width/height size, use the new ScreenWidth() and ScreenHeight() functions. **All older scripts that use the old syntax must be updated!**
- MXCOL and MXROW Matrix variables are no longer filled after the publication start-up. They are filled only if a Matrix object is used.
- MouseDown/MouseUp events are now performed only above parental objects. If you, for example, click and hold the mouse button above an active object (e.g. button), then you move mouse over another active object and release the mouse button above it, the MouseUp event of the second (target) mouse will not be called.

- String variable used as the first parameter of the LoadText function will no longer load the contents of a second LoadText parameter. The variable used as a first parameter of LoadText can be now used as a name pointer to an object. This will allow you to dynamically replace the text in a series of text objects or buttons.
- String Arrays are finally unified (we hope)! All older scipts should work as previously, but it would be safer to check and fix them to match their syntax unified. Everywhere you need to use arrays use this syntax:
  Numeric arrays:
  NumVar[n]
  String Arrays:
  StrVar$[n]
  Object Arrays:
  TextBTN[n]
- TRUE/ FALSE are now keywords (automatically colored and converted to uppercase format).
- StrToFile predefined function now supports 1/0 or TRUE/FALSE in Append/Linefeed parameters StrToFile(filename$,string$, 1,0)
- <Temp> now returns path to Windows temp directory and <Embedded> returns path to MMB temp directory.
- **SongListSave** by default save internal SongList <List> (you no longer need to declare it in first parameter).
- **SearchForSong** is renamed to **SearchForFiles**. All older scripts are automatically updated to this new name.
- MMB now insert blank lines between the If..Else..End statement, when you type "If" into the script window.
- When you type a function to the script window and the function is auto-completed, then cursor now remains inside the parenthesis or quotes.
- **ViewJPG** function now support loading the BMP files and is renamed to **ViewImage** ;)
- Numbers and times in ListBox can now be disabled (in designer as well as in runtime by script).
- Improved fast clicking detection in hotspot object.
- **SongListTime** , CBK_Total, CBK_Time, CBK_VTotal, CBK_VTime now returns their values in hh:mm:ss format.

- Added "Search for ID tags" option to the ListBox properties. This option is useful in cases when a non-audio content of the <List> is loaded. Disabling this option can significantly increase the speed of loading.
- Better rescaling algorithm is now used in **ViewImage** function.
- Improved ListBoxAddItem parameters:

  Example:

```
str$= 'item 4#item 5#item 6#'
sep$=CHR(35)
param$=str$+','+sep$
** previously only this worked...
ListBoxAddItem("ListBox1","str$,#")
** but now you can use also this..
ListBoxAddItem("ListBox1","str$,sep$")
** or this
ListBoxAddItem("ListBox1","param$")
```

FIXES:

- In cooperation with **Bokzy,** we found a true reason of occasional crashing in MMB designer. The reason of this issue are some Delphi plugins. The good news is that it can be fixed, however, the bad news is that it will require a rebuild of most of all Delphi plugins. And as you may know or not, some of the plugins are no longer developed by their authors, developers disappeared and the source codes of these plugins are not available too. In short, some of the plugins cannot be fixed and they may still crash the MMB designer.

  Bokzy (as the author of Delphi SDK fix) will probably release (in a near future;) an improved Delphi SDK with description how to fix the current Delphi plugins.
- We also found (in cooperation with **mni**) a solution to another serious issue reported by some Win98/ME users. Some of them reports no sound or crashes with Flash, Video, Embedded or external applications or a general problems with internal MMB audio when running multiple instances of MMB applications at a time. Believe or not, the right reason of this issue are old sound card drivers! For more information about this issue and how to fix it check **this FAQ topic.**
- Fixed a serious memory leak if ObjectWidth was used with Text object, or looping ScreenWidth/Height/Colors and some other functions. If your project crashes from

time to time after and without any visible reason, this memory leak was the most probably cause of these crashes :)

- Fixed another potentially dangerous place in MMB code, which was cause of many strange and non-reproducible crashes and behaviors. They could happen in cases if your application jumps between mbd projects (via RunMBD) or between pages, and at the same time it runs some background scripts in loops (both finite or infinite).
- Each change of an mbd project should invoke the "project is changed" event and then mbd file should not be closed without Save request.
- Right click menu (Undo, Copy, Paste,...) in script editor is now back.
- Fixed Alpha button preview in the Alpha button load dialog.
- Fixed bug in **Run** command, which prevent correct parsing of the program parameter.
- EditBox variables are now correctly refreshed when they are filled via **LoadText** or **DisplayValue** functions.
- **DisplayValue** function can now fill the EditBox with a numeric value (not only Text object as previously).
- ObjectWidth / ObjectHeight now returns the correct numbers for Video object.
- Fixed a long time reported, but never found, the load bitmap crash..uff..it was really tremendous ;)
- Fixed some CBK audio related problems.
- Fixed a ListBox destroy problem after **RunMBD** use.
- Fixed a ListBox numbering problem. Sometimes happened, that the 10th item in the ListBox was numbered as 1.
- Fixed crash when trying to run an non-mbd file from the command line as a parameter of compiled MMB application.
- Fixed a refresh problem in a Dimension tool.
- Fixed CBK_Total a CBK_TotalSec for continuos CD playback.
- CBK text objects now work correctly if they are placed on Master Top page or Master Layer.
- CD stop should now work correctly with the audio CBK related variables.
- Fixed scroll bars in designer.
- Fixed some memory leaks in Flash and Text object;)
- Fixed a problem that cause 1+1=1 ;)

- Fixed bug in PlaySound and MIDIPlay that cause playing midi files in loop when the LOOP parameter was not turned ON.
- Fixed conflict between Matrix object and GetArrayItem function.
- Fixed bug in OpenFile command that prevents to open OpenFile dialog with a predefined path.
- Fixed bug in EditText object, that prevent to use array variables in the numerical output variable.
- Fixed bug in "Project Settings", that cause leaving an empty space at bottom of screen (the same size as taskbar).
- Fixed bug that prevents to replace MMB icon with custom one if project is compiled with NoFMOD player.
- Fixed an issue in **PlugiRun**, which causes reloading plugin into memory every time PluginRun has been used. This fix also significantly speed up the **PluginRun**.
- Speed optimization for non-overlapped objects. This is noticeable if the Flash, GIF, Dynamic FX or shadows are used in the same project window and they are not overlapped.
- Fixed Clipboard GET/SEND on Win2k/XP systems.
- Fixed some double-click highlight problems in Script wizard (Win98/ME related)
- Fixed [Tab] jumping over the EditBoxes. EditBox no longer remains highlighted when you select an EditBox by TAB and then you click into another EditBox by mouse.
- Project window can now be moved out of the visible area of the screen even with "Full Screen Background" option enabled.
- EditBox now doesn't allow variables/arrays with incomplete brackets..like this one: myTrickyString$1234**]**
- Fixed bug in <Embedded> macro. It finally returns path to MMB temp folder (<Temp>+\MMBPLayer)
- Fixed bug with saving/loading mbd project from 4.9.5 to 4.8 format and then back to 4.9.5.
- Fixed bug in "Compress&Export" which producing unreadable files.
- Again (and we hope for the last time ;) fixed a ListBox numbering problem. Sometimes happened, that the 10th item in the ListBox has been numbered as 1.
- Fixed CBK_Volume alignment.
- Fixed some Video mask problems.

- Fixed bug in ListBox and StrFromFile which fails on the first row if the rows starts with numbers.
- BackgroundPlay now works with AudioFX object.
- Pressing F1 no longer trying to start Windows Help for given project.
- **SongListEdit** can now be used in project compiled with NoFMOD player.
- Fixed refresh of Group objects placed at Master Page and Master Top Layer
- Numeric array can now be used in CHAR function:

  test[1]=10

  i=1

  c$=CHAR(test[i])
- Fixed "Path Replace" tool (menu Project >>> Path Replace...). Path Replace now works with all objects and scripts including the string variables.
- Fixed ORD function to work correctly with numbers > 128.
- Fixed "Failed...SetWindowPosition..." error if you try to scale video object without a loaded video.
- Fixed problem with outline shaper and right-bottom border of some images.
- Fixed some "Video in FullScreen" problems.
- Fixed ListBox bug with scrolling items without enabled scrollbars.
- Fixed ListBox bug with multi selection and moving items Up/Down.
- **MCICommand** again works with <This> macro **MCICommand**("open <SrcDir>\data\video_01.mpg alias MPEG style child parent <This>").
- **MCICommand** parameters are now correctly passed to Script Wizard dialog (after double click on command).
- Loading video between the MMB pages should work without the blinking. In fact, it still blink, but much faster now and not as noticeable ;)
- Fixed crash if project was compiled with NoFMOD player.
- Fixed problem with number rounding.
- Fixed Mask in Video object.
- Windowed objects (ListBox, HTML, Binding, ...) no longer grabs focus from the EditBox if Left/Right button is pressed.
- Fixed some Video OnLoad/OnStart event problems.
- Fixed bug in ListBox OnSelection event, which affects **ListBoxGetItems** and **ListBoxGetDelectedItems.**
- Fixed various problems with syntax highlighting (Win9x/ME).

- Numeric variables now support adding numbers and parsing predefined functions (with arrays) in one line

  sel=1

  selX=ObjectX(TextBTN[sel])+ 5
- Fixed IsMinimized function.
- Fixed MoveTo command (stop working when Type of movement was not entered).
- Fixed several String array problems.
- Fixed "Allow only one instance in Designer" bug.
- Fixed bug with Image scrollbars (when Image object is moved).
- Fixed bug with <Embedded> macro and plugins.
- Fixed some next syntax highlighting issues.
- Fixed **AudioOpen** vs. "Buffer is too small..." warning. Buffer is now increased from 4 to 16kb ;)
- Fixed memory leak in Designer when Custom Shape was used.
- Fixed **LoadText** trimming.
- Fixed artist and song title problem with '&' character.
- ListBox now insert numbers for all items (not only for audio files as previously). This can be disabled over ListBox properties dialog or **ListBoxParam** function.
- Fixed **DisplayValue** bug that cause wrong text alignment.
- Fixed a problem in Flash player "live" installation that cause non-working flash until application restart.
- Fixed **FMODconfig** initialization bug that prevents to disable FMOD on some Win9x/ME systems.
- Fixed some syntax highlighting glitches.
- Fixed a problem in **GetArrayItem** that prevents to use space and comma as delimiter. Now you can use hexadecimal codes of the given characters (e.g. **GetArrayNum**(test$,'0x20') where 0x20 means 'space' character). For more hexa codes check the **ASCII character table**.
- GetArrayNum and GetArrayItem now works with the string arrays:

  x = 2

  array$[x] = '1#2#'

  number$ = GetArrayItem(array[x]$,#, 2)

  LoadText("TextBTN","number$")
- Fixed bug that prevents to group the ListBox objects.

- Fixed bug in **ListBoxDeleteItem** that prevents to delete items from ListBox.
- Fixed some ListBox problems when ListBox is resized.
- **SongListSave** now correctly save internal list <List> (e.g. **SongListSave**("< List >",""))
- Fixed a bug that prevents to move Line object.
- Fixed "Go to next page" option in Video object properties.
- Fixed "Can't find plugin..." error message if plugin was loaded from an external directory (outside the MMB\Plugin folder).
- Fixed bug in "Allow only one instance" option, that prevents to run multiple different MMB exe's.
- Fixed a problem with disappearing mouse cursor after **ReplaceImage** command.
- Empty string used in **LoadText** function again clear the Object caption or String variable contents.
- Decimal separator is no longer dependent on a Regional and Language Settings.
- StrToFile now supports <> macros (<Temp>, <SrcDir>, ...)
- Fixed a bug in FMOD file type detection that cause a MMB to crash in some very special cases (loading non-standard Audio files).
- Scrollbars in Image object no longer remains visible, when the Image object is hidden (via Image object properties).
- Fixed rotation algorithm to give much more accurate results.
- Fixed a bug in StrDel function.
- Fixed Text object Alignment if the Antialias option is enabled.
- CBK_Month and CBK_Day now returns their values according the "Regional and Language Options".
- Fixed Edit box refresh.
- Video object cannot be run in full-screen mode from the first page of your publication by Page Start script. It can be run from the second page, but not from the first.
- Fixed FullScreen toggling via the Script object with the assigned shortcut.
- Fixed a bug in **BrowseForFolder** function (opening the multiple instances of the BrowseForFolder dialog).
- Fixed bug in a **ListBoxDeleteItem** function.
- Fixed a problem with WMA/ASF playback (right after the OGG playback).
- Fixed some script highlight issues.

- Fixed a bug with reseting UI.
- Fixed a Copy/Cut bug in the Script right clik menu.
- Fixed an object width bug when the "Default text" is deleted from the Input text.
- Fixed "Allow only one instance" bug on w2k computers.
- Fixed bug that prevents to change the Video object Width/Height after video load.
- Fixed crash when ProcFreq() predefined function was used.
- Fixed a problem when < SrcDir > and < SrcDrive > macros were used in path to Video in Video properties dialog.
- Fixed a MMB freeze when a predefined function was used in If ... Then statement.
- MVM file extension in VideoLoad dialog is now corrected to WMV file extension ;)
- Fixed bug in StrToFile predefined function that prevents parsing String$ parameter and adding empty lines to the output file if the append/linefeed parameters are TRUE.
- <> macros (e.g. < SrcDir >) can now be assigned to the string variables without the need to use quotes...for example test$=< SrcDir >
- Double click on a function now pass all parameters to the Script wizard. Previously, some parameters were not passed e.g. **ScriptTimer** ("TimerA=MoveTrackSlider1"," 50 ")
- "Fill With Color" in Bitmap properties dialog now works correctly when "Keep Aspect Ratio" option is used.

MMB 4.9.0.1(bug fixed 4.9 ;)

IMPROVEMENTS:

- Added new **SoundStop** command, which will stop all currently playing sounds, including the background music.
- Added two new optional flags to **Run** command. Now you can set the first window of an external application on TOP/TOPMOST over the MMB window.
- Compiled application should be a little smaller and should start noticeable faster than the same applications compiled in previous 4.9.
- In ListBox, you can now explicitly specify if you want to load the string variable as a string or in fact you wan't to open the file which path is in the string variable. Now if you are in doubt, use the command before a variable: e.g. STRING: variable.
  Example:

```
OpenFile("Audio Files (All Files|*.*||","*.*")
FileName$=CBK_OpenFile
```

ListBoxAddItem("ListBox1","STRING:FileName$") ** will add the selected file name as an item in ListBox

- Each parameter that should be sent to an application via RUN command can be enclosed in additional pair of quotes. There is a sample script: **Run**("mspaint.exe",""<SrcDir>\images\yourpic.bmp"") However, not all programs needs to have the parameters enclosed in the quotes.
- Added new confirmation dialog (in Check Project and Compile Output Files) for preventing accidental deletion of already compiled files.

FIXES:

- Fixed bug if you tried to add a custom icon into the project.
- Fixed bug that prevents to play the embedded audio files.
- **AudioPause** and **AudioOpen** commands now work the same way as 4.8.x.
- Fixed bug that erased the source mbd file if you compiled the final exe into the same directory and with the same name as source mbd.
- Fixed bug that prevents to play the ogg file in background and wav on MouseUp/Down effects at the same time.
- Fixed problem with Enhanced Script window that prevents to save the script changes in some special cases (when pinned) .
- Fixed problem with custom font on ListBox object
- Fixed some CBK variables that don't work if the CBK labels are placed on Master Top page.
- Fixed some next CBK variable bugs when working with Audio CD (CBK_Time, CBK_Total, CBK_TotalSec).
- Fixed problem with additional backslash in m3u Winamp playlist files. Some older versions of Winamp add the backslash in front of the relative path.
- Removed an FMOD error message that appears on the system without an installed sound card.
- Removed rclick "Close" menu from external player.
- Fixed a RunMBD bug that prevents to load the external mbd files in some special cases.
- Fixed bug in embedded "Flash Player" installation.
- Embed Flash Runtime option is now connected between the Flash objects used in a project.
- Fixed bug in Copyrights Info, that sometimes saved the Mediachance related

information instead of the developer's info.

- Fixed bug that prevents to change the video file in a Video object.
- AudioVisualization object now refreshing its content only if supported music is playing.
- Fixed bug that change the name of object when you copy objects between the pages.
- CBK_AudioName now returns its content case sensitive.
- Fixed bug that prevents return the correct CBK_Total and CBK_TotalSec variables when the last CD track was played.
- Fixed support for embedded s3m, xm, it and rmi audio formats.

MMB 4.9 - ADDITIONS AND REPLACEMENTS

FMOD audio library:
MMB Audio support is now completely rewritten with the latest **FMOD** audio library (**www.fmod.org**).

What it means for you?

- No more crashes or freezes when playing MOD or variable bit rate Audio songs!
- Better CPU utilization with only the minimum system requirements.
- Added more new audio formats (OGG, WMA, ASF, XM, S3M, IT...).
- Great playback quality.
- With this library can be (and in a near future will be) implemented many new audio tools for your fun and for enhancing the MMB possibilities.

Macromedia Flash object:
It will allow you not only to load and play Macromedia Flash animations, but with this object, you can control all the MMB scripting actions directly from the **Macromedia Flash Action Script**! Therefore, you can use your own Flash file as a powerful scripting addition to the basic MMB scripting – e.g. for parsing strings, advanced math, adding some new UI elements, like a check/option boxes, combo boxes, spinning objects, tables and charts, etc.

List Box object:
The one of the most requested features is now available in MMB! This object is primarily designed for showing the audio playlists (winamp *.**pls**, *.**m3u** and MMB *.**m3l** formats), but it can be used as well as a usual list box control with **multi-selection**. You can fill this object with text, string variable, string array, text files or mentioned playlists. This new type of object also bringing new type of event to the MMB – double click (at the moment only for list box object). In addition, there are some new scripting commands, allowing you to effectively work with the list box object.

Audio Visualization object:
With this control, you can visualize the playback of OGG, WAV, XM and S3M formats. At the moment, there are two types of audio visualization objects - **oscilloscope** and **equalizer**. Of course, you can change the type of audio visualization object or change

the curve and background color in a runtime (for example after click on an object, you can change the type of AV object from oscilloscope to equalizer - ala Winamp).

OTHER ENHANCEMENTS:

- Double click on a script line cause opening the script wizard dialog. Editing scripts is now **interactive** and **fun!**
- More Actions window (MouseUp/MouseDown) is now redesigned to **tab controls** - **STOP** editing the scripts in a tiny edit box, with only three visible lines;-)
- Enhanced (resizable) script window is now accessible for all events and script objects.
- Scripts in More Actions window now have the color **syntax highlighting** (as enhanced script window).
- Added new floating window with a list of active variables (in a debug mode), for easier project debugging.
- From this version, you can define your own FileVersion (Copyright, Company Name, Description, File Version, etc.) information stored in the compiled file. Many other authoring tools didn't allow you to do this!
- HTML, Binding, MCI, Flash and ListBox objects can be overlapped with the other objects.
- HTML, Binding, Flash and ListBox can be resized by **MoveObject** command (with the optional Width/Height parameters). The binded application must support resizing its window.
- Allowed multi-selection in AudioOpen dialog. Selected items will be automatically stored it the internal song list <List>.
- Reorganized and resorted list of scripting commands in Script Wizard dialog.
- CBK variables can be now added and accessed from Master Page or Master Top Layer.
- Redone MMB help to CHM format - some information were supplemented, added

some completely new and fixed some wrong. However, there is still a lot of to do;-)

SCRIPTING ENHANCEMENTS:

- All CBK Objects are now scriptable. You can now assign them to a String or Integer variables.
- **SongListLoad** now supports Winamp native playlists ***.m3u** and ***.pls.**
- Internal playlist <List> enhancements:
  - <List > now works with ***.ogg**, ***.wma** and ***.asf** file formats.
  - Can also load the external playlists from Winamp - ***.m3u, *.pls.**
  - Now you can Save/Load or Delete items from <List> directly by new scripting commands.
- OpenFile command now returns not only the full path to the selected file, but also the "Directory" and "Filename" splitted into two new CBK variables.
- New double click event (currently available for list box only).
- Added some new predefined functions:
  - GETARRAYITEM function to return an item from a string variable with predefined delimiter and GETARRANUM function to return the number of items in an Array variable.
  - OBJECTWIDTH and OBJECTHEIGHT - functions to getting the objects width/ height:-)
- Added few new CBK variables:
  - CBK_TotalSec - Returns the song's total time in seconds.
  - CBK_TimeSec - Returns the current time of playing song in seconds.
  - CBK_Volume - Returns the current volume level.
  - CBK_NumTracks - Returns the number of tracks on CD.
  - CBK_OpenDir - Returns the file name string from the file selected in OpenFile

dialog.

- CBK_OpenFile - Returns the path (without the file name) string from the file selected in OpenFile dialog.

- Added many new functions to work with **ListBox**, **Flash** and **Audio Visualization** objects.
- Added new **AudioRewind** command.
- Added new **CDSkipForward** and **CDSkipBackwards** commands
- **AudioOpen** command now supports the multi-selection. Selected files are automatically added into the internal playlist <List>
- Added new **SongListDel** to delete song file from the internal list.
- Added new **SongListSave** to save internal list <List> or ListBox content instead of internal List.
- If you use an empty string in **ModOpen**, **MidiPlay**, **WavePlay** and **P laySound** commands, then an Open Audio dialog with predefined mask is opening.
- If you add an unknown or incorrectly formatted command to the script, then script line will be commented instead of deleting the wrong line, including the rest of the code below (as previously).
- **LoadText** command: added a \n parameter to split the long strings into the multiple lines.

FIXES:

- Fixed MMB freeze when you copy&paste something from MMB help to the script window, or if the clipboard contains some special characters - ? z · © ®
- Fixed MOD crashes or freeze (by replacing the audio library;-)
- Fixed crash when you try to go on a page that doesn't exist.
- Fixed wrong refresh when the Text object has a "Fixed Width" option unchecked and the text content is dynamically changed.

- Fixed some refreshes on HTML object, especially if you try to use the transition effects between the pages with HTML objects.
- Fixed a CBK_MENU event when you try to Invert the menu state.
- Fixed IF_IDLE optional command in PageTimer action.
- Fixed some inconsistencies between the Designer and Player.
- Fixed a serious problem that might cause the raising CPU or memory and probably slowing-down the compiled MMB projects.
- Fixed an "Out of Memory" problem when RunMBD command is used.
- Some next small fixes and improvements.

MMB 4.8

- **Fully customizable interface.**
  The new version of 4.8 MMB designer GUI is changed to the latest trends with full toolbars, menu, and keyboard customization. Now you can customize the designer anyway you like. You can practically customize any important aspect of the designer interface, including:
  - **Drag and drop buttons** between toolbars and menus
  - **User-defined** image editing of buttons (simple bitmap editor is included!)
  - Ability to create a **new**, empty toolbar that you can add buttons and menus to
  - **Context menu** customization
  - "**Alt+drag**" customization
  - **Keyboard shortcut** key assignments customization to any command
  - MMB will **remember** the new position of any toolbars, menus or page/ object list and all your customization, even menu can float or be docked to any side.
  - **User-defined tools** - you can add additional application shortcuts to the MMB Tools menu
  - **Smart expanding Menus** shows basic and frequently used commands on

personalized versions of menus. The menu learns by itself which commands you use the most.

- **Tear-off menus**: Submenus can be teared-off so they can float as a toolbar:
- Ability to **change look (skin)**, currently build-in themes: Standard, Windows XP, MAC look and Gradient.
- Change other aspects such menu animation, shadow etc..

- Added shortcut to the object actions (menu Object-Action) - you can tear-off this menu and use it as a toolbar or move the buttons anywhere you like.
- A new Rulers for a workplace
- Guides with snap to Guides feature. To add guide you drag it from the rulers, if you want to delete guide, drag it outside window. You can also Add/Edit the guides by numbers as well (right click on rulers)
- Support for updatable MEF file. This will work with Real-DRAW - you are able not only to load the design but also update it later.
- Mostly designer enhancements:
  - **Smaller fonts** in Object list, more objects are visible in the list - Is it too small or it is OK?
  - By clicking on the new list button you switch between Full object list and Separated object list.
  - In separated list the objects are divided and sorted into 5 list groups : Graphics, Text, Buttons/HotSpot, Script and Misc. This will help to better navigate in large project. Members of Grouped objects are displayed with an arrow on front: . The rest works like normal list, you can select multiple objects with Shift also across the list groups.
  - Script object display a name of the script on the project desktop
  - **ReplaceImage** for Hotspots as well
  - **Binding Object** - Added option how to close (Un-Bind) the program. Two way:- Terminate Process-default (brute force method which doesn't give the program chance to do anything about it)- Send Close and wait. This is softer method and it gives the application a chance to do closing stuff (for example ask if you want to save changes etc...) Also if application doesn't like the first method, try this one.
  - **Binding object** - advanced option, added another Caption string: Caption must NOT have string... Along with Must Have string, this is your chance if the app you binding has many windows and MMB can't decide which one to bind and bind the wrong one. You can specify which string the main caption must have and which it must not have. If empty - then it doesn't apply.

- HTML Object **Disable right-click menu** option. The browser control will not display its context menu after selecting this option.
- HTML Object: **Hide Border** Option and **Always Hide Vertical Scroll Bar** Option. This can create a web page which is seamless with your MMB page design with no border and no vertical scroll bar. (The horizontal scroll bar is created dynamically if the web page you displaying require larger width (if you use tables or frames for example).
  Note: if you use Hide Vertical Scroll Bar it will always hide the scrollbar. Even if the page is longer than the HTML object height user won't be able to vertical scroll (only with Microsoft Scroll wheel)
- Binding Object - **Un-Bind on Page Exit** option (default). If checked the Binding exe file will be terminated if user goes into other page. That means any time you go to the page with binding object the exe will start again. This is probably great for applications as Flash animations etc... you don't want them to run if you are in other pages.Unchecked - once the binding object is created it is not terminated on page exit but hidden. That means any time you go to the page with binding object, the object will stay the same as you left it last time.
- If you apply a Show(...) script command to an EditBox which is already visible it will put a focus in it and select all the text.
- If you hide binding object the screen is updated properly.
- The HTML link page: now works also with transitions.
- **SendCommand** script command to send Menu Commands to the binding Object. SendCommand("Object","a,b") where a, b are the indices of the Menu item you want to run based on zero. See detailed info in **binding.mbd** project.
- **Clipboard** script command to send/ receive a string to/from clipboard.Clipboard ("SEND/GET","stringvariable$")
- Experimental: Project-General Settings-Style, new style Windows Desktop Component. This will make the player snap to the background - it will stay always on the windows desktop, emulating the behavior of Active Desktop component, however without any need to have Active Desktop enabled or even installed. Unfortunately it can't be combined with Full Screen Background option so if you want to perhaps cover whole desktop you have to make the project non movable and resize the player with script. Example of resizing the player to full background:
  SysCommand("MoveWindow","0,0")
  SysCommand("ResizeWindow","ScreenWidth,ScreenHeight")
  This option is experimental - it may be removed from release if it cause problems. Also during testing from designer, MMBuilder will hide to uncover the background - don't be surprised then.
- Please test if the player correspond properly to the internal player.
- Keyboard input fix for HTML object. It should also fix all the keyboard problems related to Input Boxes and other windows (such a plugins), input boxes gain and lose focus properly now
- You can move between Input boxes with TAB key, the Tab order is the same as layout order
- The Variable associated with Input Box will be from now automatically initialized with the default text. You don't need to separate initialize the variable any more. This is more logical and easier. (You can still override the variable in Page Start

script)
- Smarted LoadText.
- Load custom cursors - normal or animated. The cursor dialog in page, text button, bitmap button and hotspot has now 5 Custom cursor location. If you select unassigned custom cursor it will prompt you to load either *cur or *ani file. These cursors will be auto-loaded into Embedded files. So if you don't want the cursor anymore, you can delete it from there.
- **HTML Object**

  Html Object is a full Browser object in MMB. It uses Microsoft IE control. - example **html_browser.mbd**
- Hopefully a final fix for Tooltip controls for Objects
- Binding object can be hidden or moved with Move command

- **Introducing Binding Object**

  Binding Object is a way how to easily put exe file into MMB - it will become a part of the player: for example flash movie with flash projector player, your installer, notepad with text, another indenpendednt mmb project etc...See example binding.mbd included in the beta archive, shows notepad, regedit and plays a flash movie inside page.Tested on Win2000 and Win 98SENote: Not all exe files can be binded, but many I tested could (for example you can even bind CompactDraw or PhotoBrush....) Some exe files may be unhapy to be binded and may freeze your system - save often. Some exe files can't be binded at all or shws garbage (Winamp).You can't call just the file to open associated exe yet. you need to specify the executable and then the parameter (file). You can hide Menu , status bar or toolbars from the binded applications.

Fixes:

- If you typed a space in the edit box, it displayed 2
- While in long loop, you couldn't interact with edit box even that Refresh()

  command was used
- In LoadText, you can now explicitly specify if you want to load the string variable as a string or in fact you wan't to open the file which path is in the string variable. Now if you are in doubt, use the command before variable:STRING:variable or FILE:variableAlso if the file doesn't exist the LoadText will display it as a string. Example:
  path$ = '<SrcDir>\myfile.txt
  LoadText("Text","FILE:path$") ** will load the text from a file specified in the path $
  LoadText("Text","STRING:path$") ** will load the text from the path$ directly - in our case the Text will show the <SrcDir>\myfile.txt
- You can Minimize now while in full back mode.
- When executing web link using Netscape 6, MMB displayed "file not found error"

  while the page was loaded ok. Displaying of this error was disabled, even that it is

  legitimate.

- **Generate Autorun.inf** option added into Check & Distribute - so now you can switch off generating the inf file if you don't need it.
- **Autorun.inf** is now finally generated with the filename of the exe instead of default autorun.exe
- The **Change Resolution** now don't allow up-sizing by default, only down-sizing. You can change this with Allow Up-Sizing, but you will be warned that this isn't a good idea. If users have low resolution set on the display, they have it for a good reason so we shouldn't try to up-size it by default - this will solve many problems with older hardware.
- Solving some display compatibility issues with Transitions and Dynamic FX.

MMB 4.7 and earlier

- E-Card. In this version we make sure the E-card is not dependable on any system files. E-Card is a downsized player, good for creating small presentations to be sent by e-mail. With E-card you can't use mod.
- **OBJECTX(Object label)** and **OBJECTY(Object label)** functions added in the script.
  These functions return the current position (left, top corner) of the object specified by the label.
  Example x= OBJECTX(Text1)
- **ISVISIBLE(Object label)** function was added
  This returns
  -1 if object doesn't exist on the page,
  0 if object is hiden
  1 if object is visible
- **OpenFile$** variable was added. This variable will have the path after using OpenFile script command.
- **VolumeUp** can now take a parameter - the volume from 0-100
  Example: VolumeUp("50") will set volume to half
- **ReplaceImage**can load BMP beside the JPG
- **MOUSEX()** and **MOUSEY()** functions added in the script.These functions return the current position of mouse cursor.
- Added Default cursor listbox into the Page Properties.
  The last item there is a dot cursor - great for touch screens.
- **Plug-In's loading code**
  Plugin's are now loaded before the window and they don't require to be visible.
- **MoveObject("Object","x,y,w,h")**
  Move (and/or resize) the object (or group) to the x,y position. The w and h parameters are not required and if they are defined the object is resized - However it wont resize any bitmap object or text - it resize only its active boundaries. But it works fine for rectangle, buttons, hotspots...etc..
  Yu can make a code:

```
for i=0 to 100
MoveObject("Bitmap","i,20")
Refresh()
Pause("30")
```

next i
**and it will move the object form left to right!

- **MoveTo("Object","x,y,steps,type")**
  This simple command moves object (or group) from current position to the x,y using # of steps. The type can be EASYTO, EASYFROM or none to define the linearity in time. It basically does the same as code above, however you can specify slowing or accelerating with the TYPE.
- **Refresh() command**
  This command will force to redraw the changes. It also works like a pump, if you use Refresh() in a loop the program reminds active and in fact you can still interact with the active objects (while the loop is continuing).
- **Pause("ms") command**
  Pause command in the script, takes a number of ms to pause before continuing the script
- **CBK_Time and CBK_Total** can be addressed as an integer variables
  Example: a=CBK_Total
  This always returns the number of seconds (equivalent to what would be displayed in these objects.
- **Ability to insert Contents Copyright into the project executable**
  This allows you to insert your text in the Comments field of executable properties. For example you can put your copyright for the contents you created or comments about used materials. You can put up to 60 characters. That doesn't replace the other MMB player properties, but it resolves some legal issues if your custommer doesn't want to put your copyright on the project pages but you still want to have your copyright or name there. Many other authoring tools didn't allow you to do this. You can access this option in the Check & Distribute dialog box.
- **Dynamic Changes of Display Resolution**
  You can tell the project before the start to change (or at least to try) the Display Resolution to one of few typical resolutions. You can set this option in the Project-General Settings. You can make a project which will play always on a full screen. Note: Some video cards (very old ones) don't allow to dynamically change the resolution so the project will continue in its standard size. The most common sizes are 640 x 480 and 800 x 600 and should work on most of the cards. The resolution will be returned to the original settings after you exit the project. Checking this option will uncheck "Standard Window" and "Movable" from the default values - because it makes little or no sense, however you can turn them back on if you making a "special" project.
- **Control over Process Priority**
  This new version allows you to control the priority of the process. You can change the priority in the Project-General Settings. If you select **High**, the application you create will steal more CPU for itself from other windows application, making the transitions and FX running smoother. This is great for presentation type of applications where you don't expect user to be often switching between applications. The **Normal** settings is good for most of the application (it is the default setting) and the **Low** priority setting can be used for special (background) application which shouldn't use much CPU (launch bars etc..)
- **Masking Video**
  In 4.6 you can use B/W mask for Video Masking. With the Mask, video doesn't have to play in rectangular square anymore and it can have any shape you want.

  You need to prepare an image mask with size of the video. The video will be played through this mask where the black pixels will be video and white

transparent. The mask could have any shape or image (for example black text on white background). You can load the mask in the Video Properties. (Two buttons were added - Load and Clear Mask) You can create a very interesting effects because finally you don't have to be stuck with the boring rectangular video found in many authoring

- **MCI object was added**
  This makes easier to play video or audio using MCI without doing much scripting. Let's say you want to play ASF so instead of bunch of messy MCICommand script lines you simply draw a MCI object which could play your asf automatically on page start or you can control basic functions (play, stop, close) with new script command MCIObject.
- **3 additional script timers were added**
  TimerA, TimerB and TimerC were added to the ScriptTimer command, making that four timers for script. You can use ScriptTimer command as before - then it uses the standard timer or you can use syntax:
  ScriptTimer("TimerA=Script","500")
  (You can use TimerB and TimerC the same way) These are independent timers. The same rules apply for them like for the standard timer: After the script is executed the attached timer is killed as well as if you go to another page.
- **The Secure Layer**

  In the Check and Distribute a **Secure layer** option was added.This will process the mbd data in such way that no Text or Script would be visible if you look at it in the Hex editor. Also the file can't be loaded back to the designer (no password will help!) so it can only be played. However, note that the loading of file with Secure layer would need more time and memory that without this layer so you should use it only on necessary files. The same option was added to Compress and Export to easy create mbd files which can't be viewed in Hex editors and loaded in designer.

Fixes in 4.6

- Script from On Move cursor fixed
- String arrays now works in designer and in the player the same way:
  toon$[0]='O'
  toon$[1]='k'
  string$ = toon$[1] + toon$[2]
- You can add backslash at the end of a string by using \\
  path$ = 'c:\mydir\\'
  will be displayed as **c:\mydir**
  There was a conflict because in 4.5 we used \' to enter ' into the string:
  string$ = 'That\'s great!'
  which will be displayed as **That's great!** and there was no way to add backslash at the end of the string then. Now you just use double backslash to ad a backslash at the end of a string. Complicated? Hope not that much!
- You can now exit from th For - Next loop by using Return() command
- The Path Replace, the Checking for Fixed paths in Check & Distribute and the Text Replace tools were update for new changes.

MMB 4.5

- Active Objects can have **Tool Tips**. That is a text you can see on runtime when

you move mouse over some object and wait. You can set the Tooltip simply by writing the text in the Tooltip entry of the object properties. Very easy way how to add professional touch to your application.
The objects supporting the Tooltip are: Text Button, Bitmap Button, Text EditBox and Hot Spot

- Enhanced **Object list**. It supports multiple selection. Whenever you select more objects on the workspace they will be highlighted in the Object list. In addition to this you can select multiple objects directly in the list by holding Shift and you can also unselect the object in the list or on the workspace the same way.
- **Snap to Grid** was added. You can switch it on by clicking on the Snap To Grid button on the toolbar. The Grid settings are under menu Settings. You can change both X and Y grid. When the Snap to Grid is active the grid marks are visible and you can place the objects by mouse only on these marks. You can still use arrow keyboard keys to move the object by pixels..
- **Quick Object** Roll-Up was added You can add a Bitmap, Paragraph Text, Bitmap button or MMB Object just by double-clicking on the file in the Quick Object file list.
  This is very useful if you have many graphic or text objects on the disk. The same way you can import all supported images, Bitmap buttons (BTW) or exported Objects (OBM).
  The Roll-Up window can be minimized and maximized by clicking on the white rectangle or double-clicking on the top handle.
- **Dimensions** Roll-Up window was added You can change the position of objects by entering a number.
- Pages in the project can be easy repositioned and copied with new **Page Manager**
- Animated Gifs reviewed - better support for optimized animated gifs with disposal methods. Enhanced Animated Gif properties dialog box. Run Script after last frame added. Load New file button added.
- In Paragraph text properties switching between enhanced and international editor added. Also a new button to load text from the file into the editor.
- In the page actions Go To Page was added a list box for pages just to make it easier to use.
- In Video Object added Run Script after video Stop/Finish
- Added arrays to the string variables in the Script (a$[1] = b$[a+1])
- The Text Buttons and Bitmap Buttons now change back to default image if you click on a button and drag cursor off the window before lifting it
- Looping of Playlist. Whenever the last item of the playlist have instead of audio file word **LOOP** then the playlist will loop.
- Fixed the Video & EditBox problem - after video stopped the edit boxes were no

longer receiving input.

- Video & the Go to Next Page on FW & BW problem fixed
- Bonus: Color Tweak effects 30 new bitmap color, art and special effects.

MMB 4.4

- Text Button can have customized font
- Ability to change the default cursor on most active objects - ten predefined cursors
- Full Script editor enhancements - color highlighting keywords, wizard, remembering the size and position
  New Script Editor
- Paragraph text editor enhancements (Drag and drop text)
- Audio can be also a command line parameter in the player (So now you can associate your MMB Audio player with audio files)
- Ability to Resize Page, Move Page, Stay On Top from the script
  (See Syscommand.mbd)
- Ability to copy file and to create a full path directory.
  (See Syscommand.mbd)
- Ability to Show or Hide Objects on any page from any page (Master pages included) from script (See updated **Masterpages.mbd**)
- Ability to run designer directly from CD without installing it on hard drive.
- You can use ' in string: **Don't panic** you would write with backslash:
  a$ = 'Don\'t panic'
- Smarter Copy-Paste Groups
- Added <Windows> and <System> in the Script
- Bonus Color Tweak Effects with 30 new color,art and special effects

---

MMB 4.3

- CBT is here ! Full String support in script
- Input Box Object
- Change icon in the compiled application
- Get info from MCICommand

- Video Object enhancements - actions on Video finish

MMB 4.2

- Audio enhancements
- ID3 Tags
- Audio List Enhancements
- Replace Text, global function

MMB 4.1

- MMB Plug-In support
- Templates
- Load text from external file

MMB 4.0

- New Transitions, Page Curl, Fire
- 14 cool new Bitmap Effects (menu Effects),Water, Impresionists, Warp, Bump....
- Matrix Object
- VR Panorama Object
- Dynamic FX

MMB 3.3

- Page Transitions
- Embedded MIDI and MOD
- Embedded external files
- **Animated Gif**
  New object - animated gif was added. It has its own properties which allows you to control the gif transparency, background color, speed, auto play, loop..
  For sample see Ronnie's demo **agif.mbd** on the users gallery or the included simple **animation.mbd**.
- **Embedded Wave**
  Yes you can embedded wave into mbd file. Use it for small click sounds and you will be happy.
- **Script Object - Keyboard Shortcuts**
  New Object was added. This object can be used for your global functions or any functions. It has also feature to run that script if user press some keyboard shortcut. See example keys.mbd
- **Full Screen Background**
  This will cover all windows with solid fill, image, tiled image or stretched image. Great for kiosks. See examples: kiosk.mbd and fullback.mbd
- **MIDI**
  The script now has MidiPlay and MidiStop commands
- **Size**
  If you compile the stand alone file you will see it is arround 300 kB smaller than in version 2.0. This allows you to put the file on the old floppy disk, so you can distribute your presentation on that medium.
- **For -next loop**
  Added to the script. The syntax is typical Basic syntax:
  for a=1 to 5

...
next a

- **Password**
  Now you can protect your mbd files from editing by putting password. Go to menu File- Compress & Export and there is a place to put the password. You cannot protect the files you are saving with Save or Save As commands, you can protect only files you are exporting.
- **Other**
  Move 10 pixels - If you use Keyboard Arrows, the object will move 1 pixel, but if you use **CTRL+arrow** the object will move 10 pixels.
  Interaction with other objects - the "Moving Mouse over the object" has more actions in the combo-box: **Show only**, **Hide only** and **Run script**. The first is great if you want to show Animated gif if mouse is over some object - the animated gif can hide itself after last frame (See Animated Gif properties). With the other commands you can do some other magic on mouse move.
  CBK_AudioEOF - The script object with this name will run when Audio song reach its end... for more see sample **mp3list2.mbd** in the package.

MMB 3.1

- Print Text Function
- Page Import/Export
- Bitmap Button has new option 'Auto-Button'
- This allows you to create bitmap button only from one Image. The button will have look and feel like Text Button but with a bitmap.
- Video Object has the option to load MPEG, VideoDisc or MOV - using MCI - this saves some writing in script for MCI.
- OctaMed sound support (*.med). Thsi is the first from upcoming mod formats.
- OpenFile command and internal <File> added. This allows you to pop-up Open File dialog. For Example:

```
OpenFile("MPEG files |*.mpg||","*.mpg")
```

Then you can use the <File> (the same way like you were using <SrcDir>). So for example for MCI you can use:

```
MCICommand("play <File>")
```

- VolumeUp and VolumeDown commands were added
- Custom Shape window: The B/W Mask option was added allowing you to create any shape even with holes
- The candy See Through was added. This will turn the background of project transparent and then you can cast shadow on desktop or make fully semitransparent window. However it has some limitation.
- Cover Windows taks bar in Full Screen background option.

MMB 3.0

- Animated Gif support.
- Embedded Waves.
- Midi.
- MCI commands (allows you to play MPEG or video disc.)
- For-next loop.

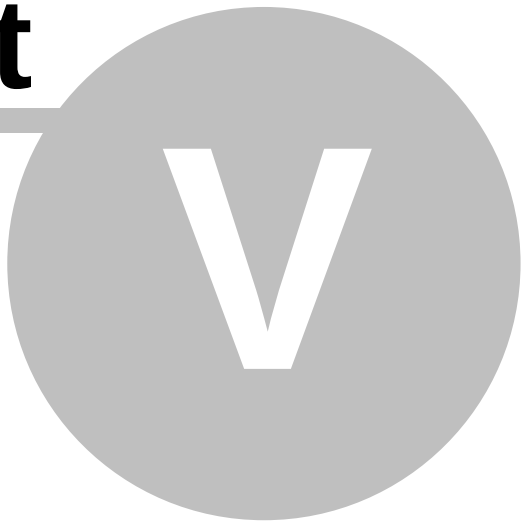
MMB 2.0

- Full Audio support with feedback.
- Script language with variables.
- Timer functions (NextPageafter, ExitAfter).

Top Level Intro

This page is printed before a new top-level chapter starts

Part



5 List of MMB Features

Here are just few major features. When working with MMB, you will see that the list of all features would be many pages long. In fact, you might never see a similar Windows applications with so many features for this price!

- WYSIWYG design
- Object Oriented environment
- Easy interface
- Click-to-create objects and actions
- Support of all popular graphic formats
- MMB supports Macromedia Flash files and can control your MMB application directly from Flash
- Compatible with Macromedia Fireworks using the same PNG format
- Multiple Undo/Redo
- Grouping and Ungrouping objects, nested grouping
- Alignment tools
- Blending graphics with background
- Alpha Transparency Masks for blending
- 24-bit color
- Create Stand-alone exe applications
- Optimized performance
- Real Glow and Drop Shadow
- Effects: Sharpen, Blur, Flip
- Special effects: Fire, Cutout, Bevel
- More than 40 bitmap effects and filters
- Background Bitmap Tiling
- Background sound can play across page boundaries, looping
- Objects are in layers

- Define your own graphic buttons, save, load to/from library
- Window doesn’t have to be rectangular - support for custom shapes
- You can directly paste graphics from your graphic editor into the MMB without saving
- Video can be played on any speed
- Fade out effect
- Wizard for objects and actions
- Support for CD audio and Mixed-mode CD's
- Stand-alone linker
- Script language for more power, variables, string variables, timers
- Animated Gif
- Embedded Waves
- Midi
- Bargain price

Top Level Intro

This page is printed before a new
top-level chapter starts

Part

6 Overview

6.1 Project Settings

 Menu Project - General Settings

You can configure different settings according to your application: kiosk, window, toolbar, etc.

- **Window Size**

  Shows the Width and Height of the application window.

- **Window Title**

  The title (caption) appears at the top of the Standard Window.

- **Standard Window**

  If checked, the window of your application will be a standard window – with caption, border and close button. If unchecked, the window will be a custom window – no border, caption or close button are displayed. Also, you can create and use your own artwork.

- **Enable Maximize**

  Enables the **Maximize** button in application (standard window) title bar. Clicking this button maximizes the application window. Afterwards (if required) you have to resize/move objects placed on the project window (via script).

- **Enable Minimize**

  Enables the **Minimize** button in application (standard window) title bar. Clicking this button minimizes the application window to Windows task bar.

- **Enable Size**

  Enables the **sizing** border around the application (standard window) window. Dragging the application border resizes the application window. Afterwards (if required) you have to resize/move all objects placed on the project window (via script).

- **Client Border**

  A thin black border will be drawn in the window's client area.
  Very good if you need to add a thin border to the window without any border (unchecked Standard Window).

- **Custom Shape Window**

  This **unique feature** moves MMB's Non Standard window even further.
  The window can have a custom shape – taken from a library or an image.

Trace Shape from Image.
To work properly, the image must be on solid background with enough space around.
Specify the start point (where the tracer will start.) It should be in the background of the image.
The tracer tolerance allows you to specify a closer shape. "0" value: no tolerance.
For maximum effect, create your background artwork on a solid background. Then, create the shape with the tracer, and use the same image for the background (in **Page Properties**)

B/W Mask
You can use another option – B/W mask. The white part will be invisible (transparent) and the black color will become a window. With the B/W Mask you can create windows with custom shape with holes.

This is the difference between B/W Mask and the Tracer. Tracer only outline the picture. B/W Mask allows you to create any shape you like. (The same way like WinAmp is using Skins.)

- **If 256 colors detected**

With the runtime, if MMB detects 256 color it can run another page (not the first one), run another file, or just continue.
On the separate page (or file), optimize graphics to display 256 colors.

- **Palette**

Note: Only for 256 colors.

Graphics MMB
Optimized palette for most full-color pictures. Use this palette when creating pictures or images. Extract the palette in the Palette directory (**palette.bmp**) using your graphic editor, and then apply it on all images.

Windows Standard
Windows standard palette is recognized by all graphics applications.
Use it whenever your projects deal with screen captures.

- **Style**

Always on top
Put the active window on the top of all windows – topmost window.

Save Last position in registry
Applications generated by MMB, remember its last position, which is stored on registry under a specified name. The next time you run it, it will open in the same position.
Remember to use a different name in the edit box in order to store the position of your applications. Otherwise, they will share the same position.
Tip: Use it for toolbars and launchers applications.

- **Background Mode**

Use this mode to hide the desktop with a specific background.
Tip: Use it to create application for kiosk or exhibition stands.
It's also great for creating **Autorun Browsers** where the background uses an image in relation to the project.

In Background Mode, chose Solid Fill (select the color), or use an image (load it with load button.)
Then, specify how it will be displayed:
***Normal*:** Top-left corner of the screen.
Tile: Tile the chosen image.
Stretch: Stretch image to full screen.
Tip: The last option allows you to create very interesting presentations.

Chose pictures with patterns, which could be stretched nicely. Don't use small pictures, pictures with text.
Use a standard display ratio like 4:3 – for example, 640 x 480 is adequate for most screens.
Corel products installation offers some nice examples.

- **Disable Alt-Tab in Win95**

Great for kiosks where you don't want a user to switch from one app to another.
It also disables **Ctrl-Alt-Del,** and other windows commands.
It doesn't disable **Esc**. You have to handle it otherwise.
Tip: To disable Esc, use CBK_EXIT (for more info see **CBK objects**).

Display Resolution
Force the display driver to set new resolution. This is great if you want to have project on full screen. However not all adapters will allow to change the resolution - in that case MMB will stay in the window.

Allow Up-Size

Normally you should leave this un-checked. By default we don't want to up-size low resolution display - it is a big chance that if user has low resolution he has old hardware. Use this only if you are certain who will view the file. If un-checked, MMB will only down-size the display (which should work on all systems)

Remember that reasonable forced resolution is 800x600 or lower.

Set Process Priority
If you select **High**, the application you create will steal more CPU for itself from other windows application, making the transitions and FX running smoother. This is great for presentation type of applications where you don't expect user to be often switching

between applications. The **Normal** settings is good for most of the application (it is the default setting) and the **Low** priority setting can be used for special (background) application which shouldn't use much CPU (launch bars etc..)

Real examples summary:

Option	Kiosk	Corel Install	MS Autorun	Toolbar
Standard Window	NO	YES	YES	YES/NO
Client Border	YES	NO	NO	NO
Movable	NO	YES	YES	YES
Always on top	YES	NO	NO	YES
Save Last Position	NO	NO	NO	YES
Full screen Background	YES	YES	NO	NO
Disable Alt-Tab	YES	NO	NO	NO

6.2 Getting Started - "Hello World" project

New project in 10 easy steps ;)

1. At the menu select: **Project** and then **Project Settings** 

2. Enter a size for the display area of your project in **Window Size** - for example enter 320 x 200

3. Enter your own **Window Title -** e.g. enter "***Hello World***" or whatever you want. Click **OK** to finish Project settings**.**

4. Click on **Text**  button and click somewhere in project window.

5. Double click on newly added text object and enter "***Hello World***" in text field.

6. Now click on **Script** icon  and insert `NextPage()` command to **MouseUp** script event.

7. Close both script and text properties dialog with **OK** button.

8. Click on **Add a New Page**  button to add new page. The page properties can be changed via dialog invoked by double clicking the selected page in **Page List**.

9. Repeat steps 4-7 but instead of "Hello World" enter for example "**Exit**" and instead of `NextPage()` insert `Exit()`.

10. Run your project to see what it looks like! - Select menu **Project** and then **Run** (or just press F5).

Complete "**Hello World**" project can be found in MMB Samples folder (accessible also via List of Samples).

More Info on "Getting Started"

In the **Project Settings** window.

Window Size: When choosing a window size consider the following: Aim to keep your project window size to 1024 x 768 or less. This is the standard setting to suit a 15-17" monitors (and we can assume that this is the smallest monitor that will be used by others to view your program!). If you make your project window larger than 1024 x 768 and a user runs it on a computer with a 15" monitor with lower resolution, or for example sub-notebook with 10-12" then a portion of your program will not fit on their screen.

Standard Window: If you remove the "tick" your project won't display the normal MS Windows style of window. That is, no title bar (e.g. "Hello World" or leave the default

"Welcome!"), no exit button, and no border.
If you then place a tick in **Window has Custom Shape** you will be able to use the **Outline Shaper**, or **B/W Mask** options to change your projects display window from the standard rectangle to any other shape that you choose. Refer to the help on **Outline Shaper** or **B/W Mask** to learn how to use these advanced options.

Movable: If this option is selected AND your project window is smaller than the screen, the user can use the mouse to drag it to another position on the screen.

Style section:

Always on top: when this is 'ticked' your program will always be on top of any other programs that you run at the same time. In other words, your program will be visible over the top of any other running programs. A typical use for this would be a small **Menu** program that starts other programs, whereas you don't want the new programs to cover over your menu!

Save last position in registry: if this option is selected MS Windows will remember the previous screen position of your program, the next time your program is run. It is advisable to give each of your programs a unique name for the registry key (default is **MyApp**) if the **Save last position in registry** option is used.

Background Mode section:

Full screen background: if your project window is smaller than the screen other programs or Windows icons can be seen behind it (and therefore selected by clicking them with the mouse). The 'Full screen background' fills the surrounding space with the color selected (Solid Fill) or a picture file (Image) and prevents the user from selecting any other background programs with the mouse.

Disable Alt-Tab in Win95: This works in Win95/98 but NOT on Win2000/WinXP/ Vista!
By selecting this option you can prevent the user from pressing the 'Alt' + 'Tab' keys to switch between programs. Also works with 'Ctrl'+'Alt'+'Delete' keys!!

Cover Windows Taskbar: select this option to prevent the user from using the MS Windows taskbar to 'Start' or switch between programs.

If 256 colors selected: If a users computer is only set to display 256 colors you can use this option to run another file or simply to go directly to a specific page in the project. The page may simply display a message telling the user that the computer needs to be set to more than 256 colors to use the program! The separate file may be another project that is limited to using 256 colors.

Palette: Note: Only for 256 colors.

Graphics MMB: Optimized palette for most full-color pictures. Use this palette when creating pictures or images. Extract the palette in the Palette directory (palette. bmp) using your graphic editor, and then apply it on all images.

Windows Standard: Windows standard palette is recognized by all graphics applications. Use it whenever your projects deal with screen captures.

Note:
For added security also refer to the **CBK_EXIT** topic for capturing the Esc key!!!

(Written by Rodd edited by Odklizec)

Top Level Intro

This page is printed before a new
top-level chapter starts

Part

7 MMB Objects

7.1 Working With Objects

7.1.1 Introduction to Objects

MMB is visual object oriented authoring tool. It means that you visually place a multimedia object (Button, Image, HotSpot, Video etc. on the screen. You can resize it and if you double click on it you will see its properties. You can find all the objects in the Object menu and most of them are on the left vertical toolbar.

Introduction to Objects

Written by Rodd.

Objects are the parts that you can insert into your projects pages of Multimedia Builder to really make it perform.
By using Objects your project can:-

- Display information, graphics, special effects, or other
- Gather information, and calculate or make decisions based on the information.
- Objects can be added to a page by either selecting the top menu 'Objects' and then selecting the particular object; or by simply clicking on the object icon on the 'Object bar' at the left side of the screen. The objects' properties settings will have a section called 'Actions'.
- Many objects may have a 'Tooltip' option. Any text entered into the 'tooltip' section of the object properties will pop-up as a prompt when the cursor is placed over the object when your program is run.

An **'Active'** object is able to perform actions including: running external command and page actions; interact with other objects and video; play sound actions; or run scripts.

Cursors
Any interactive (clickable) object can have different cursor - you can choose from predefined cursors or load your own (also animated).
The cursor dialog in page, text button, bitmap button and hotspot has assigned 5 Custom cursor locations. If you select unassigned custom cursor it will prompt you to load either *cur or *ani file. These cursors will be auto-loaded into **Embedded files**. So, if you don't want the cursor anymore, you can delete it from there.

 **Text:**
Use 'text' objects to display headlines, body text or any other words that will be displayed on the screen. A text object can be active.

 **Edit TextBox:**
This object allows you to get data from the user to use in your program. It puts the data into a variable for use within scripting.

 **Paragraph Text:**
The Paragraph Text object has automatic word-wrapping (scrollable) for long text.
Drag the text rectangle on the screen and the text will fill the rectangle.
If there is more text than can fit inside the visible rectangle, the Scroll Bar will appear.

 **Text Button:**
This allows you to create standard Window button. It behaves like a standard windows button.

 **Bitmap Button:**
This allows you to make a button from three images (for the normal view, mouse over view, and clicked view); or
you can choose 'Auto-Button' to make a button with the look of a Windows button using a single bitmap for the normal view.

 **Alpha Button:** new in MMB4.8.2
Alpha Button is a special type of button object created in another Mediachance program called **Real-DRAW Pro**.

 **Bitmap:**
Bitmap object can be active or inactive. Bitmap object can be built from two parts: Image and Transparency Map (Alpha transparency). Primitive objects can be active. A Bitmap object can actually be any of the following graphic file types: BMP; JPG; GIF; PCX; PNG; TIF.

 **Animated GIF:**
Allows you to insert an animated GIF graphics file into your project. This AniGIF can also be controlled in script by using 'AGifPlay', 'AGifStop', or 'AGifReset'. In the Animated GIF properties window you can also select Transparency, Auto play, Loop, Speed, and 'After last frame' actions (hide AniGIF, or run Script).

 **Metafile:**
Allows you to insert a Windows metafile graphic into your project. Metafile graphics cannot be active and do not have a properties window!

 **VR Panorama:**
Allows you to playback the 360 Cylindrical Panorama images.

 **ListBox:** new in MMB4.9
This object is primarily designed for showing the audio playlists (winamp *.pls, *.m3u and MMB *.m3l formats), however it can be used as an usual list box control with multi-selection.

 **Rectangle:**
Allows you to draw a rectangle and select the fill color and border. The rectangle can be

active.

 **Circle:**
Allows you to simply draw a circle and select the fill color and border. The circle object can be active.

 **Line:**
Allows you to draw a straight line on the page in your project. Lines cannot be active.

 **Polygon:**
Allows you to draw a polygon on the page. You can select the fill color and a border. The polygon object can be active.

 **Hotspot:**
HotSpot is an invisible area you can define on the screen where your viewer (user) can click or move mouse to make an action happen. The area could be around a word, on parts of an image, etc. Hot spot is invisible to the viewer. However, in the designer you see hot spots as an area with a dashed border. A HotSpot is an active object.

 **Polygonal Hotspot:** new in MMB4.9.5
It's drawing functionality is virtually identical to Polygon object and only the parts inside the region will become active.

 **Video:**
Allows you to insert a video in your project using the following formats: AVI, Mpeg, MOV, VideoCD.

MCI Object:
Another way how to play Video (or Audio as well) is to use MCI interface. This could be done by using MCIcommand from the script. However, because the working with MCIcommand is a bit complicated the easy MCI object was introduced.

 **Dynamic FX:**
Dynamic FX are animated images like Plasma, Fire, Smoke. Dynamic FX don't take much space, because they are generated on runtime, but they take more CPU.

 **Audio Visualization:** new in MMB4.9
This object allows you to visualize the playback of OGG, WAV, XM and S3M formats.

 **Script Object:**
Script Object is non-active object with only script inside. It is not visible on runtime. The Script Object can have assigned keyboard shortcut so whenever user press the keyboard shortcut the script will run. This will allow to create applications without mouse, or with hidden keyboard commands (exit, special page etc..).

 **Image Matrix:**
Image Matrix is an advanced Script object.
Imagine you would like to create a Game board with 5 x 5 stones, On each position of the board you can have one of the 3 images or nothing. This could be a lot of work with putting simple bitmap object and then all the show - hide commands for all objects! Image Matrix helps you do this and a lot of other projects.
TThe number of Columns and Rows determines the size of the Image Matrix object. You must have at least the image #1 defined. The Columns and Rows are multiplied by the image #1 width and height producing the total Matrix size.
All the 3 images should be the same size.

 **HTML Object:**
HTML Object is a full Browser object integrated in MMB. It uses the Microsoft's IE core control. - example **html_browser.mbd**. This dependency on IE is a disadvantage, however, the other advantages of using IE core library outweigh this small disadvantage;)

 **Flash:** new in MMB4.9
It will allow you not only to load and play the Macromedia Flash movies, but with this object, you can control all the MMB scripting actions directly from the Macromedia Flash **Action Script**!

 **Binding Object:**
Binding Object is a way how to easily put exe file into MMB - it will become a part of the player: for example flash movie with flash projector player, your installer, notepad with text, another independent MMB project etc.

 **PlugIn:**
Allows you to insert a PlugIn into your project. A PlugIn is a small program which has its' own capabilities that can be inserted into your project. PlugIns can be made and distributed by independent programmers. Please refer to 'instructional documents' provided by the individual programmers to set that PlugIns property. examples of existing PlugIns are: clock; multiple timers; advanced functions; etc.

7.1.2 Group Object

Group is a virtual object - a group of multimedia objects.

Grouping is effective for protecting and maintaining connections and relations between objects. If you move group you move all the objects in the group. The same for most of the actions like Show, Hide etc...

Group command lets you lock objects together to create a single object.
You can select any grouped objects to move, copy or hide them.

- **Create a group of objects**

  How to create a group of objects:
  - Select all the objects you want in the group.
  - Choose Group from the Arrange menu (or CTRL+G) to create the group.

- **Add objects to group**

  How to add object to an existing group of objects:
  - Select the object, hold down SHIFT, and select the desired group.
  - Click Add to Group button, or from menu Arrange select Add to Group.

  Also, a group can be added to another one the same way.
  - First, remember to select the object you want to add and the group where you want to place the object

- **Remove objects from group**

  How to remove object from an existing group of objects:
  You can remove (ungroup) single objects from the group.
  Select the objects inside the group in the Object list, and press Remove from Group button.

- **Ungroup objects**

  How to ungroup objects:
  - Select the group.
  - Choose Ungroup command from Arrange menu (or CTRL+U).

    Within the group, you still have access to the properties of each object through Group Properties.
    The group command also lets you create nested groups – groups composed of objects or groups of objects.
    To select and group an object by its name, use the Group Manager command.
    Tip:

    Group objects together if you want to prevent accidental changes to related objects.

 Group Manager can help you to create groups by selecting the names of objects. Select objects by their name, and assemble them into a group. This is great if you want to create a group from objects by theirs name.

Note: A group could have nested groups - for example if you group a two other groups together.

7.1.3 Align Object

Multimedia Builder provides controls to align any series of objects.
Use the align tools to line up your objects precisely (left, right, top, or bottom.)

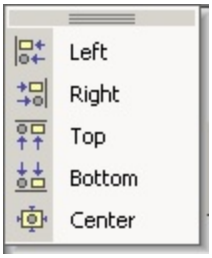
To align objects, select those you want to align:

- Click to select the first object
- Hold down the Shift key, and select the next object.
- Repeat until all objects are selected.

or

- Click near and outside the objects to be aligned. As you drag the cursor, a rectangle appears
- Hold and drag the mouse around the objects, and release it.

You can use any of the alignment tools:



7.2 Text Object

 Texts are an integral part of any multimedia design.
Use text objects to display headlines, body text or any other words that will be displayed on the screen.

Tip: Text can be active or inactive.

Label: Unique string representing the object. Use this name to interact with other objects.
Hide: Hide object. The object will not be visible for the viewer, and no object actions will be trigged.
Font: Open the font dialog box.
Align: Align the text Left, Center or Right.
Enable the actions: Enable the object to trigger some actions.
Color Interaction: If actions are enabled, text can change color when the mouse moves over or user clicks on the text.
Action Buttons: See the **actions**.

Note: For longer texts and paragraphs use the **Paragraph Text** object instead.

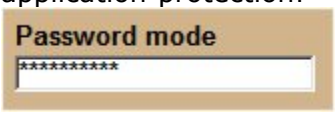
7.3 Input Text (Edit Box)

 The Edit Box exports the text to the defined string variable and in case of integer or float type input box it also exports the numerical value to the defined numerical variable.

Input Text properties:
Fixed width: If enabled, then the number of entered characters is limited by **width** of InputText object. If unchecked, the width of Input Text object is automatically updated according the length of entered text.

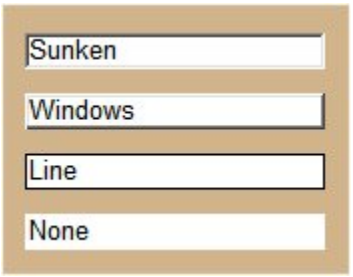
Enable scrolling: This parameter is allowed only if the above "Fixed Width" is enabled. This will allow you to enter more characters than the length of InputText object, however, the width of Input Text remains unchanged. The text inside the object start scrolling if the length of text is higher than the length of Input Text.

Password: If enabled, all characters entered to Input Text will appear as asterisks. However, inside the variables are still real characters. It's good in case you need to create an "Enter the password" box. Remember, that this option does not use an encryption or another kind of protection! So **it's not safe** to use it as your main application protection!



Enable menu: This option enables/disables the right click (in realtime) menu with Cut/ Copy/Paste operations and their shortcuts.

Border: In this section can be set the border around the Input text object. For example, with Sunken border it will appear like a standard windows edit box. Border color can be set only for Line border. Both Color and Border can be also enabled/ changed via Script (see SetObjectParam command). Here you can see the appearance of available borders..



New in 4.8: The Variable associated with Input Box is now automatically initialized with the default text. You don't need to separate initialize the variable. (You can still override the variable in Page Start script)

To change the text in the Input Box on runtime you have to use:
LoadText("OBJECT","STRING VARIABLE")
The string variable doesn't have to be the one associated with the Input Text Object.

for example:

```
my$=' Enter your name here'
LoadText("EditBox","my$")
```

For more information see LoadText **description**.

If you use non-string type (integer or float) the user will be able to enter only the characters used by this type (numbers, floating point, exponent, minus sign).

7.4 Paragraph Text

 The Paragraph Text object has automatic word-wrapping (scrollable) for long text. Drag the text rectangle on the screen and the text will fill the rectangle.

Note: If the text is longer than the visible rectangle, the Scroll Bar will appear.

Tip: A user can scroll the text two ways: by dragging the scroll bar, or by clicking on the text and moving the mouse up or down. (The cursor will transform into a hand.)

7.5 Text Button

 You can create standard Window button. It can have different fill and color.
It behaves like a standard windows button…

Actions
Text button is an active object all of the time. See the **Actions**.

Menu style
This is a special style of the button where it looks like a menu item. Great for building menus.

The Button can have its own font or it can use standard font (MS Sans Serif)

7.6 Bitmap Button

 You can create custom graphic buttons out of 3 images.
The normal Image, The Highlighted image if mouse is moved over the button and the Click Image if the mouse is clicked on the button.

Auto-Button
This allows you to create bitmap button only from one (Normal) image. The button will have look of Windows button – with your image. You need to load only the Normal Image.

Refine Bitmap position:
You can change the relative position of the highlight and click image to the normal image: To get soft delightful effect or if the images don't have the same origin.

All of the images can have one transparent color.
The color box will help you to define the background color with Automatic color. It is the top left pixel of the image.
You can save your buttons and make your own library for easy access later.

Bitmap buttons are an essential part of all large graphic multimedia applications.

You can make this object interactive by enabling the actions.
For more info see the **Actions**.

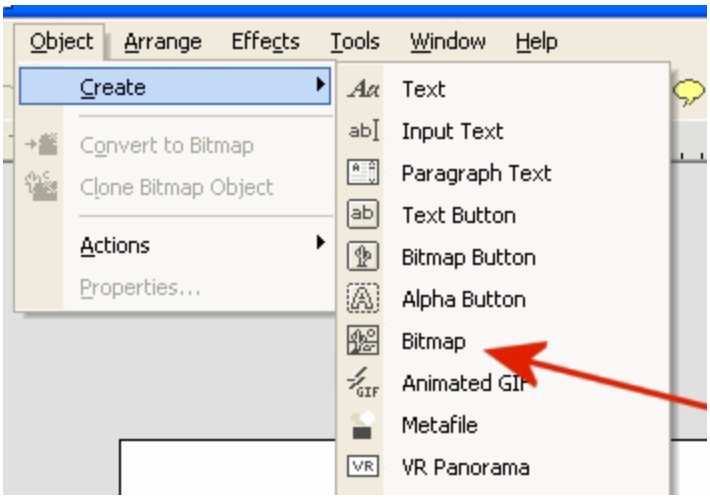
7.7 Alpha Button

 Alpha Button is a special type of button object created in another Mediachance program called **Real-DRAW Pro**. For an example on how to create and export this type of object from Real-DRAW Pro see the **tutorial chapter** from Real-DRAW help.

7.8 Bitmap Object

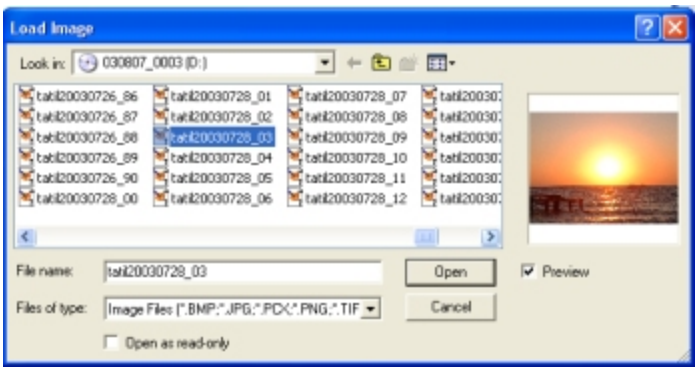
 Bitmap object is a window to display images. It support's bmp, jpg, pcx, png, tif formats. Let try to insert a bitmap to our MMB project.

1. Press Bitmap icon from the toolbar or select Create>Object>Bitmap from menu.



2. Place the cursor to the page

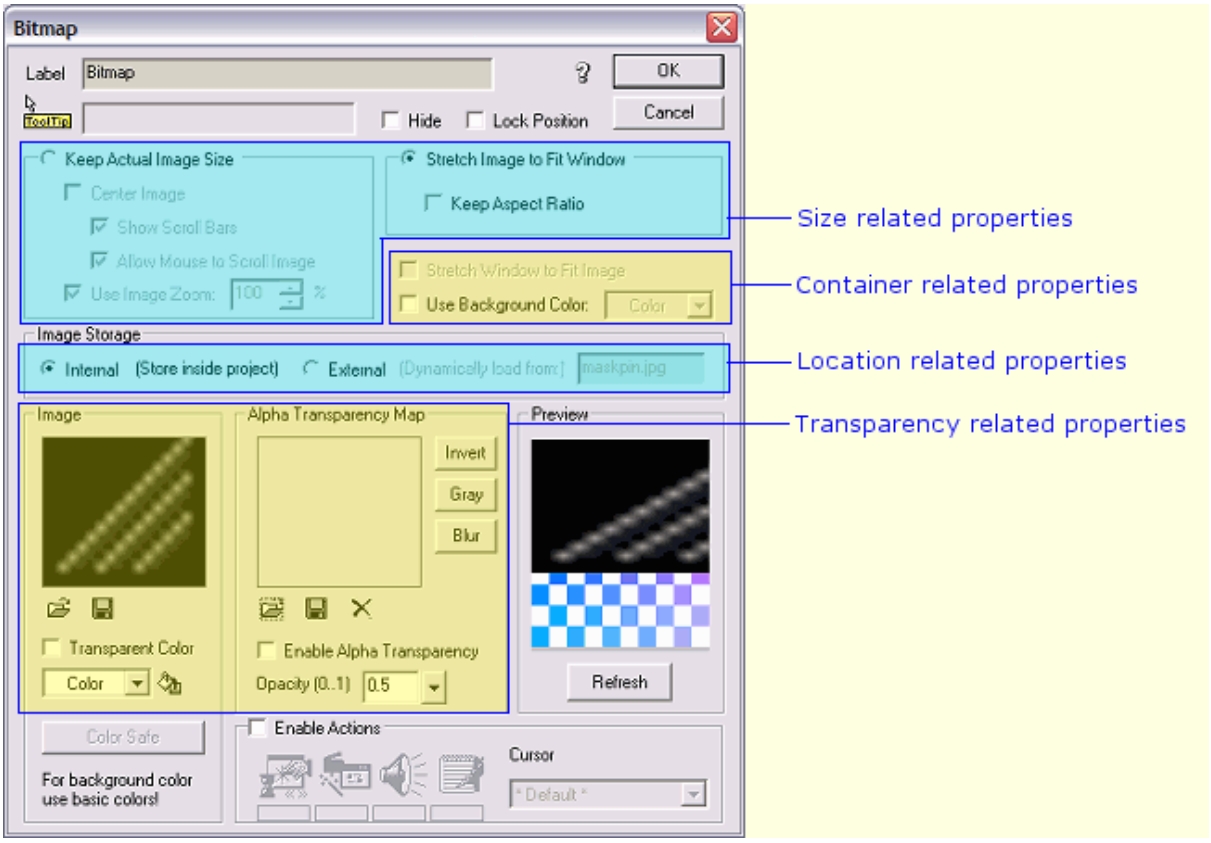
3. Select image file.



4. If necessary resize the bitmap. If you want to keep aspect ratio press CTRL while resizing. To restore to the original size select "Restore Original" from the Effects menu.

Adjusting Properties

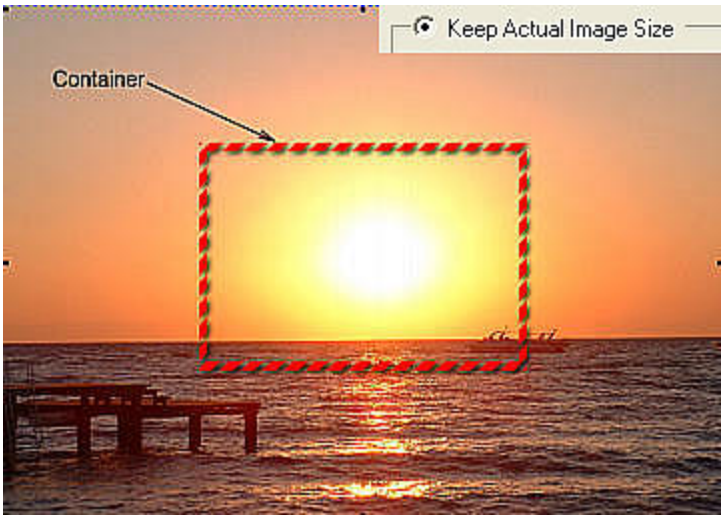
Double click on the bitmap object. The properties pop-up will be opened.



a. Size related properties

There are two main options:

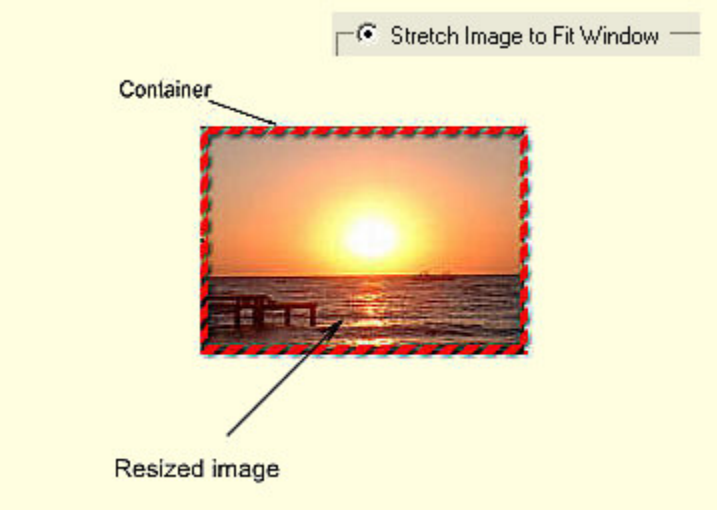
- Keeping the image at original size ***[Keep Actual Image Size]*** or



This option preserve the actual Image dimensions. Even you resize the Bitmap object, the image size stay unchanged. It will allow you to make smaller/larger Bitmap object with the scrollable content. To scroll image inside the smaller/larger

Bitmap object you can use Scroll Bars (enable **"Show Scroll Bars"** option), dragging the mouse inside the Bitmap object (enable **"Allow Mouse to Scroll Image"** option) and finally the **ScrollImageView** function. With "Keep Actual Image Size" enabled you can also zoom the image inside the Bitmap object. This will allow you to zoom the image inside the Bitmap object

- resizing the image to the containers size ***[Strech Image To Fit Window]***

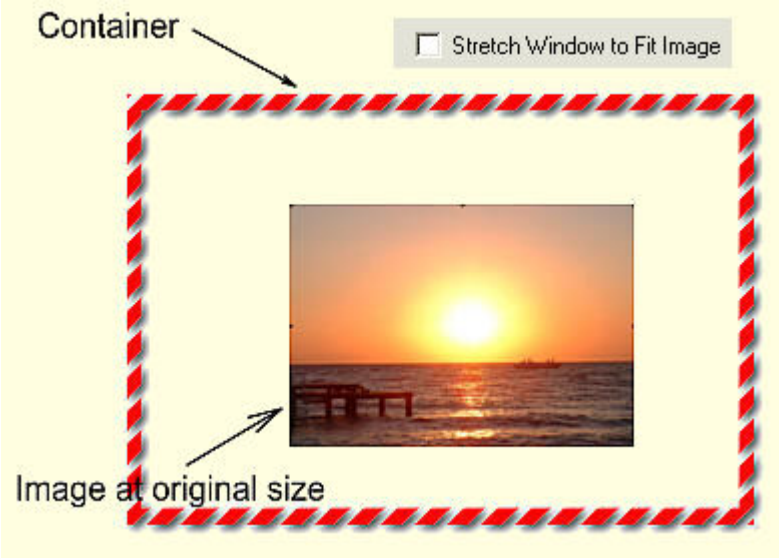


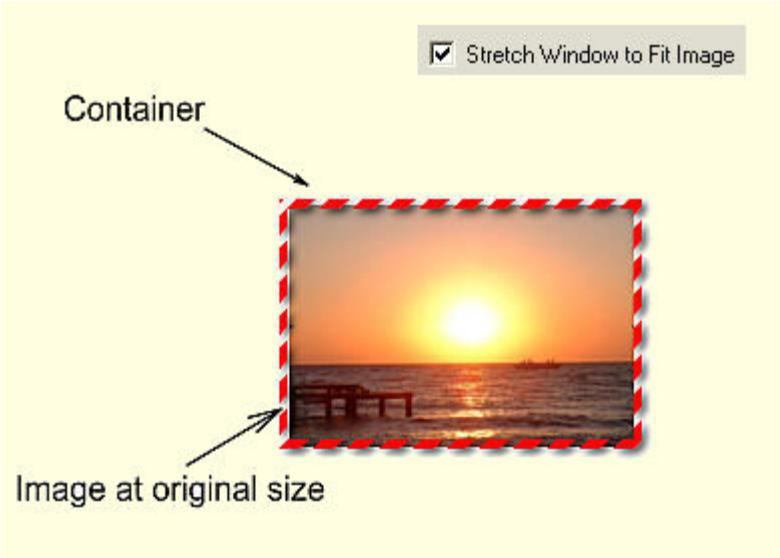
If you keep the image at original size you could also have some options

- whether image is centered within the container or not ***[Center image]***
- whether the scrollbars are visible or not and ***[Show scroll bars]***
- whether cursor could scroll the image within the container or not ***[Allow mouse to scroll image]***
- Zoom in or out at the original image ***[Use image zoom]***

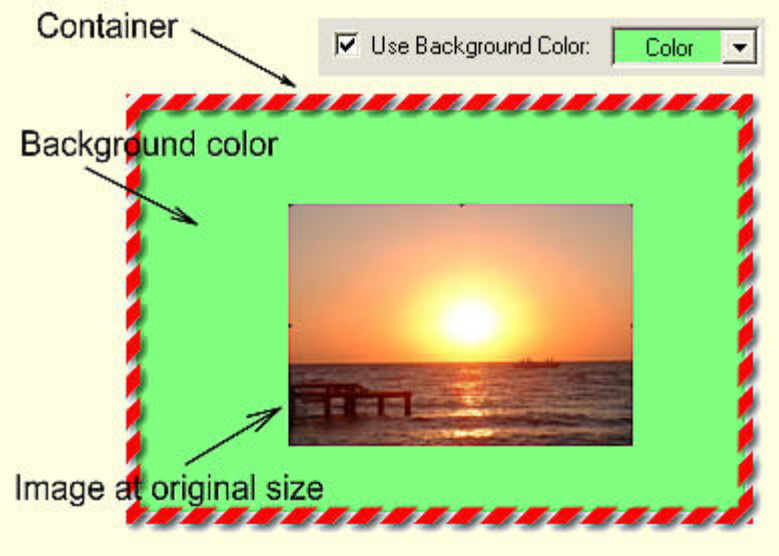
If you select resizing the image to the containers size you could have an option that whether keeping aspect ratio or not. To keep aspect ratio simply click the check box.

b. Container related properties





You could fit the containers size to the original image sizes. Look at the above illustrations. The image is at the same size at the both side. But the container is resized at the right.

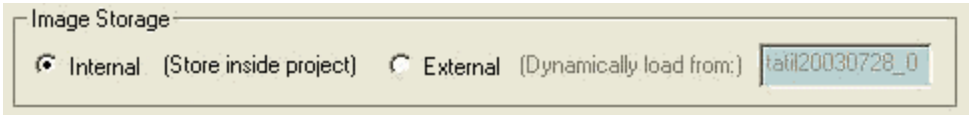


You could also select a background color for container. Check above illustarion.

c. Location related properties

Normaly, MMB embeds the media sources (images, sounds, etc.) This will give us an opportunity to make a standalone application. So it will be distributed & installed easly. But sometimes there are many media sources, many images and many sounds. If you embed all the sources in to the MMB, then the compiled EXE will be very large. This is an unexpected situtation because a large file limits distributing, sharing, e-mailing, using etc. What can we do?

The answer is not to embedding image files. This means image files could be located outside the EXE and we could load it when it needed. Check the below images:



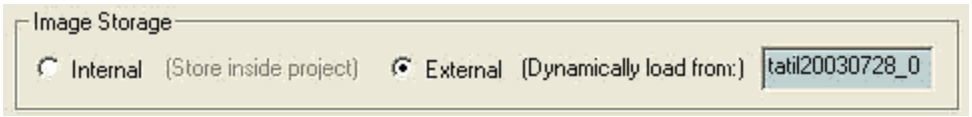
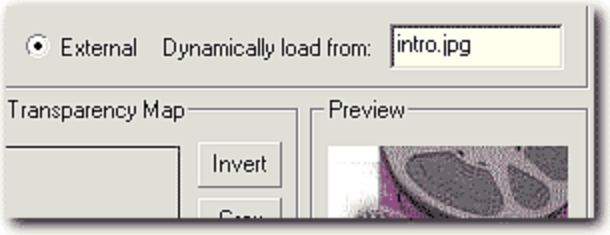


Image storage: External Storage:
Now you can specify the image object to be loaded internally (as previously on project start) or Externally (on each page by demand loaded from external file)

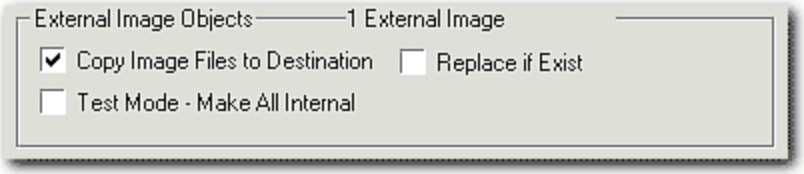


This is good if you:

- Need to often change images in your projects (for example multilingual etc..)
- Need to develop application with many images and don't want to store them in the autorun itself, the autorun will be smaller and it will need far less memory - images on each page will be loaded by demand and then when exiting from the page they will be flushed from the memory.

All is done in **seamless**way - you don't have to really worry about the external images until you Check & Distribute your project.
It is important that you acknowledge this:

- You work with the image object as before.
- In designer **all** the **images** will be stored internally in the mbd file. You will only set option if you want the image **during playback** load internally or load from external file. That means the project file doesn't depend on any external image files - all are inside the project as before.
- Test Run or Debug still uses the internal images
- All necessary things are done during Check & Distribute. External images will be copied from the internal project into the distribution location in the \Images folder.
- That means you can simply use **any project you already have** and set some images to be external. You don't need the original files - the images will be pulled from the project itself and saved.
- You can use *.jpg or *.bmp as your desired image. You can also use any other extension. (For example *.oscar) in such case the image will be saved with this extension as jpeg.
- The Check & Distribute has few more options for the External images



The **Copy Image Files to Destination** will do what was described above - copy the images from internal storage to the \Images folder. If you already have images there and you don't want to create the images every time you Check&Distribute - uncheck this option
Replace if Exist - during copying replace the files if they are with the same name in the \Images directory.
Test Mode - this will make again the external images internal, no images will be created and all will be stored in the autorun itself as it was done before.

- **Loading Time**: Internal images are all loaded on the project start, that means they take more memory and the start of the application is delayed, however the jumping within pages is fast. External images loads only on particular page. The memory is freed as soon as you exit from the page. That means less memory requirements for many pages on player machine and faster startup. However loading each page may be delayed depending on the number of images needed to load.
- The result autorun containing external images is obviously much smaller than if the images are internal - but you need to carry the \Images directory.
- If you carefully read all the above you would know that during designing nothing is changed. Not even test run from within designer - during internal Run or Debug the images are **still used from internal storage**. Only if you use Run External Player in location (Menu Tools - Testing from Designer) then the images will be indeed copied and the player will simulate the real thing. And of course during

Check & Distribution...
- Even if you can now create presentation which doesn't need much memory to run since images can be loaded dynamically, you still need memory during designing - again because in **designer** the images are not loaded dynamically.
- It is indeed very easy to use, even if our explanation could be scary.

d. Transparency related properties

The transparency related properties is as same as the other objects transparency features.

To make an image (or part of it) transparent or semitransparent, you have to use more sophisticated tools – **Alpha Transparency Map**.
It is usually Black & White pictures where Black means Full transparent and White is opaque or it has the opacity specified in the **Opacity box** (0 – fully transparent, 1 - opaque).
You can switch on/off the alpha transparency by checking **Enable Alpha Transparency** check box.
If you don't define the transparency map, but you enable the Alpha Transparency, the image will blend with the background with opacity defined in the Opacity edit box.
You can apply a few useful effects with the transparency map – **invert**, make it **gray**scale and **blur** the map.
The typical usage of the transparency map is the Glow or Drop Shadow. The image is just rectangular with some color. The shape and the effect is done by the Transparency map.

If the Transparency map is smaller than the image, it will be tilled, so you can use it for some nice effects.

Fill button – near the Color combo box.
You can fill whole images with a single color as defined in the color box. A great way to change the color of the Glow or Drop shadow like objects.

NOTE: If you resize the Image object placed in the project page or you change the Zoom level in the Image Properties dialog, then the copy (invisible) of the original bitmap will be created. The main disadvantage of this fact is that it can significantly increase the size of the project file and the compiled exe. Of course, you can remove this redundant copy of the image (File>>>Reduce Size menu), but this will avoid restoring the image to it's original size (by RestoreImage scripting command) or the zoom level to the original value.

Limitations: Rotate/Resize image functions will not work for bitmap object with "Keep Actual Image Size" option enabled. Only Zoom image is alowed for that object.

For Images related scripting commands **look here**. An image example projects can be found

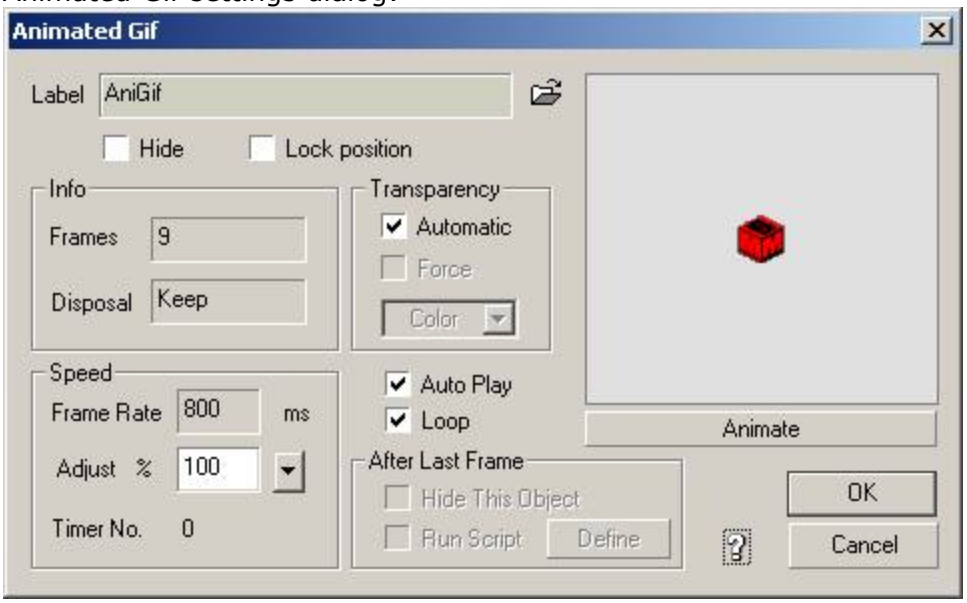
7.9 Animated GIF Object

Animated Gifs are very common on Internet. You can get thousands of gifs just browsing the net or buying a not expensive clipart CD.
You can work with ANIGIFS like with any other objects – grouping them, moving them in layers, etc.

Experiment with the settings!
This is not all what you can do with the anigif!

Note: Anigifs takes more CPU than static pictures. You can use Anigifs in layers - be careful - the alpha transparent bitmaps takes more time to redraw - if you overlap Anigif and some Alpha transparent bitmaps - the gif will slow down.
Don't try to put too much on one page ! The anigifs are great for highlighting something special.

Animated Gif settings dialog:



Basic options:

- **Lanbel:**
  String represents the new object.
- **Info:**
  Frames
  Number of frames in the gif (gif must have more than 1 frame to be animated)

  Disposal
  There are few types of animated gifs using different optimization – the number bigger than 0 shows the gif is using one of the optimization (keeping the first frame and showing only differences, keeping the prev. frame …)
- **Speed:**
  Frame rate

The rate in ms (the time between the frames) , it is the rate of first frame – other frames can have different rate

Adjust
Adjust the frame rate by percent (200% 2-times slower).

Note: Speeding up the animation doesn't have to have a desired effect all the time.
Example: you have Frame rate 10 ms, you adjust the speed 50% (5 ms). No effect ! The 5 ms is not enough for the computer to redraw the frame. (Even the 10 ms is not enough!)

Advanced options:

- **Transparency:**

  Automatic - Force
  You can choose how to create transparency in your image. Note - this goes beyond standard gif transparency, you can turn non-transparent gifs to transparent by forcing the transparent color.

- **Auto Play:**

  The gif will start playing as soon as it will be visible

- **Loop:**

  The gif will play in loop

- **After Last Frame:**

  this works only if Loop is not selected. You can choose what to do when animation will reach its last frame. It could hide the object or run a script.

If you Show anigif in your actions - anigif start animate from the first page, if you Hide it, it disappears.
In script, you can control more with next three **GIF commands**.

7.10 Windows Metafile

 Windows Metafile (WMF) format is very popular vector format. You can export vector graphics (for example from CorelDraw) into the vector format. Metafile format can be scaled without loosing the quality.

MMB also has support for this format.
In this version there are some restrictions: Metafile object cannot be active, it doesn't even have properties.

To place metafile objects on the screen:
Select Metafile from the Object menu.
Draw a rectangle on the screen, then the Open dialog will appear. Select .WMF file.
Resize the rectangle to desired size.

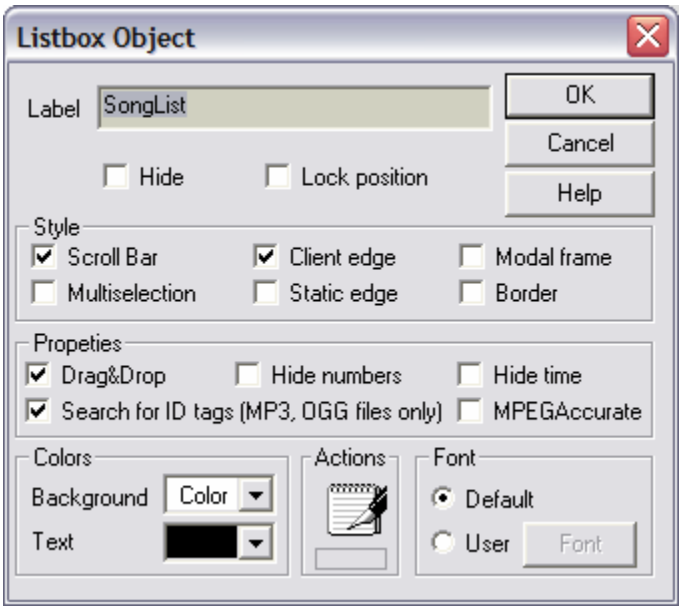
7.11 VR Panorama

360 Panorama pictures are special pictures taken with digital camera in few angles and then stitched together - producing one long Panorama image.
MMB can play back the 360 Cylindrical Panorama images with the VR Panorama object.

7.12 ListBox Object

 The List box object displays a list of items from which the user can choose one or more items. MMB ListBox object is primarily designed for showing the audio playlists (winamp *.pls, *.m3u and MMB *.m3l formats), however it can be used as an usual list box control with multi-selection or as a playlist for all supported (in MMB) audio/video files.

You can fill this object with text, string variable, string array, text files, audio/video files (with their times) or the early mentioned playlists. This new type of object also bringing new type of events to the MMB – "Double Click" and "Drag&Drop" (at the moment only for ListBox object). In addition, there are many new scripting commands, allowing you to effectively work with the MMB ListBox object.



In ListBox settings dialog you can set the Background and Text colors, displaying the ScrollBars (both horizontal and vertical), set style of the ListBox border, custom font and of course action scripts for "Double Click", "On Selection" and "Drag&Drop" events.

Style options

Sets several usual ListBox styles..just try them and you will see the result ;)

Properties

Drag&Drop - this option allow/disallow drag&drop to ListBox (by default ON).
Search for ID tags (OGG and video files) - it search for time of audio files loaded to ListBox via playlist or **internal song list**. This feature is dependent on FMOD audio library, therefore it must be enabled and project must be compiled with Full player (by default ON).
Hide numbers - hide numbers of ListBox items (by default OFF).
Hide time - hide time of audio/video files loaded in ListBox (by default OFF).
MPEGAccurate - This will allow you to load VBR (variable bitrate) audio files into ListBox with correct time. But the dark side of this option is much longer loading time, mainly if you load too many files at once (by default OFF).

Check the ListBox **scripting commands** for advanced work with the ListBox.

Here are some example files:
listbox_tut.mbd - a general ListBox example
move_up_down.mbd - move UP/DOWN items example
listbox_on_master_page.mbd - ListBox object used as a page navigation object
listbox_fileextensions.mbd - ListBox Drag&Drop and adding items with filename +extension to ListBox

7.13 Primitive Objects

Primitive objects can be active (except Line).

 **Rectangle**
It can be filled, with a border (Line), without a border (None), or using a windows border – raised or sunken.

 **Circle**
Uses the same properties as the rectangle object.

 **Polygon**
With this version of MMB the polygon has very limited functionality, However, from MMB4.9.5 it can be resized and it can have a different fill and border, or no border at all.

 **Line**
Line is an inactive object. It cannot have an action.

You can make this object interactive by inserting some actions.
For more info, see the **Actions**.

Tip: Nice multimedia applications can be created just by using the primitive objects and text!

7.14 Hot-Spot Objects

 Rectangular and Polygonal Hot-Spots are the active objects.

Hot-spot is an invisible area you can define on the screen where your viewer (user) can click or move mouse to make something happen. The area could be around a word, on the image etc.
Hot spot is invisible to the viewer. However, in the designer you see hot spots as an area with a dashed border.

From 4.9.5 you can define new polygonal hot-spot, which drawing functionality is virtually identical to Polygon object.

TIP: Hot-Spot object can contain the image data loaded by ReplaceImage command. In case of the Polygonal Hot-Spot, the Image will cropped by defined polygon.

7.15 Video Object

 In order to play, stop or control the video, you have to place other controls on the screen with assigned «Interaction with other objects and Video» action or you have to start and control video from script via Video related functions (**see VideoCommands**).
The video object has two files: The Still image and the video itself.
The Still image stays on the screen when video is closed (it is not playing) and it is a normal bitmap object.

TIP: Wizard can insert video controls for you.

After you create a video object, select it and click on Wizard button (the Magic Wand on top), and select «Insert Video Controls». This will create a few bitmap buttons attached to this video object.

Check this article for a detailed **VideoTutorial**.

Limitation:
Even you can place a rectangular objects on top of Video object, any non-rectangular or transparent Image placed over the Video object will still appear like a rectangular object with a white or gray background. There is nothing that we can do with that.

Basic options:

Video File
Video is an external file, it will stay outside the mbd file. Video file is loaded into Video object via **Video import dialog**.
It is essential to specify a relative path to the file using < SrcDir> or < SrcDrive> commands in the path.
You can do it later for all objects with «**Path Replace**» in Project menu. More about relative paths in **Constants** topic.

Hide Still Screen
Video, if it is not playing is represented on the screen by the still image. However, you can hide this still image (with Hide option in Video Properties or by Hide scripting command (and then show it by Show).

Sound
You can disable sound if the sound channel exists in the AVI file
Note:
AVI Audio plays through the standard audio output and it cannot be played at the same time with DirectSound channels.
Be aware!, video with sound will shut down DirectSound Channels and any background music will stop playing.
Shortly, the sound of video cannot be mixed with background sound.

Speed
MMB allows you to play video at any speed. The default value (normal speed) is 1000. Less than < 1000 , and the video will play slower and in reverse.

Loop
Video can play in the loop until the user triggers the STOP **action.**

Save Still
Maybe you would like to save the still picture for the future or to use it somewhere else. This will be handy.

Load New
Load new AVI file or reload the old one and select different still image.

Advanced options:

Full Screen:
The video will play in full screen mode. Clicking inside the video will turns it back to normal view mode. This is fully supported by AVI. However, some MPEG drivers allows full screen some not or doesn't show the video with proper dimensions or aspect ratio.

After Video Finish events.

- Do Nothing (default option) - As the description says, it does nothing after Video finish.
- Reset Video - Reset video to starting position.
- Close & Hide Video - Unload video from Video object and Hide still screen.
- Go To Next Page - Simply go to next page when video reach its end.
- After the video Load/Start/Stop/Finish you can chose one of the actions for example to go to next page or run script.

Run script after Video Load/Start/Stop/Finish events.
After the video Load/Start/Stop/Finish you can define script for each of the above mentioned Video events. This will allow you to react if user for example stops Video during playback.

Mask:
With the Mask, video doesn't have to play in rectangular square anymore and it can have any shape you want.
You need to prepare an image mask with size of the video. The video will be played through this mask where the black pixels will be video and white transparent. The mask could have any shape or image (for example black text on white background). You can load the mask in the Video Properties. (Two buttons were added - Load and Clear Mask) You can create very interesting effects because finally you don't have to be stuck with the boring rectangular video found in many authoring tools...

Recommended SW configuration for successful Video Playback:

- Win9x/W2k/WinXP
- Windows Media Player 7 or higher (if you want to play ASF/WMV files)
- DirectX6 or higher

LIMITATION:
Color depth of the video files (and display color depth) must be 16bpp or

higher! Playback of the 8bpp video files is possible, but MMB cannot create still image for these files (or that low display color depth).

Supported Video (Audio) formats:

Note: Audio formats mentioned in the below list can be used only with **VideoLoad** function. Audio CBK variables as well as AudioVisualization object will not work with Audio files played over Video object.

- **Audio Visual Interleave (.avi)**
- **Windows Media (.asf, .asx, .wma, .wmv)**
  Advanced Streaming Format (ASF) is a file format that stores audio and video information
  Advanced Stream Redirector (ASX) files are used to direct users to streaming media content, usually on multimedia Web sites
  Windows Media Audio (WMA) is an audio codec that has been created by Microsoft. The codec is designed to handle all types of audio content.
  A **Windows Media file with Audio and/or Video (WMV)** is used to download and play files or to stream content. The WMV format is similar to the ASF format.

- **Moving Pictures Experts Group (.mpg, .mpeg, .m1v)**
  The **MPEG** standards are an evolving set of standards for video and audio compression developed by the Moving Picture Experts Group (MPEG).

  MPEG-1 was designed for coding progressive video at a transmission rate of about 1.5 million bits per second. It was designed specifically for Video-CD and CD-i media. The most common implementations of the MPEG-1 standard provide a video resolution of 352-by-240 at 30 frames per second (fps). This produces video quality slightly below the quality of conventional VCR videos.

  MPEG Audio Layer-3 (MP3) has also evolved from early MPEG work. It is an audio compression technology that is part of the MPEG-1 and MPEG-2 specifications. Developed in Germany in 1991 by the Fraunhofer Institute, MP3 uses perceptual audio coding to compress CD-quality sound by a factor of 12, while providing almost the same fidelity.

  MPEG Audio Layer-2 (.mp2, .mpv2)

  A proposed **MPEG-3** standard, intended for High Definition TV (high definition television), was merged with the MPEG-2 standard when it became apparent that the MPEG-2 standard met the HDTV requirements.

  MPEG-4 is the latest of the Moving Pictures Experts Group standards to be approved by the International Standards Organization (ISO). Microsoft has created the first implementation of this standard in the United States in Windows Media Technologies with the release of the Microsoft MPEG-4 version 3 video codec. This standard was developed specifically for encoding multimedia content efficiently in a variety of bit rates: from low Internet rates to rates that reproduce a full-frame, television-quality presentation. The Microsoft MPEG-4 video codec intrinsically supports streaming

multimedia by allowing for multiple streams within one encoded data stream. It also has an advanced motion estimation algorithm, which allows for greater compression.

- **Musical Instrument Digital Interface (.mid, .midi, .rmi)**
  Musical Instrument Digital Interface (MIDI) is a standard protocol for the interchange of musical information between musical instruments, synthesizers and computers.

- **Macintosh AIFF Resource (.aif, .aifc, .aiff)**
  Audio Interchange File Format (AIFF) is an audio file format developed by Apple Computer for storing high-quality sampled audio and musical instrument information.

- **Sun Microsystems and NeXT (.au, .snd)**
  Unix Audio (AU) files are UNIX-generated sound files.
  A **Sound (SND)** file is an interchangeable sound file format used on Sun, NeXt and Silicon Graphics computers and usually consists of raw sound data followed by a text identifier.

- **Audio for Windows (.wav)**
  Wave Form Audio (WAV) is a file format in which Windows stores sounds as waveforms.

- **QuickTime Content (.mov, .qt)** (LIMITED SUPPORT!)
  QuickTime is a file format developed by Apple for creating, editing, publishing and viewing. QuickTime supports video, animation, graphics, 3D and VR (virtual reality). While Microsoft does support the MPEG format itself, AVI files encoded with this particular codec (MPEG4v3) are not supported in AVI format. Windows Media Player supports MPEG4v3 files in . asf streaming format only. The .avi file can be converted to an .asf file using Windows Media Tools Windows Media Encoder.
  Files in the QuickTime versions 2.0 and earlier .mov and .qt file formats are supported in Windows Media Player 7 if you have upgraded from a previous version of Windows Media Player. If you have not previously installed Windows Media Player, Windows Media Player 7 does not support playing QuickTime files.

7.16 MCI Object

 Another way how to play Video (or Audio as well) is to use MCI interface. This could be done by using MCIcommand from the script.
However, because the working with MCIcommand is a bit complicated the easy MCI object was introduced.

This makes easier to play video or audio using MCI without doing much scripting. Let's say you want to play ASF so instead of bunch of MCICommand script lines you simply draw a MCI object which could play your asf automatically on page start or you can control basic functions (play, stop, close) with new script command MCIObject.

You can use it even without any scripts if you select Run afterpage start. Then the video will simply play and finish.

To control MCI object you use **MCI script command**.

Note: MCI objects use the windows drivers to play selected format! User has to have Windows Media Player installed to support formats like mpeg, ASF, mpeg4 ...

7.17 Dynamic FX

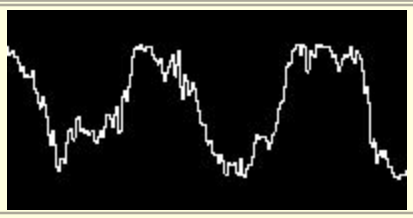
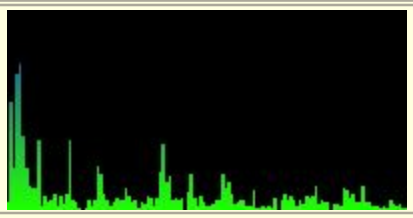
 Dynamic FX are animated images like Plasma, Fire, Smoke. Dynamic FX don't take much space, because they are generated on runtime, but they take more CPU.
On slow computers, they can take so much CPU so your Script Timer or Page Timer commands will have no time to execute. Always remember this when you working with Dynamic FX.

There are many parameters, just experiment and you will get very impressive results.

7.18 Audio Visualization Object

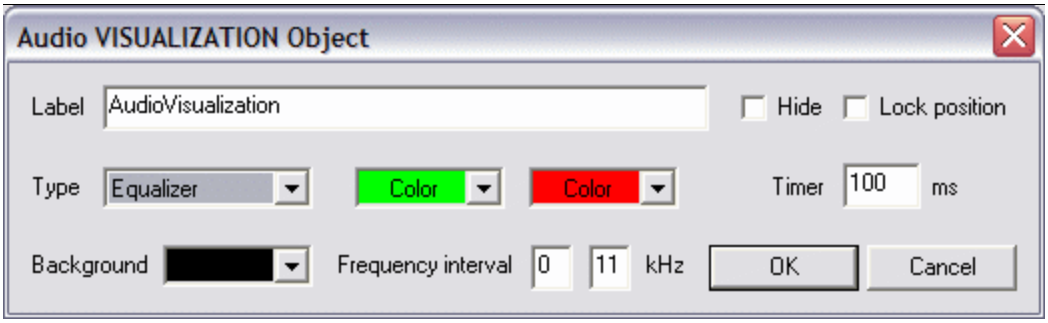
 With this new type of MMB control, you can visualize the playback of OGG, WAV, XM, S3M, IT, MOD formats. In fact, it has no a practical sense, but it looks great and users mostly loves the moveable and blinking gadgets:-)

At the moment, MMB supports the two types of audio visualization objects:

oscilloscope	equalizer
	

Of course, you can change the type of audio visualization object or change the curve (Oscilloscope) and the background color in a design time as well as in runtime (by script). For example, you can change the type of AV object from oscilloscope to equalizer by click (ala Winamp) on an object (most probably HotSpot).

Here is the AV settings dialog:



Frequency interval (kHz):
Most usable Spectrum values for Equalizer type of Audio visualization is around 0 to 11 kHz. Higher values cause a higher sensitivity, but smaller visible effect (too big range to display).

Timer (ms):
Recommended Timer value is around between 50-100ms. Smaller values cause often refreshes, but they also raise the CPU usage.

You have to experiment with both values for obtaining the best possible result.

Limitation:

- You can't have multiple AV objects of a different type on a page at the same time.

AV Scripting commands

7.19 Script Object

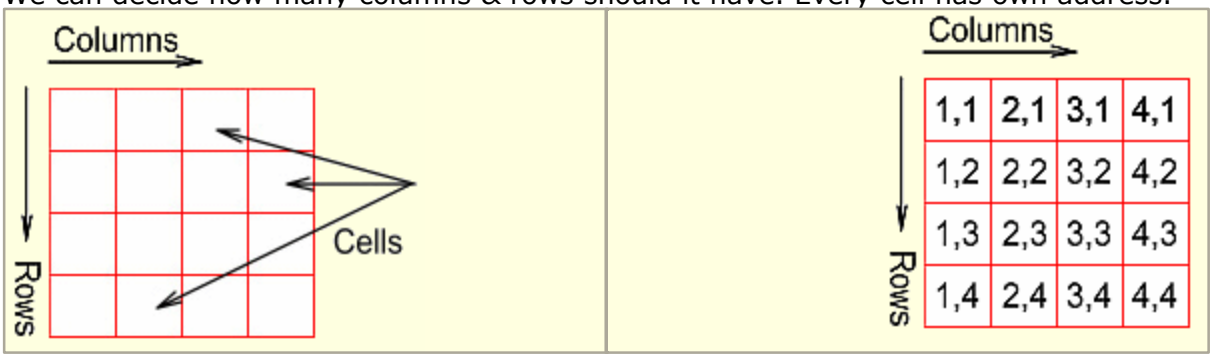
 Script Object is non-active object with only script inside.
It is not visible on runtime.
The Script Object can have assigned keyboard shortcut so whenever user press the keyboard shortcut the script will run.
This will allow creating applications without mouse, or with hidden keyboard commands (exit, special page etc..)

In order to use the shortcut option check «Run Script by Pressing» and specify the keyboard shortcut – any combination of a character, shift, control, alt. (You can have only Shift, Control or Alt without any character)
In the combo box are special keys like up,down,home …

For more info about MMB Script Language see the **Scripting Unleashed** chapter.

7.20 Image Matrix

Matrix Object is an advanced script object in MMB. It contains rows and columns. We can decide how many columns & rows should it have. Every cell has own address.



Imagine you would like to create a Game board with 5 x 5 stones, On each position of the board you can have one of the 3 images or nothing. This could be a lot of work with putting simple bitmap object and then all the show - hide commands for all objects! Image Matrix helps you do this and a lot of other projects.

The number of Columns and Rows determines the size of the Image Matrix object. You must have at least the image #1 defined. The Columns and Rows are multiplied by the image #1 width and height producing the total Matrix size.

All the 3 images should be the same size.

You can reach each cell with **MatrixName[Column, Row]** format.

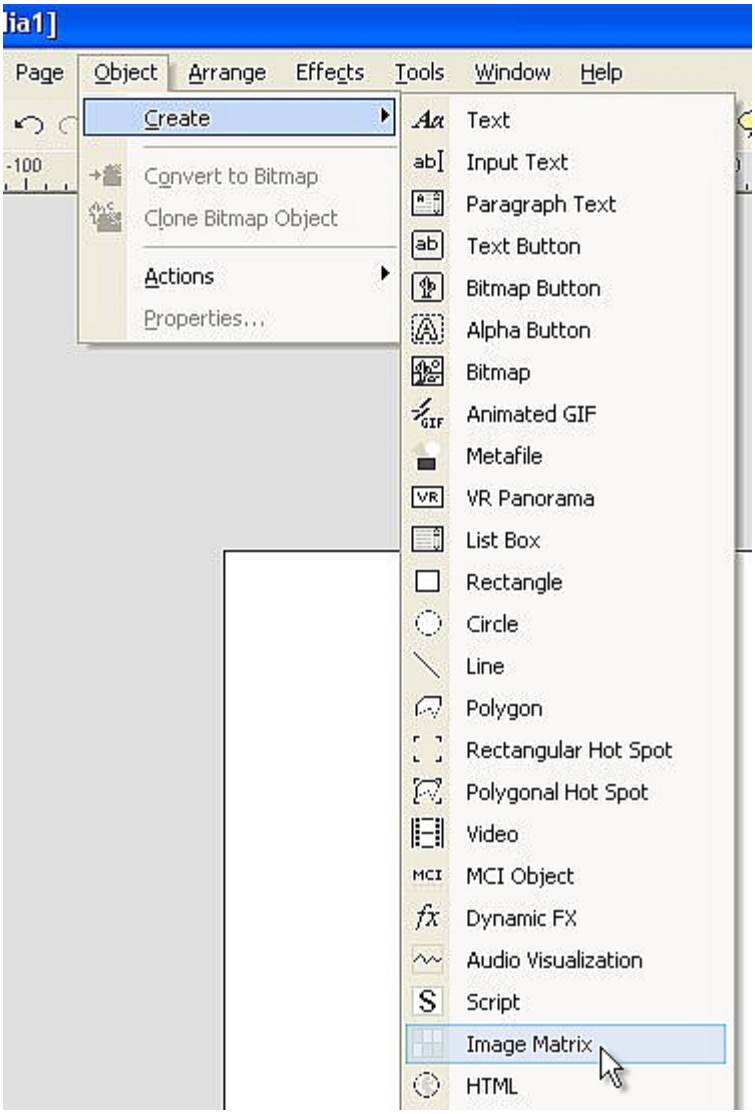
Images?

You can use max. 4 images with the matrix object. Suppose that we have selected below images:

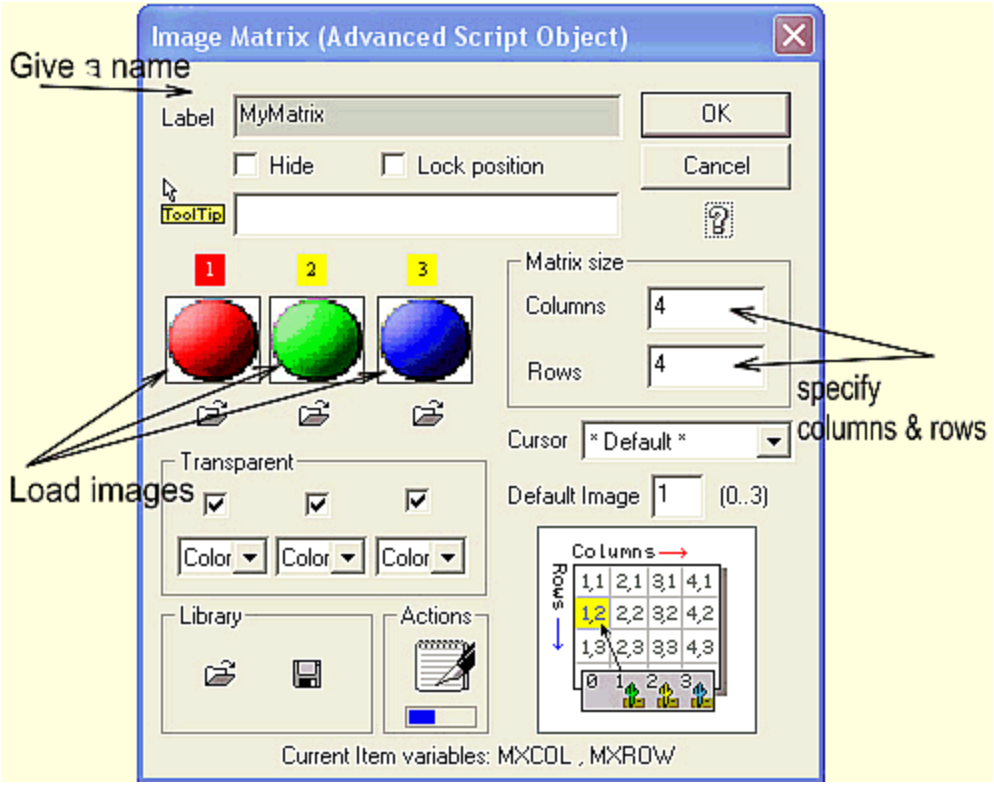
	RESERVED FOR NULL IMAGE			
Name	image0	image1	image2	image3
Image index	0	1	2	3

Putting It Together...

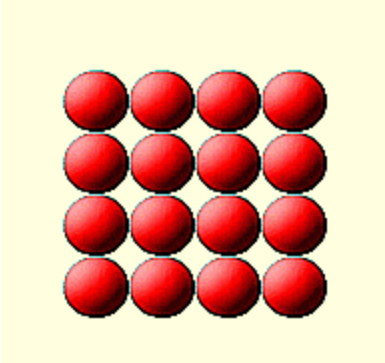
First run MMB & open a new project. Put an image matrix to the page.



Adjust the properties.



Matrix object will be filled with default image.



See also the **Matrix Functions**.

7.21 HTML Object

 Html Object is a full Browser object in MMB. It uses Microsoft IE control. - example **html_browser.mbd**
this dependency on IE is a disadvantage, however, the below advantages of using IE core library outweigh this small disadvantage...

- One of the most complete HTML control
- You can print from it
- You can use forms, Java, Java Script, Applets, Frames anything IE can
- You can view files from CD, disk or connect directly to Internet
- You can use FTP
- It works also as ActiveX container enabling you to view **Acrobat pdf files**, Flash movies and anything to what you have plugin/ActiveX installed
- No need to pay royalties - comes free
- No need to carry additional dll's
- Works with IE 4.x and newer, no need for special version
- Most of the users already have it installed, all users using Win98SE and 2000
- If other authoring tools use it so why not MMB

MMB specific issues:

- You can show or hide Status Bar
- All commands are invoked with Script command, so you can create your own buttons (if you want) or actions for Back, Forward, Stop, Print, Open, Reload.... or you can have no buttons at all
  MMB Html Object can insert ready made buttons for you if you need them and you are in hurry.
  See the **Html Object command** description.
- Html browser can be hidden by Show, Hide or moved by Move Command
- You can use Embedded Html files - but remember to embed all other dependent files such images or other pages.
- If user doesn't have IE, then the Object will run whatever default browser user has installed.

Hide Border and Always Hide Vertical Scroll Bar
You can Hide the border of the control and eventually hide the vertical scrollbar. This way you can display HTML file in your MMB project without disturbing the look of your application.

Tip 1:
You can use a special links in your HTML files to tell the MMB player to go to other MMB page or run script.

Use the page name to move to a specific project page.
Go To Page 2

To jump to first, next, last page or back to previous page (in page history), use these four keywords -**first**, **next**, **last and back.**
Go To Page 2

To run a specific Script object from current page:
** Run Script1**

To run a specific Script object from another page:
** Run Script1**

From **4.9.8** there is a new exciting option to run the MMB script from HTML/JavaScript code! This open whole new world of option, because now you can not only run the MMB scripts from MMB but you can also send the variables from HTML/JavaScript to MMB or even call the MMB scripts after a specific action/event in JavaScript code!

To run the MMB code from HTML code (in MMB Browser) use the following syntax:

Loud script parsing - in case of error in MMB script, you will receive a message box with script error:
** Run Script1**

Quite script parsing - in case of error in MMB script, no error will be generated. Be careful about using this option!
** Run Script1**

..where **"** is an "escape" sequence replacing the **"** (double quote) character. The double quote written in escape sequence is then taken as a text and not as part of HTML code. For some more HTML "escape" character, see this page:
http://www.html-reference.com/Escape.htm

In cases you wish to pass more than just one MMB parameter on single link click, you can do this by adding **%0D%0A** (Carriage Return + Line Feed) sequence. The code should then look like that:

Fill MMB Variable and show MessageBox

As mentioned before, the **scriptcode** can not only be run from HTML but also from JavaScript code! Say "Good Bye!" to all nasty tricks (like using clipboard) to pass the JavaScript variables to MMB! The syntax of calling **scriptcode** from JavaScript is a bit messy, but it's not that complicated as it may look ;)

The basic rule is that each **quote** and **double quote** used by MMB script must be started with backslash character. So instead of using just **"** (double quote) you have to use **\"** (backslash + double quote). Although the JavaScript code with MMB code sequences may look strange, it's probably much easier to read and understand than the HTML escape sequences.

Here is a sample passing the JavaScript string variable to MMB variable:

// this is just a JavaScript string variable
JavaVar='HelloWorld';
// this passes the JavaScript variable to MMB

window.location.href= 'scriptcode1:test$='\''+JavaVar+'\'';
...where the red marked parts are MMB parts of code. The rest of text is just JavaScript syntax.

Description of individual parts of code:
window.location.href is a JavaScript equivalent of <**a href="url">link** action in HTML.

scriptcode1: calls the MMB script from Browser object.

test$='\''+JavaVar+'\'' is just a combination of MMB **RunScriptCode** command, JavaScript **JavaVar** variable and some backslashes ;)

After calling the above mess of quotes and backslashes, the MMB project will receive this:
test$='HelloWorld'

And as in case of HTML, if you wish to run more than just one MMB script line, you need to use the previously described **%0D%0A** (Carriage Return + Line Feed) escape sequence.

// this reads text from HTML form input:
var InputText = document.form1.formtext1.value + '';
// this line of code fills the MMB variable with above obtained variable, load the variable in MMB Text1 object, Refresh the screen and finally show the MMB Message box.
window.location.href='scriptcode1:Input$=\''+InputText+'\' %0D%0A LoadText(\"Text1\",\"Input$\") %0D%0A Refresh(\"\") %0D%0A Message (\"This message is created from JavaScript\",\"HTML Input: ' +InputText+ '\")';

All the above HTML/JavaScript tricks can be seen in the **html_mmbscript.mbd** project.

Tip 2
You can use Embedded html files. If you use it don't forget that you have to embed all the images and related pages your html file use or link to. Embedded files can't be in different directories so all html files and images must be in the same directory.

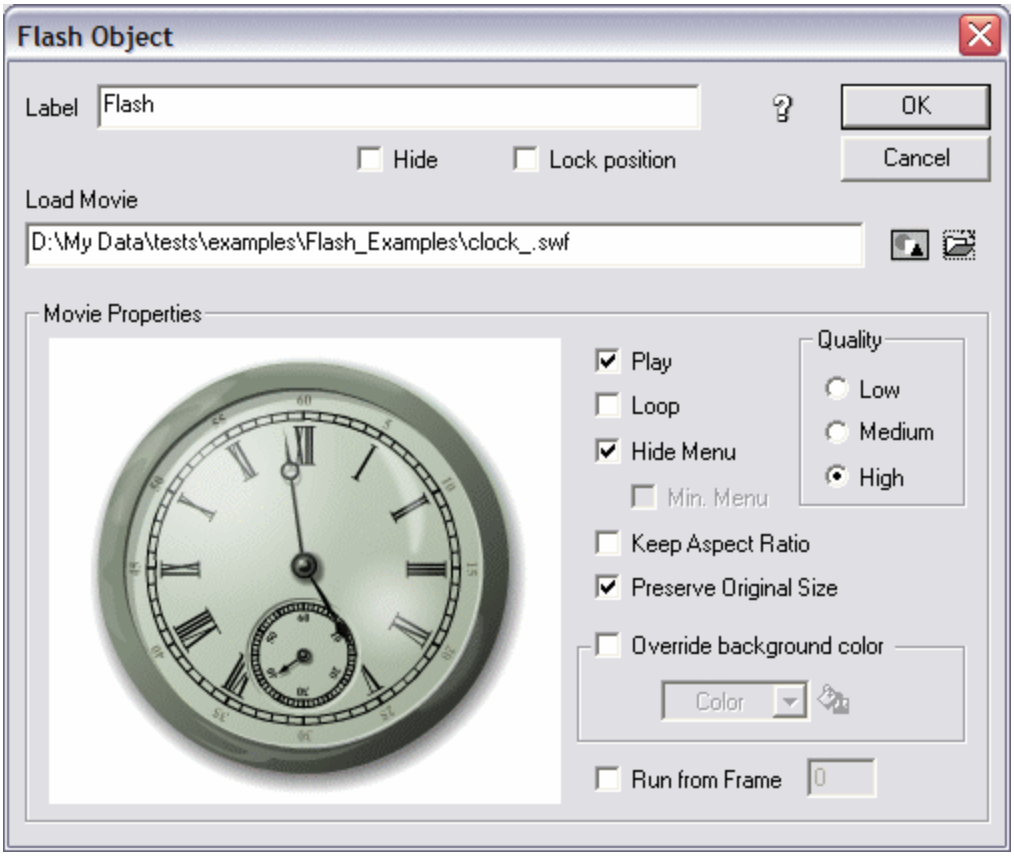
!!! A better solution than just using the htm/html files with separate linked Images seems to be an IE option to save the entire HTML page with all images or flash movies into a single "Web Archive" (mht) file. In IE you can save entire HTML page into a single MHT file via menu File..SaveAs. Then just embed that web archive file into your project or leave it external, if the file is too big for embedding. Then you can load that web archive via HTML properties (in a design time) or via **Browser** script function.

Note:
Browser Object needs IE 4.x-5.x installed on users machine. If user doesn't have IE, then the object will launch default html viewer.

LIMITATION! The TAB key to jump between the HTML form fields cannot be used in MMB HTML object. It's because the HTML object is just an ActiveX object insert in MMB project (parent) window. At the moment, we have no solution to this issue.

7.22 Macromedia Flash Object

 The Flash Object allows you not only to load and play Macromedia Flash animations, but with this object, you can control all the MMB scripting actions directly from the **Macromedia Flash Action Script!** Therefore, you can use your own Flash file as a powerful scripting addition to the basic MMB scripting – e.g. for parsing strings, advanced math, adding some new UI elements, like a check/option boxes, combo boxes, spinning objects, tables and charts, etc. However, this option requires at least a minimal knowledge of Flash development and especially the Flash Action Script language.



In The Flash Object settings dialog you can specify the Flash file and some usual Flash options like a quality level, if Flash movie should be run after loading a page, running the movie from a certain frame, enabling/disabling the right click menu or change the background color of movie, etc.

The Flash movie can be specified in a "Load Movie" input field by an absolute or relative path (by <SrcDir> or <SrcDrive> commands) to the files on your harddisk, however you can specify also a "live" Flash movie placed on Internet (http://www.myweb.com/yourflash.swf). In addition, you can use the Flash file embedded to the project.

Play: Simply run the Flash file after page load.

Loop: It will cause Loop the Flash playback.

Hide menu: completely disable the Flash right click menu.

Min. menu: switch the Flash right click menu to minimized state with only the basic Flash controls. This option is disabled if **Hide menu** option is enabled.

Keep Aspect Ration: It will peserve the aspect ratio of loaded flash file.

Preserve Original Size: Flash file will be loaded in its original size.

Override background color: Replace the Flash background color.

Run from Frame: Run the flash from a defined frame.

For Flash related scripting commands **look here**. Some Flash example projects can be found or

7.23 Binding Object

Binding Object is a way how to easily put exe file into MMB - it will become a part of the player: for example flash movie with flash projector player, your installer, notepad with text, another independent MMB project etc...
See example **binding.mbd** included in MMB samples, shows notepad and regedit inside page. Tested on Win2000 and Win 98SE

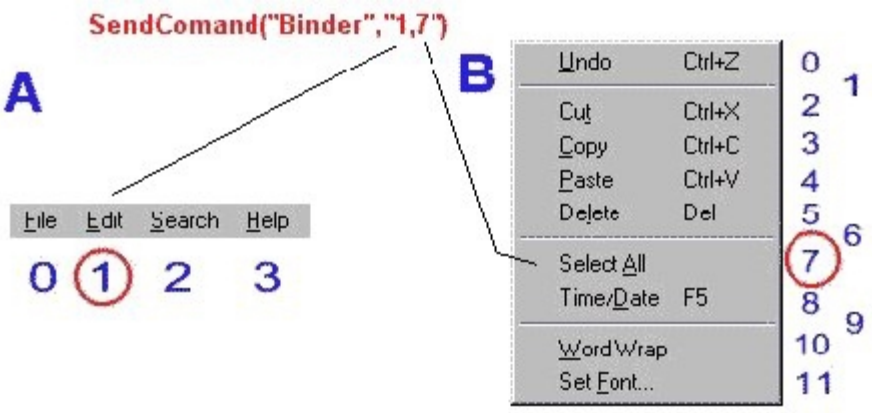
Note: Not all exe files can be binded, but many could (for example you can even bind CompactDraw or PhotoBrush....) Some exe files may be unhappy to be binded and may freeze your system - save often. Some exe files can't be binded at all or shows garbage (Winamp). You can't call just the file to open associated exe yet. You need to specify the executable and then the parameter (file). You can hide Menu, status bar or toolbars from the binded applications.
Binding objects has many advantages - you don't need to create a special version of your exe file, no ActiveX, no DLL..
You can have many BO on one page, you can have some even in Master Top Layer...

Communication Between MMB and Binding Object

MMB can send Menu Commands to the binded Object. Menu Command is the index of Menu as it appears on the original software. The Menu Index is zero Based.

SendComand("Binder","1,7")

Example: Binding notepad.exe



You need to count the menu separators as well.
In case of Nested Submenus, you can use the third optional parameter C

Sending String to Binding Object

To send string to the object (here Notepad), we can use MMB command - Clipboard. First, we send the string to the clipboard, and then we invoke Select All and after that Paste command in the Binding Object. Click the Send Text button.
See the script below:

C$ = 'Hello from MMB!'

Clipboard("SEND","C$")
**select all
SendCommand("Binder","1,7")
** Paste
SendCommand("Binder","1.4")

With the same idea, we can get string from the object.
First, we invoke Select All, and then Copy and then we get the string from clipboard.

**select all
SendCommand("Binder","1,7")
** Copy
SendCommand("Binder","1,3")
Clipboard("GET","C$")
**display it
LoadText("TextDisp","C$")

(Un-Bind) the program.
- **Terminate Process**-default (brute force method which doesn't give the program chance to do anything about it)
- **Send Close and wait**. This is softer method and it gives the application a chance to do closing stuff (for example ask if you want to save changes etc...) Also, if application doesn't like the first method, try this one.

Un-Bind on Page Exit option (default). If checked the Binding exe file will be terminated if user goes into other page. That means any time you go to the page with binding object the exe will start again. This is probably great for applications as Flash animations etc... you don't want them to run if you are in other pages.

Hide Menu - Hides the menu of the binded application. If the app has nonstandard menu size or you need to hide larger height, use advanced options.

Hide Sizing Grip - Some application have Sizing Grip (like for example notepad.exe). You can mask it with this option so user won't be able to resize the binded application.

Advanced Properties
Help To locate window:

Some complex application may have many "Parent" windows - MMB may not always choose the right window so you have additional options which window of the binded application to bind in MMB. You can specify which string the main caption must have and which it must not have. If empty - then it doesn't apply.

Client main window Caption must have string
Client main window Caption must NOT have string...

Note: In simple application (like notepad, Flash player etc. you don't have to worry about this. You should use this only if MMB choose wrong window from the application to bind.

Wait Before Binding - some applications may take longer time to start and MMB may not catch the window (it try to find the window too early, but the application does something else at this moment and don't yet have window created) This will tell MMB to

wait before binding.

Additional Cut from Top or bottom - you can simply Crop the binded application from top or bottom. Great in cases when you need to hide toolbar or statusbar for example.

7.24 MMB Plugins

 Plugins are windows dll build with MMB SDK (software development kit). Plugins expands the functionality of MMB.

Plugins can behave the same way as other objects. This section is for developers. The plugins are compiled using VC++ and the MMB SDK. If you are user of some MMB plugin please refer to the documentation included with the plugin.

Note: The MMB plugins are dll's and plugins made for MMB. Other plugins like plugins for Photoshop will be not working !

To load plugin: Menu Object - Plug In - draw plugin rectangle on the work area

The rectangle says: **Plugin (Empty)**, double click on it and from the properties window you can load the dll.
Dll can be external or embedded. If the plugin DLL stays external, you don't have to write <SrcDir> if the plugin will be in the same directory as the autorun.exe or in the subdirectory Plugins.
After you load the DLL, MMB will ask you if you want to use embed the plugin or not. If you select yes, the plugin will be embedded.

Plugin can have its own **properties.** If this is the case of the plugin - the Plugin properties will be enabled.
Also some plugins require one, two or three bitmaps to be loaded - then the Image 1...3 will be enabled.
The plugins for MMB are in the sub-directory Plugins. By default there is only one sample plugin **TenBlobs.dll**

Script Action: Plugin can have also a script on click assigned to it. If user clicks on the Plugin Object this script runs.

Some plugins (like standard Dll's) doesn't require any MMB interface and they don't display anything as objects.
For example dll which has its own separate window or dll, which play some special sound format etc..

TIP1:
If you would like to develop your own plugin, take a look at **this page**.

TIP2:
You can load any system dll without MMB interface as a plugin (use **PluginRun** ("PlugIn","GlobalVoidFunction")), the properties will say Unknown Interface but you can still run a global void functions from it. The first parameter is the loaded Plugin object. The DLL doesn't have to be made with the SDK.

There are few Functions in the script for such dll's or plugins. A detailed description of MMB Plugins and a description of Plugin related scripting commands, can be found **here** .

And where you can find the Plugins? There are few basic plugins in MMB\Plugins folder. But you can find much more plugins for example in **MMB File Library** and don't miss

also this **MMB Plugins** discussion.

If you don't know what plugin you are looking for, we would strongly suggest you to look at . This great help compilation contains list of all important MMB plugins available for MMB, along with detailed help for each plugin! **This is definitely something you don't have to miss!**

Top Level Intro

This page is printed before a new top-level chapter starts

Part

8 Pages

8.1 Introduction to Pages

A page is used to contain a group of objects and their functions. The groups of objects and functions are arranged on a page so that the page can perform a chosen operation. Therefore, you can think of pages as a way of organizing your program into separate operations needed by your program to do its job. You can use pages as a way of jumping around from one operation to another in your project.

Although many projects can be built using only one page, using multiple pages can make the building of the program easier and more logical to understand. This can also be handy if you want to update your project at a later date.

The disadvantage is that each page added can add to the size of your final project, making it less suitable if you want to distribute it across the internet.

Multimedia Builder also has two special pages that provide added functions to your project:
Master Page: objects on this page will show on all pages (under that pages objects!)
Master Top Layer: objects on this page will show on all pages (over that pages objects!)

(Written by Rodd)

8.2 Page Properties

 A page is the place where you design your multimedia project. Each project could have one or more pages.

To access the page properties, double click on the page icon on the bottom of designer or select from menu Page - Properties

- **Label**

  It's the name of the page.
  A label is a unique string of text, which represents a page, or can be called upon by an action.

- **Background**

  A page can have a solid background defined by a color. It can also be a bitmap background.
  To load a Bitmap as a background, press the Load Image button.
  If the image is smaller than the page size, you can Tile the image.
  Tip: You can find some pretty interesting background in the directory Background. They can be tiled for exciting effect.
  You can copy all graphic properties included in the first page to any other subsequent pages.
  The background graphics are very important.
  Put as much graphics as you can into the background. Redrawing the background is much faster than any other objects.
  You can combine objects with background:
  Select the objects you would like to combine.
  From the menu Arrange, select Combine – Objects with background.
  From the next dialog, select Delete source objects, and Use this image to create a new background.

  A page can use the background included in the first page, or from the Master Page.

- **Background music**

  MMB can play different background music on each page.
  The music can play across page boundaries. If the next page has no background music, the last one will continue playing.
  You can loop music. You can find music on the Internet, or on CD's like Corel Stock Music Library.
  You can select to play Audio file on page start. This doesn't go through multichannel sound.
  You can use embedded waves as well – just write the Internal Name of the embedded wave file.

- **Digital Audio**

  MMB can play audio tracks from CDs. A track can start playing as soon as a user enters a page.
  Use this function to build your Mixed-mode CD where the first track is a data track (your player and MBD data file and other data), and the other tracks are audio tracks.
  Mixed mode CDs are widely used for games and multimedia, because playing audio tracks doesn't take much resources like playing WAVE files.

- **Script**

You can trigger any actions or change variables using **Script language**.
Note:
Background music plays through the DirectSound channel. It's mixed with other sound effects without interruption.
The user must have DirectSound, which is installed on almost 95% of the systems. (In the future, it will be 100%, since Win95 SP 2 and Windows98 have DirectSound by default.)
MMB allows you to create Distribution Files, and can detect the system of the user. In the worst case (user without DirectSound), a user will not be able to hear background music, but he will be able to hear Sound Actions, which are played through standard audio output. See **Check Project and Distribute Files**.

There are also 2 separate special "pages" Master Top Layer and Master Page.

Note: You can also add **Comments** to your project.

From the menu File – Comments...
You can add comments to your project.
If you want this window to open automatically after you open the project in designer, check the check box «show next time on Open».

Page Properties

(Written by Rodd)

You can alter the properties of the current page by selecting 'Page' and then 'Properties...' from
the MMB drop down menu; or you can simply 'double-click' the pages 'icon' in the page selection
area.

Label: This is the name of the page. By default the first page of the project is labeled 'Page 1'.
You might like to change the 'label' of the page to be more descriptive of the operation of the
page. The page label is used by MMB to locate the different pages of your project, so that it can
switch from one page to another.

Background: This allows you to choose the background graphic or background colour of the current
page. You have the option of using the background graphic/colour that was chosen for the
master page, OR a specific graphic/colour for the current page only.
From Master Page: the current page will simply show the background graphic/colour that was
chosen for the [Master Page].
Color: choose a colour from the drop down list as a background for the current page.
Image: allows you to choose a graphic file from disk to use as a background graphic. If the
graphic is smaller than the current page, you can also choose 'Tile' to fill the current

page with the
graphic.

Play audio track from CD Audio or Mixed-mode CD
This allows you to select a music CD track as background music for this page. You have the option
of selecting which CD track you want to be played. Please note that when referring to a Mixed-
mode CD the first audio track will be track 2 (since track 1 will not be music but will be data)!
Note: the music doesn't automatically stop when leaving the current page.

Script:
This will allow you to write a script that will be run automatically each time the page is accessed.
Please note that 'plugins' tend not to function correctly when run directly from this page script. It
is advisable to have the page script run a separate script (using the 'ScriptTimer' command), which
runs the plugin.

Page Transition:
This allows you to change from the current page to another page using a special graphical/optical
effect. There are many graphical effects to choose from (including special page effect plugins), and it
is best to experiment with the different transitions to see its effect.

Background Music:
This allows you to select a 'wave', Midi, MOD, or MED file to run as background music for
this page. You can choose to have the music continually replay by ticking 'Loop (WAV, OGG)'.
Note: the music doesn't automatically stop when leaving the current page.

Select 'OK' to accept the changes or 'Cancel' to discard the changes.

8.3 Master Page and Master Top Layer

 &  **Note**: This is a very important feature for project with many pages.
Don't copy common objects to all pages, which is a waste of space.
Instead, put common objects on the Master Page or Master Top Layer.

What's the difference?

Master Page
Will appear as a bottom layer on each page. The objects on the page will appear all the time on the top of the objects from the Master Page.
Tip: Great to insert your artworks. You can also insert interactive objects, but make sure no other objects cover them.

Master Top Layer

Will appear as a top layer on each page. The objects on the page will be behind the Master Top Layer objects.
Tip: Great for active objects like menus, buttons, windows, etc.

Version 4.4 has the ability to Show or Hide objects on any page from any page (Master pages included)

This is done by using syntax:

Page::Object

For Example to LoadText in Text object on Page 2:

LoadText("Page 2::Text")

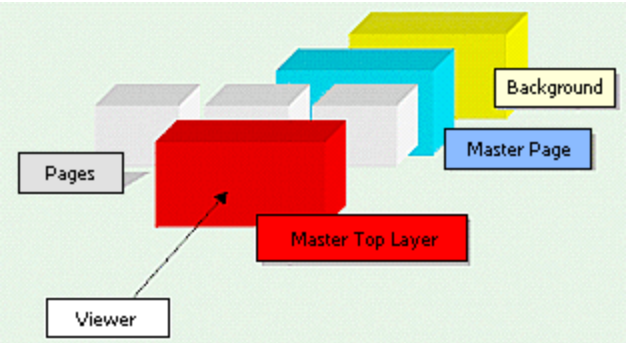
!!!This may not work with **windowed** objects like HTML, ListBox, Flash, Video or Binding, because these objects are created/destroyed on page Start/Exit, in other words, these objects doesn't exist if the page in not visible. If you want to control these objects from ordinary Pages, add them to Master Page/Layer and then hide them on the other pages where they shouldn't be visible..

The Master Page and Master Top Layer has the syntax:

Master Page::Object
Master Layer::Object

Hide("Master Page::ListBox")

This diagram represents some elements in a project relative to one another.



8.4 Master Page Properties

 A Master Page can have a background. (The same as a "regular page".)

To use a Master Page background on another page, check **Background – From Master Page** on the desired page properties.
Otherwise, the page will have its own background or the one from the first page.

Top Level Intro

This page is printed before a new top-level chapter starts

Part

9 Sound

9.1 Sound Actions

 Moving the mouse over the object or clicking on the object can trigger a sound action.
The supported files are .WAV files.
Wave files are external files and are not included in the MBD file. You have to provide the relative paths to the .WAV files. See comments in **External Commands**

The On Click section can play instead of a Wave file the Audio track from Audio CD or Mixed-mode CD.
To use this function you will write instead of wave file name CD Audio command.
Commands:

CD: TrackNumber	will play desired track number. Note, if you're using Mixed mode audio the audio track will starts from 2. (example CD:2)
CD:STOP	Stop playing
CD:PLAY	Start Playing – use after CD:PAUSE or CD:STOP
CD:FW	Play next track
CD:BW	Play previous track
CD:PLP	Play or Pause (the same action can play or pause playing)

You can use Script to do the same – using CDAudio commands.
As you can see you can build your own little CD player with Multimedia Builder.

You can use also embedded wave files by writing the name or by clicking on the Embedded wave button. The **Embedded Wave Dialog Box** will appear where you can add new wave or select existing wave.

9.2 Embedded Wave

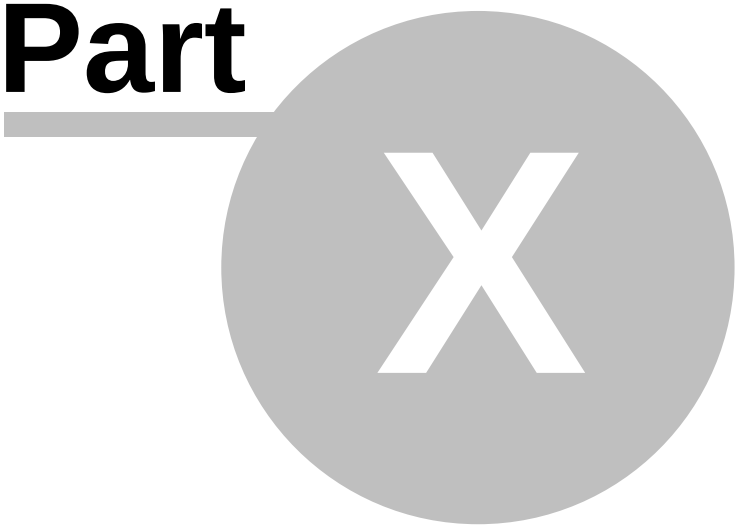
 An Embedded wave allows you to add small sound into the project without having external files.
This is great for Actions – sound on button click etc… Don't put music there – it will make the project file very big. The big files should stay external!

How to use it? Just load some wave, give it a name (**without extension**) and then use this name instead of the path in your sound action or PlayWave script commands.

For more details about usage of embedded wave files look **here**.

Top Level Intro

This page is printed before a new top-level chapter starts



10 External Commands & Page Actions

10.1 External Commands and Page Actions

 You can use External Commands to launch an executable when your viewer clicks on an interactive object (With Enabled actions)

For example, you can:
Run another project

This is a special action. It can run another MBD file project in the same window (like another page) or in a separate window.
If the Second parameter is **NEW_WINDOW,** the MBD file will be in a separate window.
If the second parameter is **THIS_WINDOW** the new MBD will be in the same frame – the user will think it is just another page.

If your project has a lot of graphics, it's a good idea to separate pages into different MBD files.

Multimedia Builder is designed primarily for small projects – like Autorun Browsers etc. You can create bigger projects, but you have to have different pages or groups of pages in a different MBD file, otherwise the MBD file will be very large.

Do not provide a fixed path unless you're certain that your application will reside in a specific directory on your viewer's system.
Provide a relative path. The path is relative to the location of the MMB player on your viewer's system.
Use the <SrcDir> or <SrcDrive> to tell MMB to use the actual directory where the player will be located on your viewer's system or actual drive.

To view the complete list and description of <> constants look **here**.

TIP:
You can change all paths to the relative paths at the end of development using **Path Replace** in Project menu. During development, you can use fixed paths just to make it run. After you finish you can run Check & Distribute files in Project menu and you will see a list of errors if you are using fixed paths.

Example:

External Command Example

The player is named AUTORUN.EXE and it is located on a CD inside the **INSTALL** directory.
Let's say the CD-ROM is drive **D:** on the viewers system.

The command: <SrcDir>\BIN\INSTALL.EXE

will be translated into: D:\INSTALL\BIN\INSTALL.EXE

And <SrcDrive>\VIEWER\VIEWER.EXE

will become: D:\VIEWER\VIEWER.EXE

Page Actions
Only for *Goto Page (Label)* you have to specify the Label of the page (example: Page 1)

External Commands can Run a program, browse the disc, jump to a web page (you need to specify http:// and URL), send an E-mail through default E-mail client (you need to specify mail to: and the e-mail address) and much more…

10.2 Actions

Any one object can trigger (at the same time) four different actions:

External Commands and Page Actions

 If the user clicks on the object, this action can :

- Start an installation, run any external program, jump to your web page, send an e-mail, open a document, display a help file, run another MBD file, Browse the CD, etc…
- Move between pages of the MBD project
- Exit

Interaction with other objects and Video

 Object can interact with other objects on the same page. (show/hide object, play video) on user actions:

- Mouse move over the object.
- Mouse click on the object.

This action allows you to add a professional look to your application (buttons glow when mouse moves over …)

Sound Actions

 Object can play a sound if mouse moves over the object and/or another sound if mouse clicks on the object. Multimedia Builder supports multiple channels, so you can still play a loop in the background without interruption.

More Actions (Scripting)

 An advanced scripting option. Meanwhile the previous actions were intended for creating the simple reactions on the basic events (like a mouse over, or mouse clicks), the "More Actions" is useful for experienced MMB users to writing advanced scripts to better controlling MMB and creating many interactive effects.

10.3 Interaction with other objects and Video

 An object can interact with other objects. For example, a glow appears around a text when the mouse moves it.

- **Moving mouse**

  Show/Hide
  Shows the object specified in the Object list box and then hides it if the mouse is not over the object anymore.
  Show/Fade out
  Instead of simply hiding an object, the object will slowly fade out. The object must only be a graphic object with Alpha transparency enabled.

- **Mouse click on object**

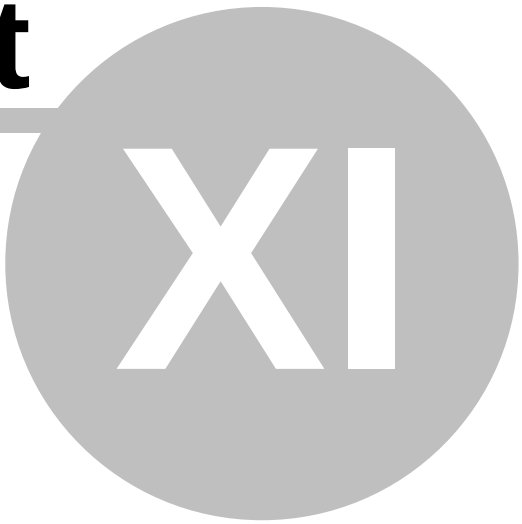
  Show/Hide
  Show and then hide an object when the user releases the mouse button. Video actions on Video Object - Play, Stop, Pause Skip, 2x slow or 2xfast
  Show
  Show an object or a group. This will stay until an object hides it.
  Hide
  Hide an object or a group. This will stay until an object shows it.
  Invert (Show-Hide)
  Show or Hide an object or a group. If the object is hidden, this will show it and vice-versa.

Clicking the mouse on an object can trigger another action after the first one.
For example, the first action can hide a whole group and the second action can show a single member of the group.
For two actions with a mouse click and nested grouping, you can create almost any effect.

Top Level Intro

This page is printed before a new top-level chapter starts

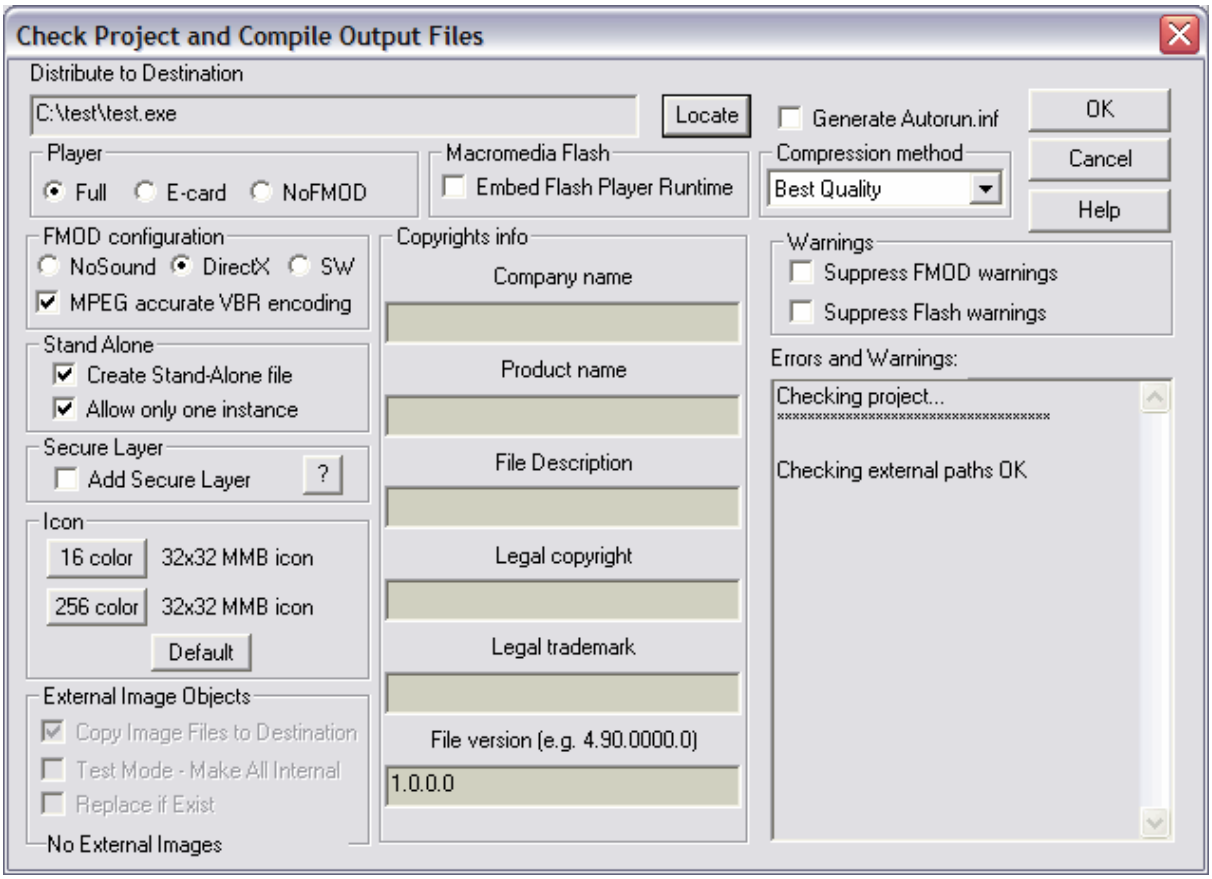
Part



11 Finalize Projects

11.1 Check Project and Distribute Files

 At the end when you finish designing the multimedia application, you would like to create distribution of your files.
From menu **Project** select **Check & Distribute** (Same as from menu File - Compile)



Distribute Project

You have to select the location where you would like to copy all necessary files. Select the location with the Locate button. The standard windows dialog will allow you to create a new directory if you'd like.

Player:

- **Full**

  This is the default player - it has all the libraries included. In most of the cases, you will use this one.

- **E-card**

  E-Card is a smaller stand-alone executable, which you can send to somebody via e-mail.

The most important thing for e-card is the size, because it takes a while when you download files with e-mail client.
Some of the functions of the standard player were taken off.

The Restrictions:

- No sound
- No full screen background
- No plugins
- No Replace Image or Image Resize/Rotate functions
- Some other small restrictions.....

E-card should work on all Windows95/98/NT systems the same way as standard Stand Alone player.
From version 4.6 E-card is not dependable on any windows system dll's.

- **NoFMOD**

Almost the same player as Full but without the FMOD audio library. If you don't want to use sound in your projects and/or you need to decrease the application size as much as possible (but with possibility to use the plugins and advanced Image related functions), this is probably the best option for you. BTW, with this option you can use a 3rd party Sound engines provided via MMB Plugins.

Checking the Project

This tool checks the project against fixed paths for WAVE, AVI and other external files.
(See comments about relative paths in ***External Commands and Page Actions***)
You can create a distribution even if you have errors. You will be able to play the application, and you will only have problems with external files with fixed paths.

Resizing bitmaps and applying effects will create temporary bitmaps inside the project.
Those bitmaps are not necessary for distribution and removing them can significantly reduce the size of the project.

Compression
For distributing files on Internet or if the requirement is the size of file you can use powerful compression.
If you don’t have the problem with space, you should select uncompressed – the project will load much faster then with compression.
The compression will change the quality. The significant quality change can be seen after Good Compression.
Only one compression doesn’t change quality – ZIP (100% Quality)
If you want to compress files, but you don’t want to create *exe files (for example to send mbd to user gallery on Internet) you can use from menu File – Compress & Export instead.

Stand Alone
Standard MMB output is the player and the MBD data file.
With Stand–Alone checked, the compiler will produce only one .EXE file – the MBD data file will be linked inside the .exe file.

Important Note:
Check Project and Distribute files doesn't copy the external files (WAV, AVI or other externals) to the distribution location. (They could be very big!)
You will be informed after you press the OK button.
In order to finish distribution of the files, you have to press OK. The result message will pop-up.

The Secure Layer

This will process the mbd data in such way that no Text or Script would be visible if you look at it in the Hex editor. Also, the file can't be loaded back to the designer (no password will help!) so it can only be played. However, note that the loading of file with Secure layer would need more time and memory that without this layer so you should use it only on necessary files.

The same option was added to **Compress and Export** to easy create mbd files, which can't be viewed in Hex editors and loaded in designer.

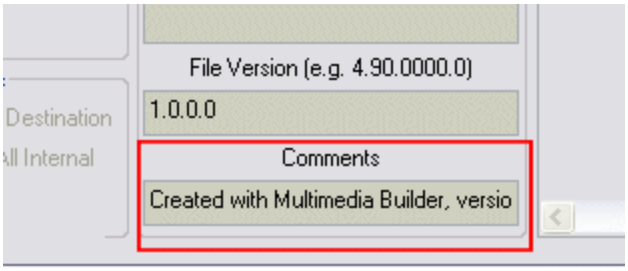
Icon

Here you can set the application icon, which will be then displayed in program title bar, application taskbar or the Alt-Tab menu. There is currently no way to use XP/Vista icons in Compile dialog. But you can replace the 16/256color icon after compiling the project. There is a tool to do this right in MMB installation folder called "". Simply run this tool, chose your compiled project, your XP/Vista multi-icon and simply press [**Replace**] button.

Copyrights Info MMB4.9
This "Copyrights Info" allows you to insert your own file version information and comments to the final executable. For example, you can put your copyright for the contents you created, comments about used materials or publication version number. It resolves some legal issues if your customer doesn't want to put your copyright on the project pages but you still want to have your copyright or name there. Many other authoring tools didn't allow you to do this. You can access this option only in the Check & Distribute dialog box. Remember, that this work only if the "***Create Stand-Alone file*** " option is checked. You can put up to 35 characters per field. If you want to show File Version tab in your executable (or anyone else *.exe file), then select the executable, press ALT + ENTER and click on a Version tab.

NOTE 4.9.8: With purchased "Comment Unlock Code" you can change also the Comment field, which by default contains this string..
"**Created with Multimedia Builder, version 4.9.8.0**".

After purchasing the code (here) and entering it in "About MMB" dialog, you will see a new Comments field in compile dialog..



Generate Autorun.inf
If you creating autorun menu for CD then you need to generate autorun.inf and put it on CD together with the exe, mbd and all other files. Autorun.inf and the MMB project (exe or player.exe) must be in the root of the CD.
If you creating other than CD projects then you don't need autorun.inf generated.

IMPORTANT! Keep the exe filename in 8+3 format and **without empty spaces**! If your autorun doesn't work, the empty space in program name (like "**my program.exe**") is the most common cause of this problem! If you still wish to use empty spaces in the program file name, you need to edit the Autorun.inf and enclose the program name in double quotes.
so instead of this..
OPEN=autorun.exe
use this..
OPEN="autorun.exe"

Warnings

Suppress FMOD warnings
Enabling this options will suppress all error and warning messages generated by FMOD audio library (e.g. "Unsupported file format" or "Sound Card is not installed").

Suppress Flash warnings
Enabling this options will suppress all Flash player information/warning/error messages generated by Flash player (including the Flash player installation..in other words, this will make the embedded Flash player installation completely silent).

Embed Flash Player Runtime:
This is a special option for embedding the Flash runtime files that have to be installed on the user's system. According the Macromedia's website, the current penetration of the Flash player is about 98% of all users already connected to the Internet. It's probably because the Flash player is installed with most of the current Internet browsers, like a MS Internet Explorer or Netscape Communicator. Therefore, you don't need to embed the Flash runtime into your project in most cases. On the other hand, this option is always useful in cases when you are not sure about the version of Flash player already installed on user's system or if the Flash player is ever installed on the user's system. With MMB4.9 is shipped Flash player version 6. If you don't check this option and the Flash player will not be installed on the user's system, or the version of already installed Flash player is older than expected, then application will try to connect to the Macromedia web site and will download and install the most recent version of Flash player.

11.2 Paths Replace

Menu "Project" - "Path Replace"
Wave files, AVI files and other external files (setup files etc..) are not part of the MBD file (they could be very big) and they will stay separate.
It is very important they have paths relative to the player on the users system, not fixed paths.
(See comments about relative paths in ***External Commands and Page Actions***)
This tool allows you to change all paths in the project at once for Commands, Wave files Objects and Script.

Click on an item to get more information:

- **Search string**
  The string that will be replaced. Search for common strings, for example C:\WAVES.
- **Replace with**
  Replace strings with this for example <SrcDrive>\WAVES
- **Replace whole path**
  Whole path to the file will be replaced. Great if you don't have sources with some part of the path common. (If you use sound files from all disks on your computer but on distribution disk have to put them on one directory)

  Example:
  In project you use Wave files
  C:\Windows\sound.wav
  D:\MSOffice\Ppoint\beep.wav

  We put *Replace With* string: <SrcDir>\WAVE
  Check the *Replace whole path* and press *Replace All*
  The result will be:
  <SrcDir>\WAVE\sound.wav
  <SrcDir>\WAVE\beep.wav

  Now you have to copy all wave files to your distribution into the directory WAVE.
- **Replace all**
  This is the actual button to make the replace action. You cannot cancel this operation, so it is a good idea to save the project before.

Note: The commands, waves and video files are separate files because it is good idea to put them separately on the distribution disk. See

Example:

Separate Files Example
All external files (Setup, etc.) will be put in directory <SrcDir>\BIN
All wave files we will copied to <SrcDir>\Waves
And all videos we will copied to <SrcDir>\AVI
The <SrcDir> will be replaced at runtime with the actual directory of the player.

11.3 Text Replace

Menu Project - Text Replace
With this Tool you can simply replace text items on each page.
You can replace the caption of Text button, andt the text on Text Object and Paragraph.
Also it allows you to replace a ToolTip on any object where tooltip was defined.

Note: Remember to press Apply button after you make a changes.
Tip: This tool is excellent to make a language clone of your project , or fix an error made on multiple pages.

11.4 Compress and Export

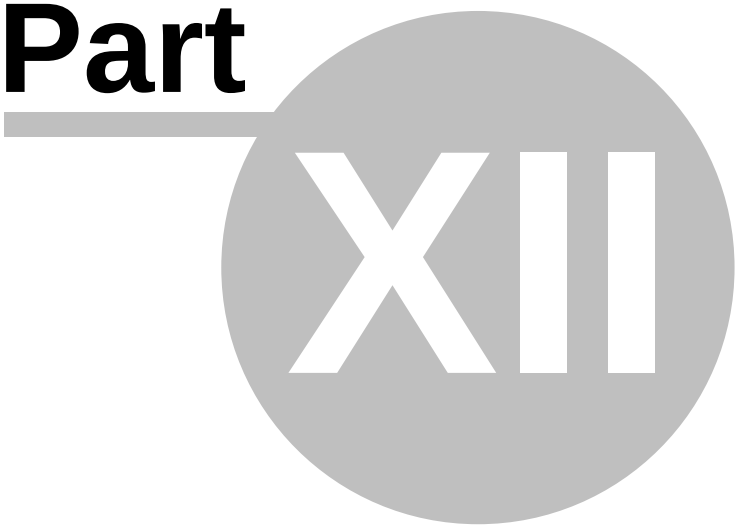
Menu File-Compress and Export
Sometimes you just want to export the MBD project without actually creating the executable player. (For example, you want upload your project to the discussion board) This allows you to chose the compression different than the default compression.

Password
You can chose a password which prevents to open the file in designer. The file will play in player without limitation.
This options can be use for example if you creating a large presentation, which is divided into few mbd projects. With password you can prevent from seeing this file in designer.

The Secure Layer
This will process the mbd data in such way that no Text or Script would be visible if you look at it in the Hex editor. Also the file can't be loaded back to the designer (no password will help!) so it can only be played. However, note that the loading of file with Secure layer would need more time and memory that without this layer so you should use it only on necessary files.

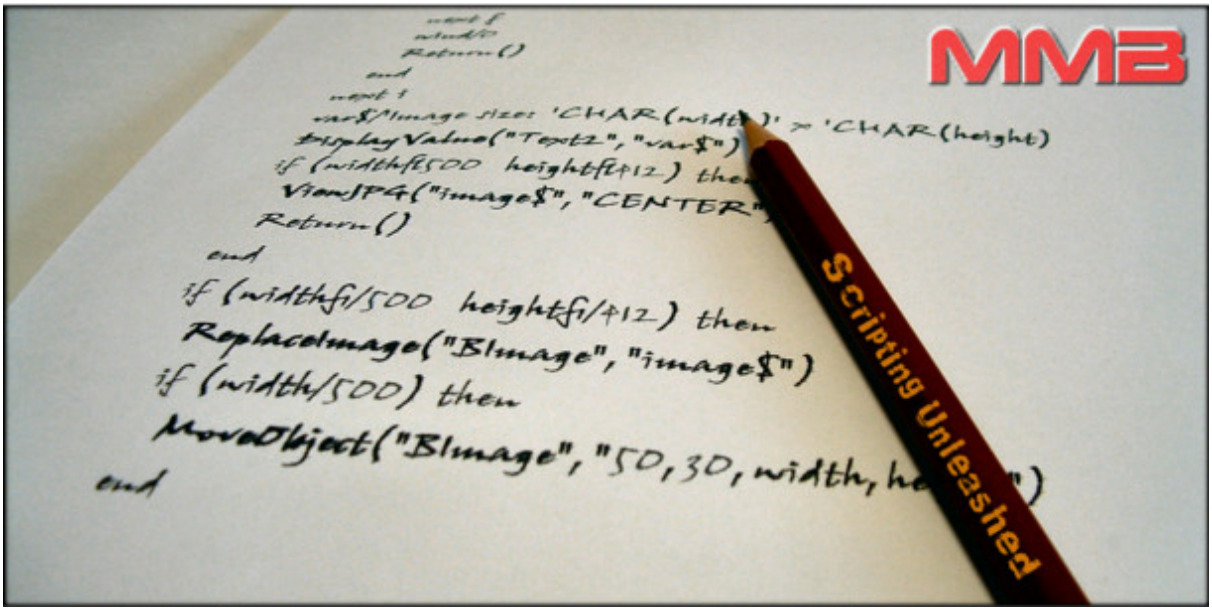
Top Level Intro

This page is printed before a new top-level chapter starts



12 Scripting Unleashed

12.1 Introduction



MMB Scripting: Table of contents

SCRIPT FLOW FUNCTIONS
- **Introduction**
- **If..Then statements**
- **For..Next loops**

MMB SCRIPT COMMANDS

Create and Delete Objects in a Runtime

Global Object and Project Functions

Project Commands

Object Commands

Time & Script Run Commands

Dialog Box Commands

System Commands

Audio Commands

Image Commands

Print Commands

Video Commands

MCI Commands

Flash Commands

ListBox Commands

SongList Commands

TTS Commands

Browser Commands

Image Matrix Commands

AGif Commands

Script Manual Credits

MMB PLUGINS

PlugIns Operator Manual

12.2 Programming?

One of questions asked multiple times in MMB community: is MMB scripting - programming ?

And once for all times, let's give the answer: YES !

You become a programmer the same moment you write one line of code in MMB.

It's nothing unusual, indeed. When you set VCR to auto-record your favorite show on TV - programming is done. Writing scenario for a movie is programming (of actors & objects).
Even telling your children how to write their very first letters or instructing someone how to swim or skate - no matter how unrelated to computers it is - you're programming, all the time ! As a matter of fact, after all these centuries of natural programming, we just recently started to instruct machines what to do instead of us.

Take for example some everyday products:

They include instructions on the back, telling what you must do with the product:

So if these manufacturers can do some programming, nothing stops you from doing it:

And what's the main difference ?

You have much more powerful tool in your hands !

12.3 Scripting Basics

Yeah, just like with any language - there are some rules we must obey to use it.

Compared to human languages, MMB script is one simple & fast-learning language.

Basic Facts

1. Scripting is done using lines (rows): as you would type some story (for e.g. detective story about who killed the butler) on a type-writer, you'll in MMB case use keyboard to write MMB script story - and guess what: by default, one sentence per line :)

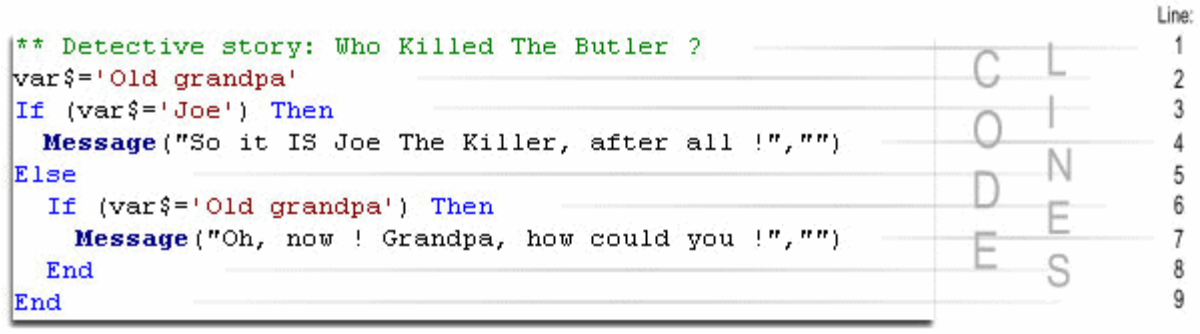
2. Sentence in MMB script story is called "code line": one line usually represents one command (there are just a few exceptions)

3. You don't have to remember command names ! MMB will help you using commands without knowing their attributes, and it'll provide help in more ways:

- *syntax completion*: MMB keeps watching what you write and if it finds identical command in it's list - completes code line with required syntax - now all you have to do is enter parameters specific to your project
- *right-click command list menu*: quick list of available commands is on your right-click mouse button - select command and move on !
- *script wizard*: using built-in command list in script editor, you really don't have to remember any command name - selection of command will show you it's explanation and enable entering of parameters
- *MMB help file*: every time you want extended explanation of some MMB script command, come here, to MMB Script Manual, and read all about it !

4. Syntax highlighting & formatting: MMB will automatically colorize command parts, making entire code easily readable. In addition to this, it will automatically arrange indent of code lines once you're finished with scripting, further enhancing code readability.

Preview of script lines



Here you see what these code lines look like - they're rows of commands that tell the story and depending on situation - perform assigned action.

In example above, story goes like this:

After gathering all suspects in one room, detective is ready for his final act.

- Detective reads key evidence, exposing the killer - "Old grandpa"
- Although he's already prepared with line accusing Joe for violent act..
- After reading key evidence, he screams:

Oh, now ! Grandpa, how could you !

And that's it ! 9-line story & Grandpa's going behind the bars !

Details you can notice on preview displayed above:

- names (Old grandpa, Joe) are colored red
- Detective's thoughts (if it's Joe / if it's Grandpa) are colored blue
- Detective's screaming (So it IS / Grandpa, how could you) are left in black color
- indent of every line is arranged to make story/code reading easier

12.4 Scripting Syntax

Commands are most important part of code lines. Using commands, you tell MMB what to do in line you're writing. A kind of electronic army : you're an officer and MMB is just a soldier, executing your ideas ! ;)

Of course - pay attention to **expressing ideas** ! As a difference from the real-life army where command like...

Down soldier ! Gimme 50 !

...is taken for granted, resulting in 50 push-ups performed by soldier...MMB would ask...

...50 of what !?...

Yeah, yeah, you already know how it goes when working with these machines - you gotta explain, specify, draw everything to 'em and they still dare to ask questions !

So let's see what we must do to keep MMB quiet & kickin'...

There are more kinds of commands.

Hey, don't run away just yet - they're basically the same ;)
Every command comes with unique name:

JoeIsMyName

...followed by both parenthesis:

()

...put together, every command at least looks like this:

JoeIsMyName ()

Are there commands like this above in MMB ? Yes.
Are there different commands than this above ? Yes.

Just like you can have vase with or without flowers - you can have commands with parameters or without parameters.

Here's a summary of **command styles in MMB**:

Command style	Command example
Without parameters	`CommandName()`
With one parameter	`CommandName("parameter")`
With more parameters	`CommandName("param1","param2")`
With variable parameters	`CommandName(variable1,variable2)`

These are just general command styles - read explanation of every command in particular to learn what it requires.

Parameters provide extra info required by some commands; is it natural to put flowers in vase ? Yes. Is it natural for some commands to require flowers... oops, parameters ? Yes ;)
Long live the diversity - there are many kinds of parameters:

Parameter style	Command example
Text	`CommandName("Hello, MMBers")`
Numbers	`CommandName("4329")`
Object labels	`CommandName("MyTextObject")`
Time/Percentage	`CommandName("60")`
File names/paths	`CommandName("c:\MyVoice.ogg")`
Fixed parameters	`CommandName("THIS_WINDOW")`

You'll learn all about 'em later. Notice how parameters are divided from the rest of the command - using quotes. If there are more parameters, they're divided with commas:

```
CommandName("param1","param2")
```

Here comes another important fact about commands and code lines:

You enter MMB script commands in rows, one after another, and you can enter as many lines as you like. As long as code lines follow script syntax, MMB won't mind having large number of lines.

And one about code line writing:

Writing of commands in MMB is not case-sensitive. Meaning, you can either write:

```
CommandName("param1","param2")
```

or:

```
COMMANDName("param1","param2")
```

MMB will eat it and automatically adjust letter cases. This works for command names only (parameters/variables are case-sensitive).

12.5 Script Comments

Some people write (and draw) comments in books, some use post-it notes... and you'll save paper and write comments in MMB ! Either to separate parts of code lines, describe code sections or even make reminders - in Script Editor you're able to write whatever you want under only one condition; you must put single or double (recommended) asterisk at beginning of the line where you'll write comments. It looks like this:

```
** Hey ! Now I can write whatever I want !
```

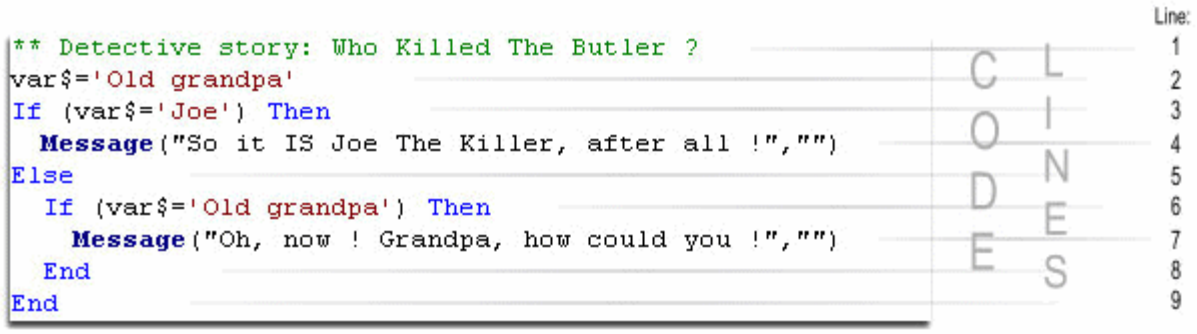
From **4.9.7** you can use also a comment block! With this sequence of characters /* code */ you can comment multiple lines of code at once.

```
/* first line of code
second line of code
third line of code */
```

Note that the **/*** sequence must be used as the very first characters of the first line of code you would like to comment (not counting the spaces) and the ending sequence ***/** as the very last characters of the last line of code you would like to comment. This block comment cannot be used inside the same line of code!

There's a case when comments will be added automatically: if MMB notices mistakes in code lines you wrote, it won't keep bugging you about errors, but simply "isolate" those lines using asterisks, making them - comment lines. Once you correct errors, simply remove comment asterisks and MMB will use these lines again.

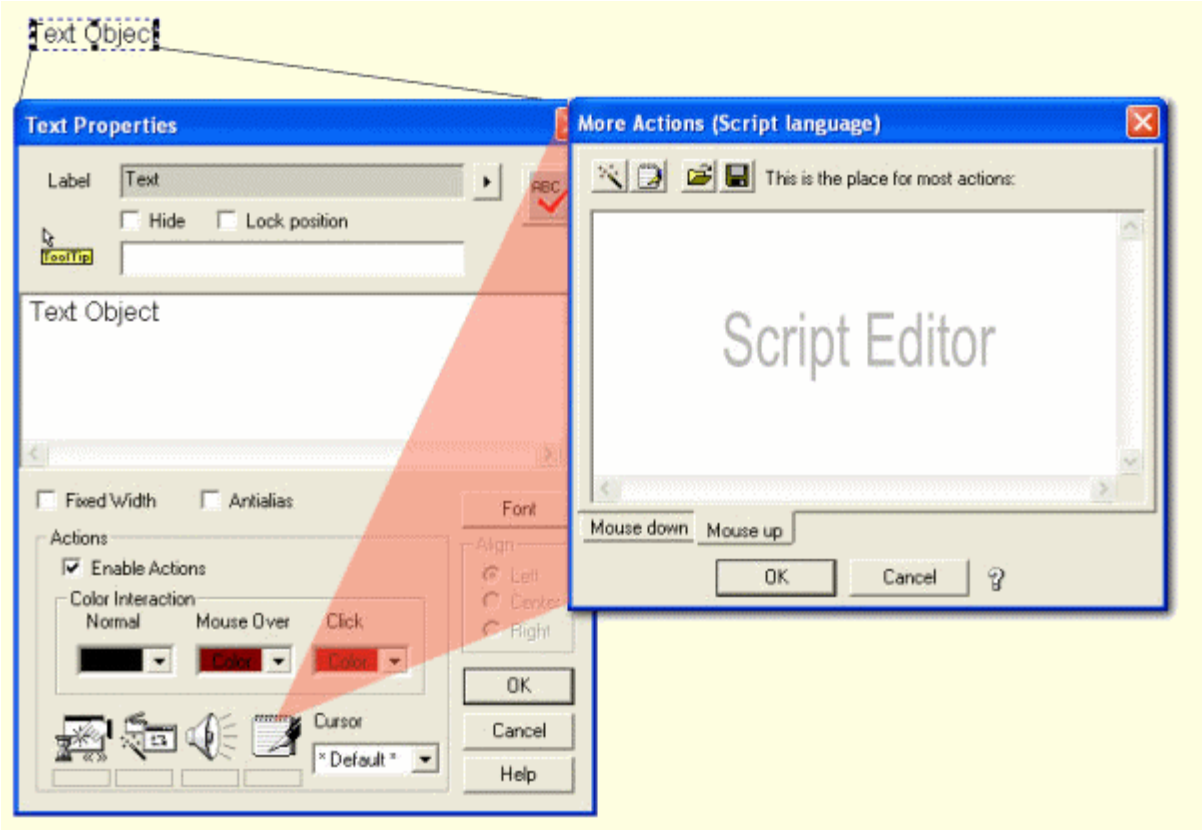
In already mentioned code line preview, you'll notice comment in the very first line:



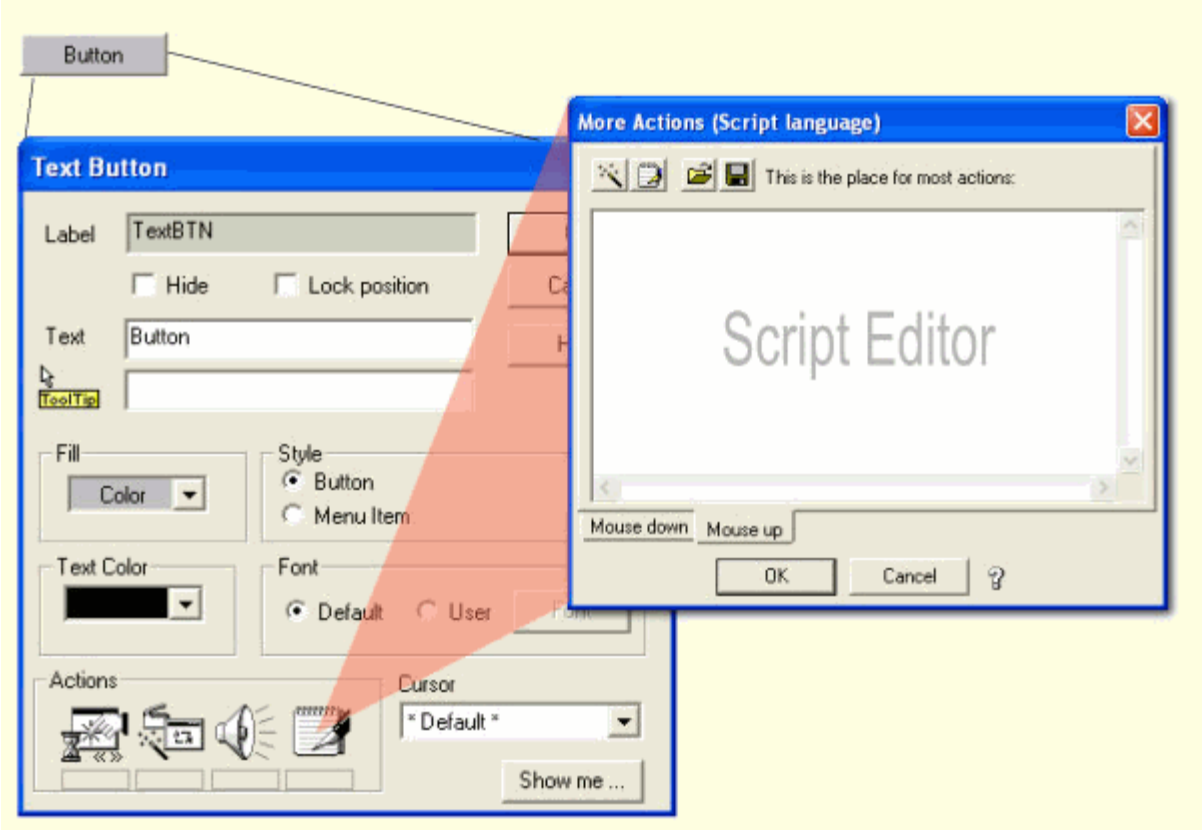
12.6 Script Editor

Before learning about actual command names & their usage, here's an introduction to MMB objects that contain script editors:

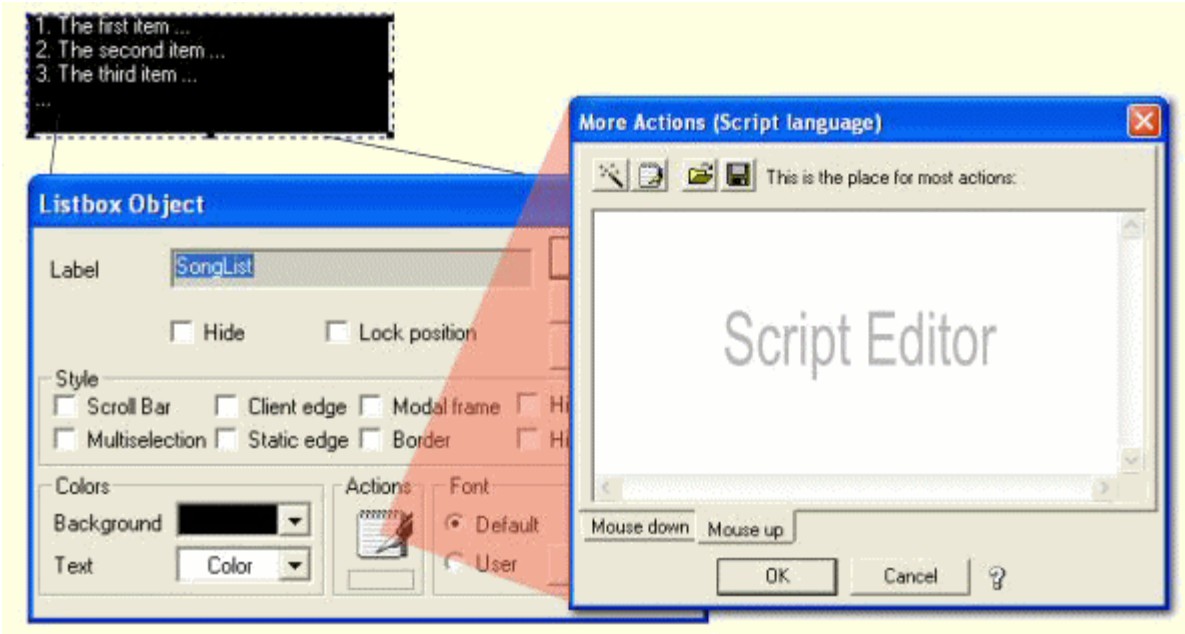
Text Object



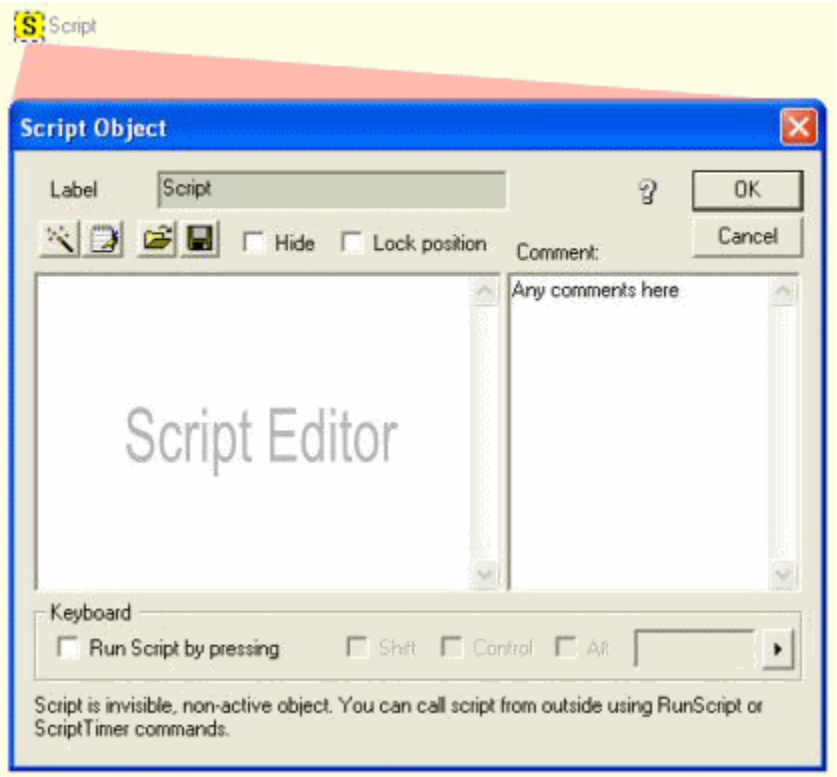
Button Object



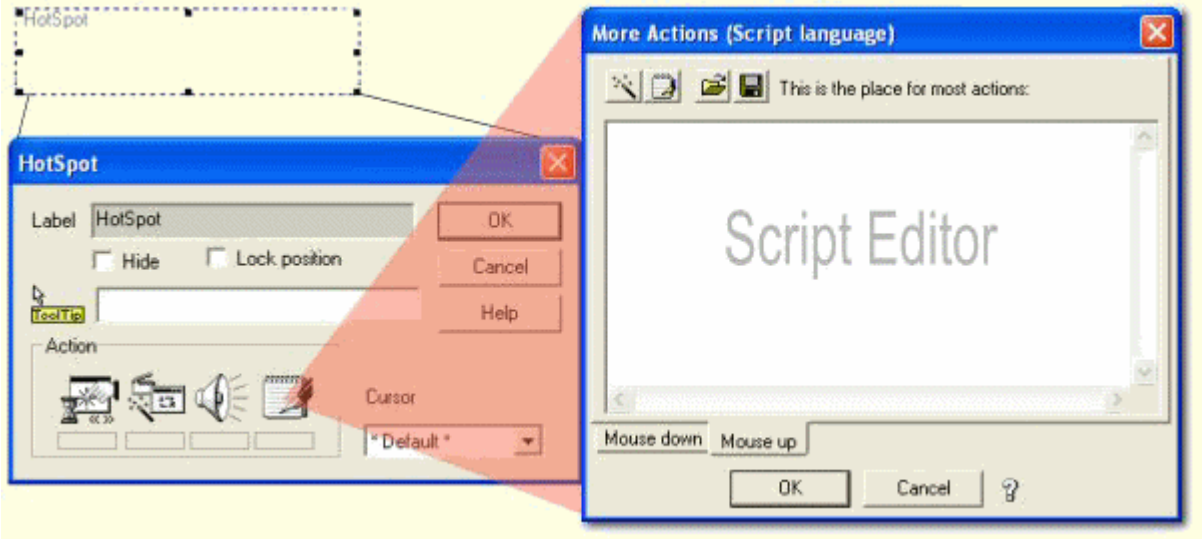
ListBox Object



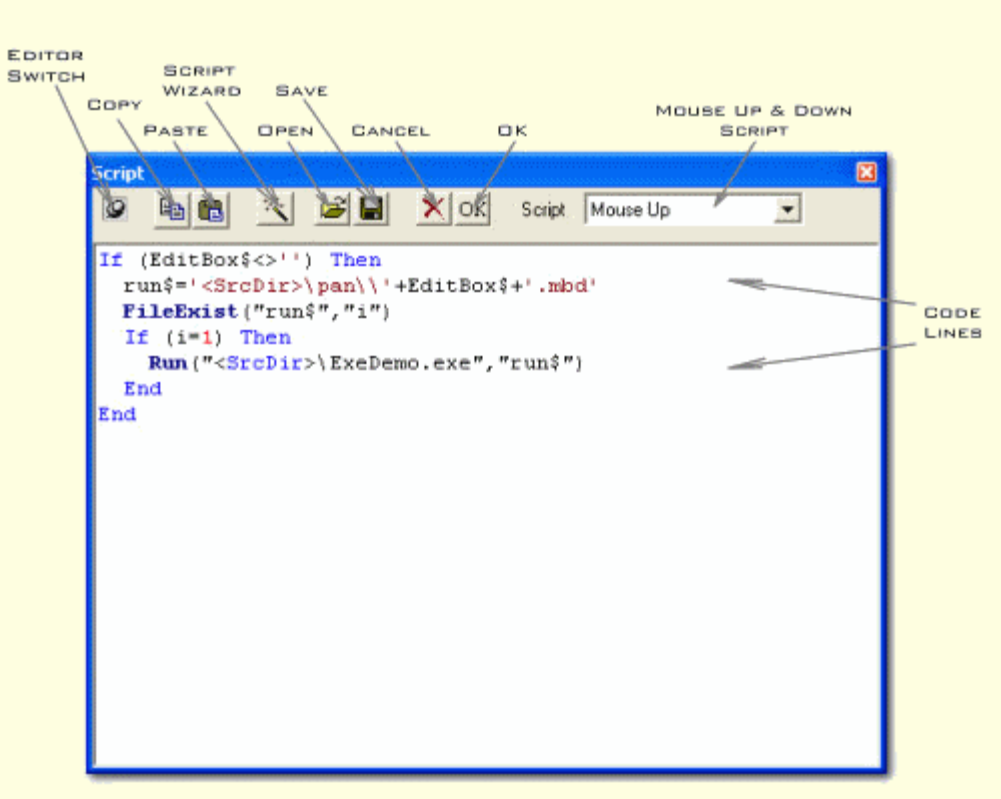
Script Object



HotSpot Object



And now, **mr. Editor** himself:

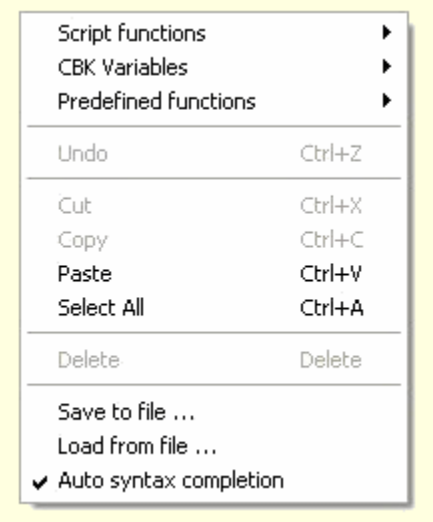


Here are explanations of Script Editor options:

Option	Explanation
Editor Switch	Switches view between Small Editor and Advanced Editor (recommended while it uses syntax highlighting and script wizard).
Copy/Paste	Clipboard commands for script copy/pasting
Script Wizard	Opens list of available MMB script commands with explanations, giving ability to specify parameters. When selection is confirmed, script line will be inserted into editor.
Open/Save	Enables opening & saving of MMB scripts using TXT and RTF files.

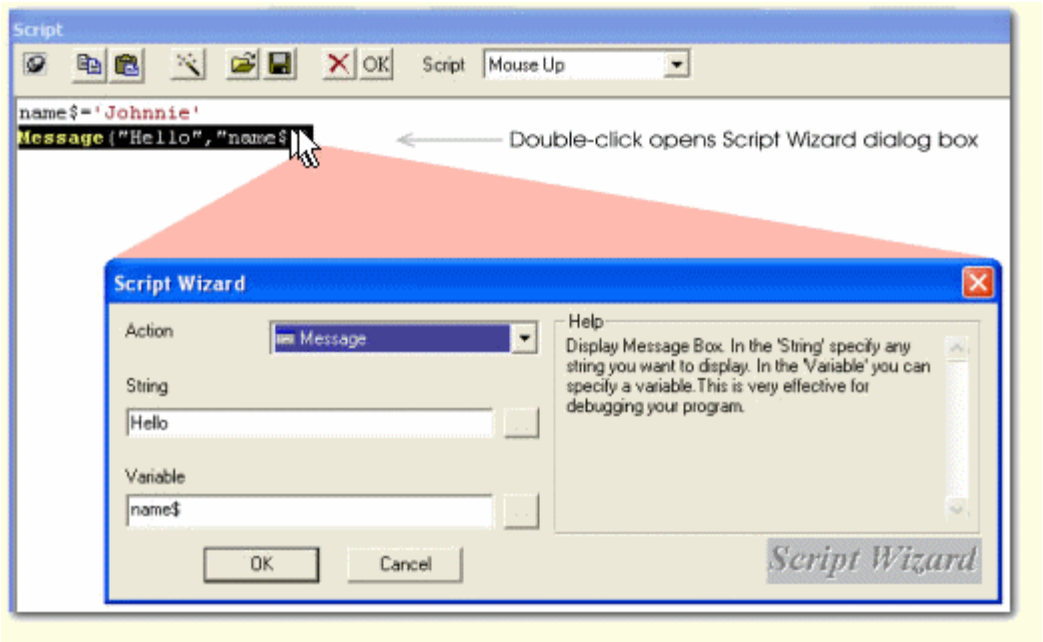
Mouse Up & Down Script	This switch opens separate script windows for actions that happen either on • Mouse Up - when user releases mouse button • Mouse Down - when user presses mouse button By default, you enter code in Mouse Up script.

Don't forget to try: right-click menu in the script editor !



As you can see - it's an advanced menu, not containing just cut/copy/paste commands. Here you can quickly reach script commands, save & load scripts.

Another handy feature of Script Editor is ability to edit written commands through Script Wizard dialog box by double-clicking on commands in Editor:



Right ! Now you know where scripts must be written and where you'll spend numerous nights making project of your life - let's take care of the script content now !

12.7 Constants

12.7.1 Introduction

There are plenty of things we can change. And then, there are many we can't.
Let's talk about the second ones.

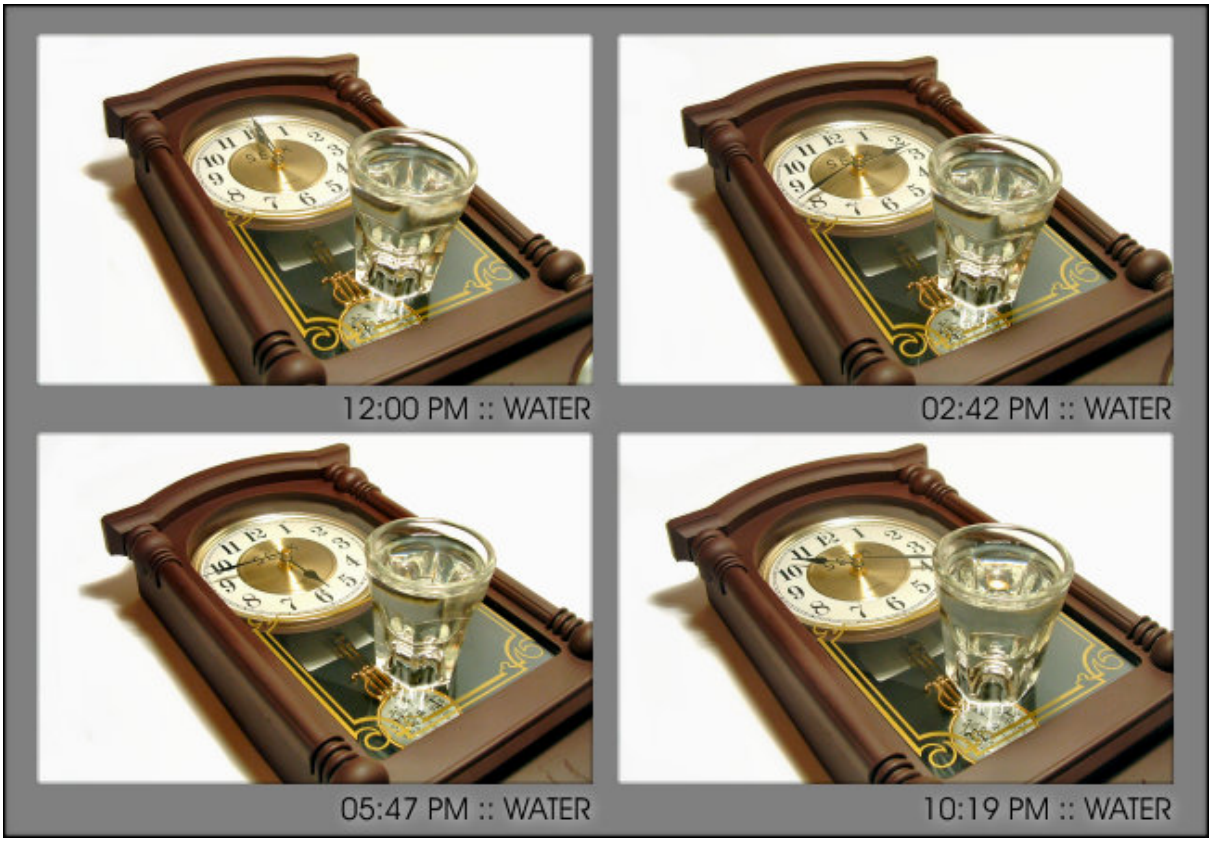
Earth spins in ~24 hours - we can't change it.
If you throw a rock, it'll fall to ground - there's no way around that.
Engines need power to run - at least for now, we can't make 'em run without it.

And computers ? They use constants all the time, although it looks like you can never catch their tail. Not only these silly machines can't work without constant power input (and constant voltage), but software part also enjoys using constant data values.

And constant data values are not only letters and numbers - they are represented with certain labels, used as **value descriptors**. Like postcards have addresses of their recepients...

...constants in software are also labeled, making it easy to determine what kind of data is assigned to which constants.

Data put into constants is not meant to be directly changed by your program.



Let's say the glass on images above represents constant **label** (container).
Water in glass is constant. It's not changing in time.

Constants in MMB are used:

- for various **multimedia & system info**, like screen size, audio & video duration, windows & project folder paths, media titles, etc.
- as **command parameters** when you want to say something special to MMB (like: please open this in new window, please put this on top, please make some more coffee)

And guess what - some constants can be changed, but indirectly - if you change played audio or video file, it's expected to affect duration & title constants. But MMB performs changes to constants automatically - so you don't have to worry about 'em at all.

Important feature of MMB's constants is ability to retrieve their values into string and numerical variables. This enables wide range of operations over retrieved info.

To copy constant info to numerical variable, you'll start by:

a) writing label of numerical variable:

```
CurrentYear
```

b) followed by equal sign:

=

c) and writing constant you want info from:

CBK_Year

All together, MMB code line for retrieving current year from CBK_Year constant to numerical variable CurrentYear looks like this:

CurrentYear=CBK_Year

To copy constant info to string variable, you'll start by:

a) writing label of string variable:

WindowsVer$

b) followed by equal sign:

=

c) and writing constant you want info from:

WinVer()

MMB code line for retrieving of Windows version from WinVer() constant to string variable WindowsVer$ looks like this:

WindowsVer$=WinVer()

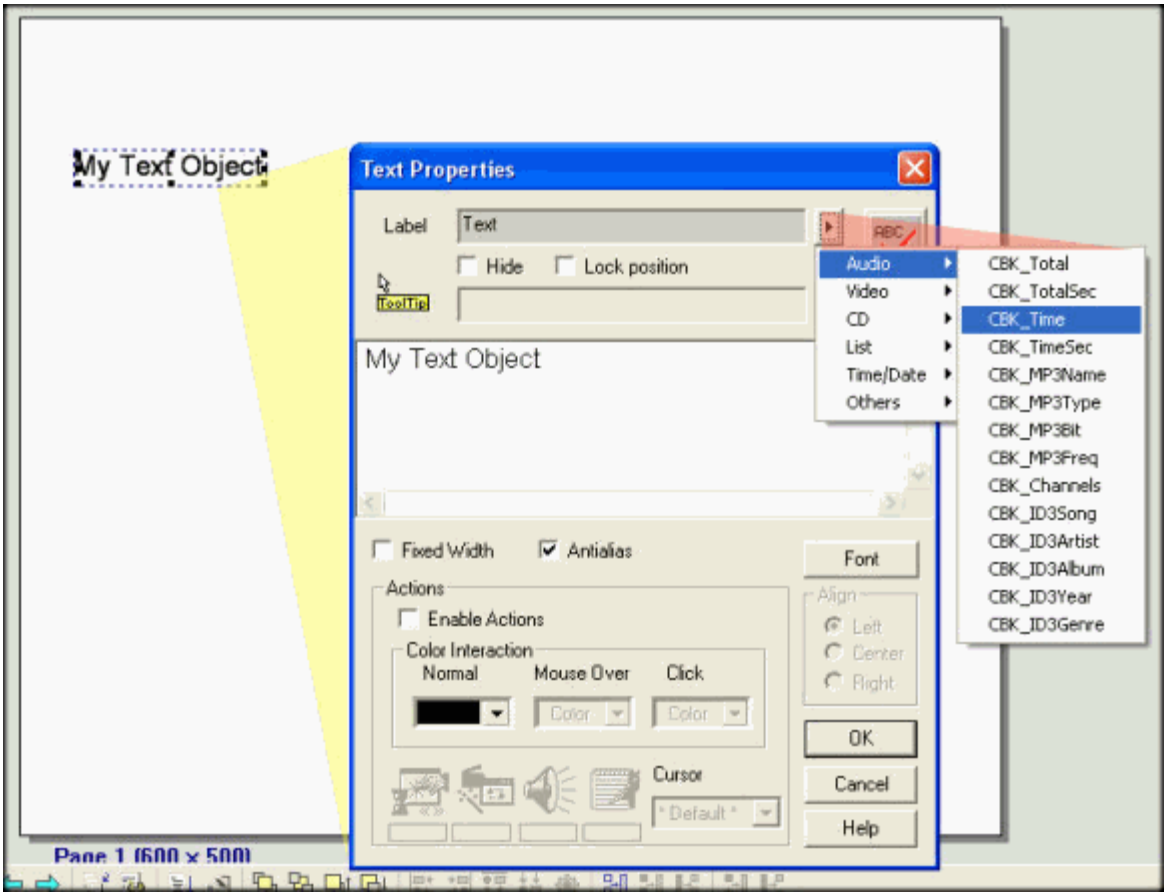
As visible from examples above, some constants return numerical values and some return string values (depends on info they contain).

12.7.2 CBK Constants

They sound so scary and ugly (CBK !), but they are nothing more than just a Multimedia & System info constants that can be displayed in text objects, like all multimedia players do !

Ha ! Who would guess it ;-)

CBK type of constant is a property of MMB's **Text Object**. So let's go there and see what can we find, change, learn:



Next to the "Label" property of text object, there's an arrow button. Click on it opens a menu with list of CBK constants that can be used in your project. They're separated into 6 categories, depending on their info coverage. After setting CBK constant, text object you assigned it to will in runtime mode (when application runs) display info from that constant.

Explanations of all CBK's coming up.

Audio CBK constants

CBK name	Explanation	Example
CBK_Total	Displays **total (duration) time** of currently loaded audio file. Format of displayed time is **mm:ss** (**m**inutes:**s**econds)	21:12
CBK_Total Sec	Displays **total (duration) time** of currently loaded audio file. Time is displayed in seconds.	360
CBK_Time	Displays **current time** (position) of loaded audio file. Format of displayed time is **mm:ss** (**m**inutes:**s**econds)	11:08
CBK_Time Sec	Displays **current time** (position) of loaded audio file. Time is displayed in seconds.	110
CBK_AudioType	Returns **type** of loaded Audio file.	MPEG 1 Layer 3
CBK_AudioBit	Returns bitrate of loaded audio file. Bitrate format: **kbit/s**	128kbps
CBK_AudioFreq	Returns **sampling frequency** of loaded audio file. Frequency is expressed in kHz (n x 1000Hz)	44.1kHz
CBK_AudioName	Returns **name** of loaded audio file, without path and extension.	Barry White - Baby Blues
CBK_Channels	Returns audio file channel mode. Can be either **Mono** or **Stereo**.	Stereo
CBK_ID3Song	Displays **title** of Audio file retrieved from ID3 tag.	Baby Blues
CBK_ID3Artist	Displays **artist** of Audio	Barry White

	file retrieved from ID3 tag.	
CBK_ID3Album	Displays **album** of Audio file retrieved from ID3 tag.	The Ultimate Collection
CBK_ID3Year	Displays **year** of Audio file retrieved from ID3 tag.	2000.
CBK_ID3Genre	Displays **genre** of Audio file retrieved from ID3 tag.	Soul
CBK_NumTracks	Returns **number of Audio CD tracks**.	12

Song List CBK constants

CBK name	Explanation	Example
CBK_CurItemList	Displays item currently selected in the song list.	Delirium - After All
CBK_NumInList	Counts and displays number of items in song list.	31
CBK_TotalList	Counts and displays **total duration** of **all items** in song list. Format of displayed time is **mm:ss** (**m**inutes:**s**econds). You will have to use **SongListTime()** function to compute **CBK_TotalList.**	135:28
CBK_TotalListSec	Counts and displays **total duration** of **all items** in song list. Time is displayed in **seconds**. You will have to use **SongListTime()** function to compute **CBK_TotalListSec.**	1130

Video CBK constants

CBK name	Explanation	Example
CBK_VName	Returns **name** of playing video file, without path and extension.	TheMatrix
CBK_VTotal	Displays **total (duration) time** of currently playing video file. Format of displayed time is **mm:ss** (**m**inutes:**s**econds)	180:00
CBK_VTotalSec	Displays **total (duration) time** of currently playing video file. Time is displayed in seconds.	10800
CBK_VTime	Displays **current time** (position) of playing video file. Format of displayed time is **mm:ss** (**m**inutes:**s**econds)	94:32
CBK_VTimeSec	Displays **current time** (position) of playing video file. Time is displayed in seconds.	48000
CBK_VTotalFrames	Returns number of frames of playing video file.	394024
CBK_VFrame	Displays **current frame** (position) of playing video file.	249253

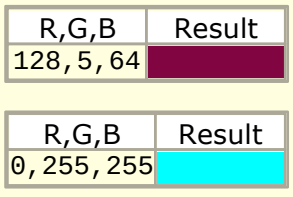
Date & Time CBK constants

CBK name	Explanation	Example
CBK_Year	Displays **current year**.	2004
CBK_Month	Displays **current year's month** using text format.	July
CBK_Month Num	Displays **current year's month** using integer (number) format in range **1 - 12**.	7
CBK_Day	Displays **day of current week**	Tuesday

	using text format.	
CBK_DayNum	Displays **day of current week** using integer (number) format. Range 1-7 represents days: • Sunday = 1 • Monday = 2 • Tuesday = 3 • Wednesday = 4 • Thursday = 5 • Friday = 6 • Saturday = 7	3
CBK_DateNum	Displays **current month's day** using integer (number) format in range **1 - 31**.	22
CBK_DateShort	Displays **current date** using short Windows format **DD/MM/YY** (**d**ay:**m**onth: **y**ear).	21/03/2002
CBK_DateLong	Displays **current date** using long Windows format **Month Day, YYYY**	July 23, 2004
CBK_TimeHMS	Displays **current system time** using **H:M:S** (**h**ours:**m**inutes: **s**econds) format and AM/PM time division.	01:06:12 PM
CBK_Time24	Displays **current system time** using **HH:MM:SS** (**h**ours:**m**inutes: **s**econds), 24-hour format.	13:06:12
CBK_Hour	Displays **current time hour** in range **0 - 23**.	13
CBK_Minute	Displays **current time minute** in range **0 - 59**.	06
CBK_Second	Displays **current time second** in range **0 - 59**.	12

General CBK constants

CBK name	Explanation	Example

CBK_AppFileName	Returns the filename of actual project in format "name"."exe"	test.exe
CBK_PageName	Displays **label of** currently opened **project page**.	Intro Page
CBK_Error	Returns (eventual) **errors sent by FMOD audio engine or Video Object**. FMOD errors are returned using integer values. **See list here**.	180:00
CBK_Volume	Returns **current master volume** in percentage (**0-100**).	60
CBK_URLpath	Displays **current URL** from MMB's **HTML Object** (if it's available).	http://www.go.com
CBK_OpenFile	After using MMB's Open File dialogbox, this CBK displays selected file name without path.	TurnAround.mpg
CBK_OpenDir	After using MMB's Open File dialogbox, this CBK displays path of selected file.	c:\videos\
CBK_ReturnVal	Run command sends return values from the running application (if return codes were implemented in application) to this CBK object.	1
CBK_SelColor	Returns RGB value from MMB's Color Picker dialogbox. Color picking uses RGB (Red, Green, Blue) system containing 3 components. Each component can have one of 256 levels. Greater value of component increases color intensity. If you specify value 0, color component will not be used. Value 255 uses maximum color intensity. Result of Color Picker is a comma-separated array of string values, each representing one RGB component (first: red, second: green, third: blue) in value range 0-255.	

	To retrieve color value, simply assign contents of CBK_SelColor to a string variable: color$=CBK_SelColor	
CBK_Font	Returns the string array containing Font parameters selected in **FontPicker()** function.	Arial|REGULAR|10|238| STRIKEOUT|
CBK_MsgEx	Returns the value of pressed button in **MessageEx** dialog.	Success: Returns the ID of the button pressed. Failure: Returns -1 if the message box timed out. Button Pressed Return Value OK = 1 CANCEL = 2 ABORT = 3 RETRY = 4 IGNORE = 5 YES = 6 NO = 7 TRY AGAIN ** = 10 CONTINUE ** = 11 ** Only valid on Windows 2000/XP and above.

Script CBK constants

CBK name	Explanation	Example
CBK_AudioEOF	If you set label of script object either on current project page or Master Top Layer to match this CBK constant, that script will be launched when audio file (**ogg**) playback ends. MMB will first look for this script object on current page and then on Master Top Layer.	**View Example**

CBK_Menu	If you modify label of object group to start with CBK_Menu, that group will be hidden every time user clicks outside that object group. This helps you in making pop-up menus, disappearing when user clicks away. On user's click on the screen, all CBK_Menu labeled objects will be hidden except the menu under mouse cursor. For example check **masterpages.mbd** that comes in MMB package.	**View Example**
CBK_EXIT	Use this to catch the **Close button** on the title bar or the **Escape key**. **Note:** The object or group MUST be in the **Master Top layer**. If you set label of object either on Master Top Layer to match this CBK constant, pressing of Escape key on keyboard will not (by default) close your application. Instead, script under CBK_Exit labeled object will be run, enabling you to create "Do you want to exit ?" message boxes or whatever you want to do when user wants to exit application.	• On Master Top Layer create rectangle. • Insert the text: «Do you want to exit?», and two buttons: «OK» and «Cancel». • Now group all 4 objects. • Rename The Group to CBK_EXIT. • Expand the group in the object list so you can access the items in it. • On Button OK: Trigger the Exit action. • On button Cancel: Hide the CBK_EXIT group. • Now hide the whole CBK_EXIT group. **Tip:** From the Wizard tool (on tool bar) select **Capture Exit** and **ESC** key, and then select the type from submenu - this will insert the object for you. **View Example**
CBK_CMDLINE	If you set label of Script object either on Master Page/ Master Top Layer to match this	**View Example**

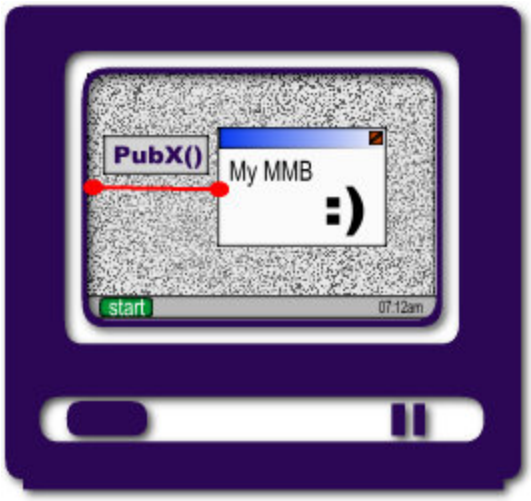
	CBK constant, each time you start your MMB application with some **command line parameters**, this Script object will be invoked and you will be able to process the passed command line parameters via this script object (e.g. start playback of Audio file passed by command line parameter or process each of the passed parameters individually).	

12.7.3 Publication Constants

These constants are related to application window of your project. You can retrieve various window sizes, position and check if project window is minimized.

PubX()

returns X (horizontal) position of project's window in pixels as an integer value

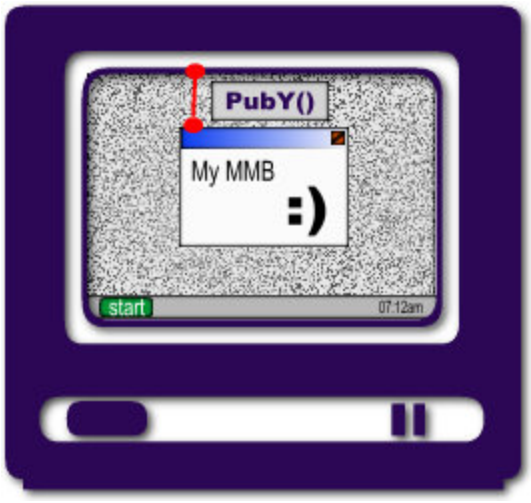


Script example:

```
Message("Project window X position: ","PubX()")
```

PubY()

returns Y (vertical) position of project's window in pixels as an integer value



Script example:

```
Message("Project window Y position: ","PubY()")
```

PubWidth()

returns width of project's window in pixels (with border) as an integer value

Script example:

```
Message("Project window width + border is: ","PubWidth()")
```

PubHeight()

returns height of project's window in pixels (with border & title) as an integer value

Script example:

```
Message("Project window height + border and title is: ","PubHeight()")
```

ClientWidth()

returns width of project's window (workarea) in pixels (without border) as an integer value

Script example:

```
Message("Workarea width is: ","ClientWidth()")
```

ClientHeight()

returns height of project's window (workarea) in pixels (with border & title) as an integer value

Script example:

```
Message("Workarea height: ","ClientHeight()")
```

IsMinimized()

this constant will check if application window of your project is minimized.

Integer value is returned, having one of two available states:

1 = project window is minimized
0 = project window is not minimized

Script example:

```
Message("Project window status: ","IsMinimized()")
```

IsMaximized()

this constant will check if application window of your project is maximized.

Integer value is returned, having one of two available states:

1 = project window is maximized
0 = project window is not maximized

Script example:

```
Message("Project window status: ","IsMaximized()")
```

12.7.4 Object Constants

ObjectX(ObjectLabel)

returns X (horizontal) position of specified object

Position retrieving starts from upper-left corner of project's window.
Returned position represents object's upper-left corner.

Specifying of object you want position for is done through label inside parenthesis.
For example: ObjectX(Button)

Script example:

```
Message("Current object X position: ","ObjectX(Button)")
```

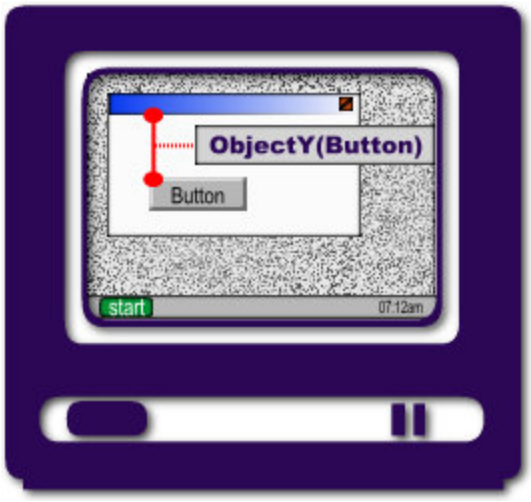
Script line above will retrieve upper-left corner X position of object labeled "Button"

ObjectY(ObjectLabel)

returns Y (vertical) position of specified object

Position retrieving starts from upper-left corner of project's window.
Returned position represents object's upper-left corner.

Specifying of object you want position for is done through label inside parenthesis.
For example: ObjectY(Button)



Script example:

```
Message("Current object Y position: ","ObjectY(Button)")
```

Script line above will retrieve upper-left corner Y position of object labeled "Button"

ObjectWidth(ObjectLabel)

returns width of specified object in pixels

Width retrieving starts from object's upper-left corner

Specifying of object you want width for is done through label inside parenthesis.
For example: ObjectWidth(Button)

Script example:

```
Message("Button width is: ","ObjectWidth(Button)")
```

Script line above will retrieve width of object labeled "Button"

ObjectHeight(ObjectLabel)

returns height of specified object in pixels

Height retrieving starts from object's upper-left corner

Specifying of object you want height for is done through label inside parenthesis. For example: ObjectHeight(Button)

Script example:

```
Message("Button height is: ","ObjectHeight(Button)")
```

Script line above will retrieve height of object labeled "Button"

CurrentObject()

returns name of current object from which is this function called.

Script example:

```
obj$=CurrentObject()
Message("Current object name: ","obj$")
```

For example, if this function is placed in object called "TextBTN", it will return "TextBTN" string. It's good mainly in case you want copy/paste an object multiple

times, especially if there is reference to a current object inside the script.

IsVisible(ObjectLabel)

this constant will check if specified object exists on current page and is it visible or hidden

Integer value is returned, having one of three available states:

-1 = object doesn't exist on current page
0 = object is hidden
1 = object is visible

Specifying of object you want status for is done through label inside parenthesis.

Script example:

```
Message("Object status : ","IsVisible(Button)")
```

Example above will check if object labeled "Button" exists and is it visible or hidden. If object doesn't exist, message box will display "-1".

ImageScrollX(ObjectLabel)

returns X position of left/top point of the image inside the Image object.
This is useful in case if image inside the Image object is bigger/smaller than Image object itself and it's scrollable inside the Image object ("Keep Actual Image Size" option must be enabled).

ImageScrollX returns negative values if left/top corner of the image object is out of the visible area of the Image object.

Script example:

```
Message("X position of Bitmap is: ","ImageScrollX(Bitmap)")
```

Script line above will retrieve X position of Left/Top corner of Bitmap.

ImageScrollY(ObjectLabel)

returns Y position of left/top point of the image inside the Image object.
This is useful in case if image inside the Image object is bigger/smaller than Image object itself and it's scrollable inside the Image object ("Keep Actual Image Size" option must be enabled).

ImageScrollY returns negative values if left/top corner of the image object is out of the visible area of the Image object.

Script example:

```
Message("Y position of Bitmap is: ","ImageScrollY(Bitmap)")
```

Script line above will retrieve Y position of Left/Top corner of Bitmap.

ImageWidth(ObjectLabel)

returns full width of image in pixels.

While ObjectWidth returns width of the object, ImageWidth returns width of entire Image, even if some parts of Image are invisible (because image is bigger than Image object).

Specifying of image you want width for is done through label inside parenthesis.
For example: ImageWidth(Bitmap)

Script example:

```
Message("Image width is: ","ImageWidth(Bitmap)")
```

Script line above will retrieve width of image labeled "Bitmap"

ImageHeight(ObjectLabel)

returns full height of image in pixels.

While ObjectHeight returns height of the object, ImageHeight returns height of entire Image, even if some parts of Image are invisible (because image is bigger than Image object).

Specifying of image you want height for is done through label inside parenthesis. For example: ImageHeight(Bitmap)

Script example:

```
Message("Image height is: ","ImageHeight(Bitmap)")
```

Script line above will retrieve height of image labeled "Bitmap"

GetImageOpacity(ObjectLabel)

returns current opacity of a given Bitmap object.

The ImageOpacity can be set both in a design time (via **Bitmap properties** dialog) or runtime (via **ImageOpacity command**).

Script example:

```
obj=ImageOpacity(Bitmap)
Message("Current opacity:","obj")
```

ScrollBarSize(ObjectLabel)

returns width (if ScrollBar is vertical) or height (if ScrollBar is horizontal) of image ScrollBar in pixels.

It's useful in cases when ScrollBar in Image object is enabled and you want to know the size of Image object including the ScrollBars. ScollBar size can be different on various OS.

Script example:

```
Message("Image ScrollBar height is: ","ScrollBarSize(Bitmap)")
```

Script line above will retrieve size of ScrollBar from image object labeled "Bitmap"

GetVideoParam(ObjectLabel)

returns the current state (0/1 = OFF/ON) of a given video parameter:

fullscreen	FullScreen mode is enabled (1) or disabled (0)
loop	Loop is enabled (1) or disabled (0)
mute	Sound is enabled (1) or disabled (0)
time	Video is switched to Time mode (1). 0 = Video is in **frame** mode
frame	Video is switched to Frame mode (1) 0 = Video is in **time** mode

Script example:

```
GetVideoState=GetVideoParam(Video, fullscreen)
```
returns 0 or 1 according the current state of Video screen

```
GetVideoState=GetVideoParam(Video, mute)
```
returns 0 or 1 if the audio Mute is disabled/enabled

12.7.5 System Constants

System info plays very important role in adjustments of your project to work on various end-user machines; depending on retrieved info, your project will be able to select most suitable presentation for current machine. In MMB, system constants hold **screen, window, processor and memory info** and these constants **are used as parameters for script commands.**

For example:

```
ScriptCommand(SystemConstant(),Param2)
```

Script command will use system constant as it's first parameter. See descriptions of real script commands to find out which ones can accept this kind of parameters.

ScreenWidth()

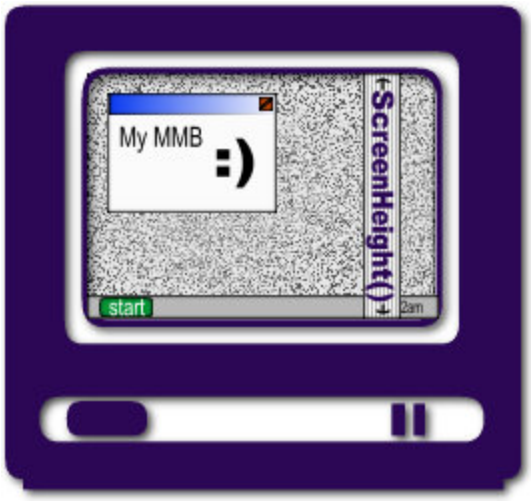
returns width of computer display in pixels as an integer value

Script example:

```
Message("Display width is: ","ScreenWidth()")
```

ScreenHeight()

returns height of computer display in pixels as an integer value



Script example:

```
Message("Display height is: ","ScreenHeight()")
```

WorkAreaWidth()

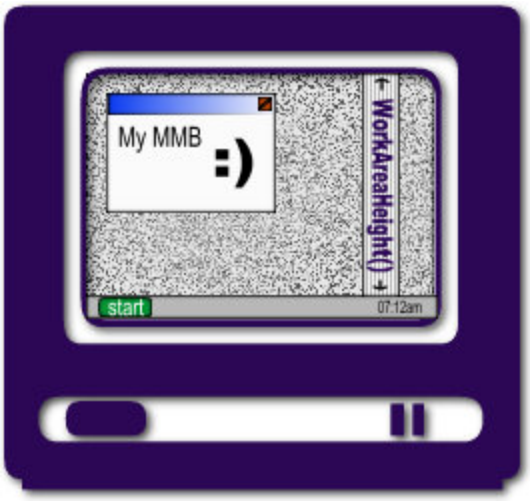
returns width of computer display work area in pixels as an integer value

Script example:

```
Message("Display work area width is: ","WorkAreaWidth()")
```

WorkAreaHeight()

returns height of computer display work area in pixels as an integer value

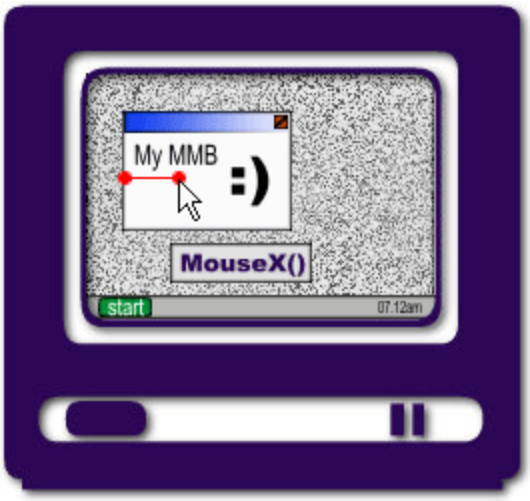


Script example:

```
Message("Display work area height is: ","WorkAreaHeight()")
```

MouseX()

returns X (horizontal) screen position of mouse in pixels as an integer value

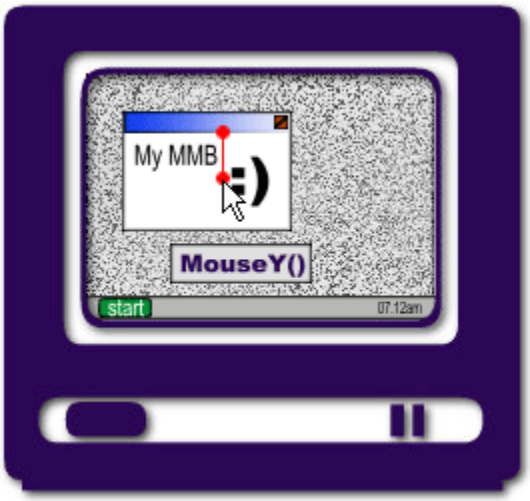


Script example:

```
Message("Current mouse X position: ","MouseX()")
```

MouseY()

returns Y (vertical) screen position of mouse in pixels as an integer value



Script example:

```
Message("Current mouse Y position: ","MouseY()")
```

MouseLButton()
MouseRButton()
MouseMButton()

returns a state of mouse buttons.
If mouse button is pressed then return value is **1** otherwise **0**.

Script example:
To Page Start code insert ScriptTimer, which will start a global MouseState detection script
ScriptTimer("Timer1=Script","50")

Create new script object called Script and insert the below code into it:

```
MStateL=MouseLButton()
MStateM=MouseMButton()
MStateR=MouseRButton()
MState=MStateL+MStateM+MStateR
** one button is clicked
If (MState=1) Then
  If (MStateR=1) Then
    LoadText("Text","RIGHT CLICK IS DETECTED")
  End
  If (MStateL=1) Then
```

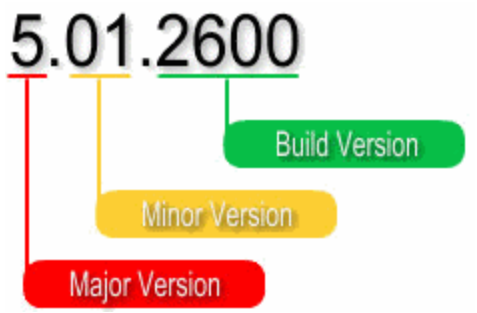
```
    LoadText("Text","LEFT CLICK IS DETECTED")
  End
  If (MStateM=1) Then
    LoadText("Text","MIDDLE CLICK IS DETECTED")
  End
  ScriptTimer("Timer1=click","50")
  Return()
Else
  ** none or more than one button is clicked
  LoadText("Text"," ")
End
** restart mouse detection script
ScriptTimer("Timer1=click","50")
```

For more advanced example of mouse state detection procedure check the **mousestatedetection.mbd** example project.

WinVer()

returns major, minor and build version of running Windows as string (text) value

These values are separated with dot **.** so use MMB's advanced string functions to extract individual version values.



Script example:

```
var$=WinVer()
Message("Windows version : ","var$")
```

Returned value will be for e.g:

```
5.01.2600
```

Windows have somewhat standardized version numbers. Here's a list:

Windows 95	4.00.950
Windows 95 SP1	4.00.(>950) / 4.00.(<=1080)

Windows 95 OSR2	4.(<10).(>1080)
Windows 98	4.10.1998
Windows 98 SP 1	4.10.(>1998) / 4.10.(<2183)
Windows 98 SE	4.10.(>=2183)
Windows Me	4.90.3000
Windows NT 3.51	3.51.1057
Windows NT 4.0	4.00.1381
Windows 2000	5.00.2195
Windows XP/SP1	5.01.2600
Windows Vista	6.00.6000

UsingWinNT()

returns integer (number) value having one of two available states:

1 = machine runs WinNT-family operating system
0 = machine doesn't run WinNT-family operating system

Script example:

```
Message("WinNT-compatible OS present: ","UsingWinNT()")
```

If WinNT family OS is present, returned value will be:

```
1
```

ScreenColors()

returns number of colors currently being used by Windows & graphics card

Value returned is integer (number) and represents number of colors (16,256) for lower modes or number of bits (16, 32) for higher graphic modes.

Script example:

```
Message("Current graphics color mode: ","ScreenColors()")
```

Returned value will be for e.g:

```
32
```

ProcType()

returns manufacturer, type and speed of central processor unit (CPU) as string (text) value

Script example:

```
var$=ProcType()
Message("CPU in this machine is: ","var$")
```

Returned value will be for e.g:

```
Intel (R) Pentium (R) 4 CPU 2.40 GHz
```

ProcFreq()

returns speed of central processor unit (CPU) in MHz as an integer (number) value

Script example:

```
Message("CPU frequency in MHz: ","ProcFreq()")
```

Returned value will be for e.g:

```
2405
```

GetMemory()

returns quantity of Total and Free RAM memory as string (text) values.

These values are separated using slash **/** character, so use MMB's advanced string functions to extract individual memory values.

Script example:

```
Message("Machine memory status: ","GetMemory()")
```

Returned value will be for e.g:

```
512/231
```

(first value represents **Total Memory**, second **Free Memory**)

12.7.6 Path Macros

In MMB, there are fixed and dynamic paths:

Fixed path: complete drive + folder + file path that is static, pointing to one location. For example: c:\My Wonderful Project\App\MyApp.exe

Fixed path downside is zero-adjustability of source folder by your end-user; he can't select where application will be installed, and that's not professional approach to end-users these days.

Dynamic path: in MMB path macros are being used to make easier locating of frequently used folders. While path macros read current path to your application, using them you assure end-user adjustability of installation folder.

Path macros in MMB are string (text) values and you will use them as a script command parameters. For example:

```
ScriptCommand("Param1","<ProjectFolder>\MyApp.exe")
```

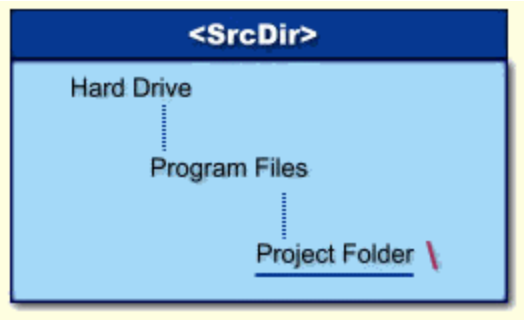
Example above uses path macro called <ProjectFolder>\ to retrieve folder where your application is located. After path macro you write full file name (in this case: MyApp. exe).

Besides those available in MMB, there are additional path macros available in MMB PlugIns; refer to their documentation for more info on this subject.

Now, let's see what path macros we can use directly in MMB:

<SrcDir>

The most often used path macro, represents folder where your (running) application is located.
While other project files are usually also located there (or in subfolders), Source Directory (SrcDir) macro becomes common part of file-oriented commands in your project.



If complete path to your application is:

c:\Program Files\My Project Folder\MyApp.exe

...then <SrcDir> version of this path would be:

<SrcDir>\MyApp.exe

It's much easier that way, isn't it ? For every file path inside your project's folder you will substitute fixed path with dynamic version and enjoy all benefits of self-adjusting paths !

Inside MMB script command, this path macro would look like this:

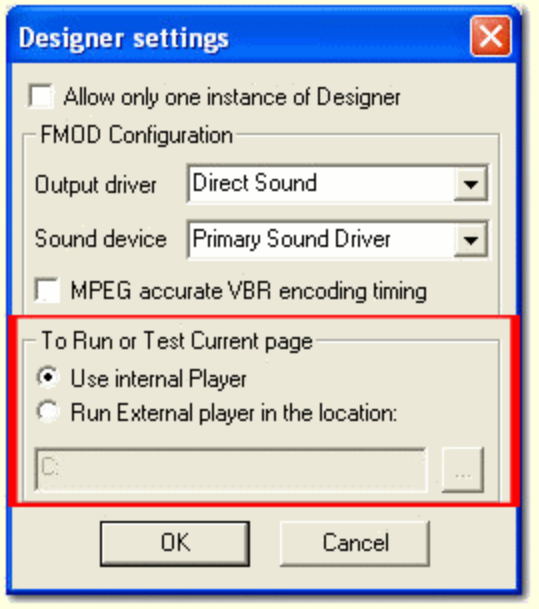
```
ScriptCommand("Param1","<SrcDir>\MyApp.exe")
```

Example above instructs script command to use <SrcDir>\ inside second parameter and locate application's source directory before looking for file (MyApp.exe).

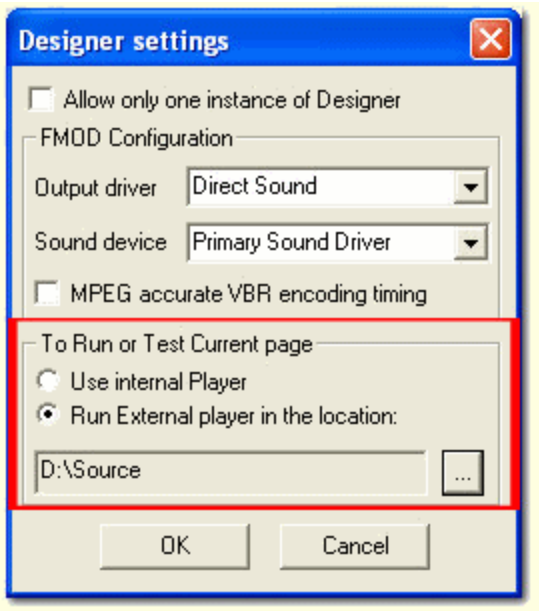
Once MMB translates this path macro, it's again full path (for e.g. c:\Program Files\My Project\MyApp.exe) and script command can be run.

Notice: MMB runs project files in two modes:

- **Design Mode** - while editing your project in MMB, using Project -> Run option will execute project internally, where <SrcDir> becomes MMB's installation folder (for example c:\Program Files\Multimedia Builder) and all files you refer to using <SrcDir> should be placed there.
  This mode is enabled when *Tools -> Designer Settings* is set to internal player:



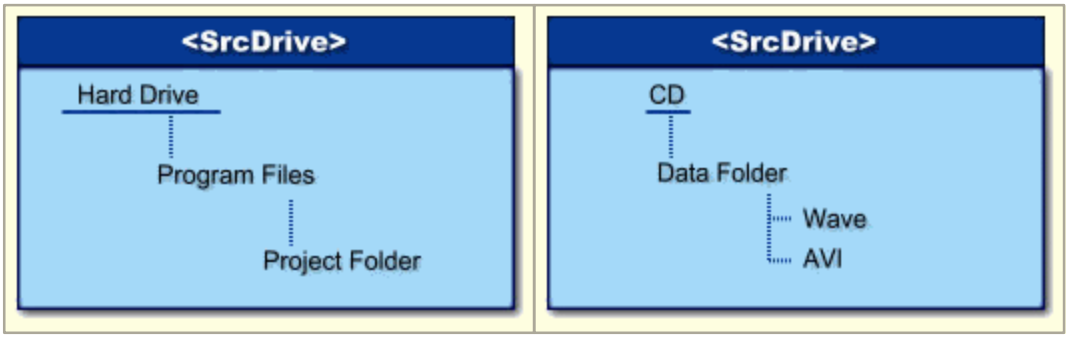
- **RunTime Mode** - executes project either after compiling (File -> Compile) as stand-alone EXE file, or when *Tools -> Designer Settings* is set to specified external player:



What is "external player" ? It's an instance of compiled (EXE) file placed in some folder (for e.g. D:\Source\Autorun.exe). Once you refer to it in *Tools -> Designer Settings* (for e.g. D:\Source), <SrcDir> for projects you run in MMB editor becomes source folder of external player and all files you refer to using <SrcDir> should be placed there.

<SrcDrive>

Represents root folder of (source) drive your application is running on.



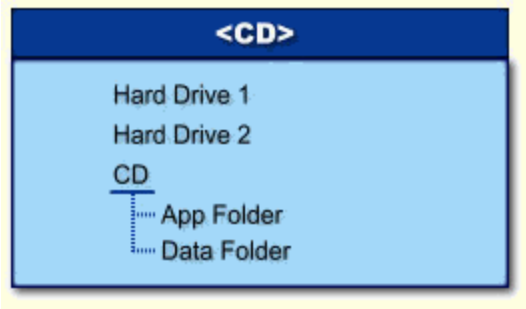
If your application is located on C drive, <SrcDrive> path macro will return:

c:\

This macro is suitable for ROM-oriented media projects (run from CD-ROMs, DVDs) that must reach data folders starting from the root folder of media.

<CD>

Returns root folder of the first available CD drive.



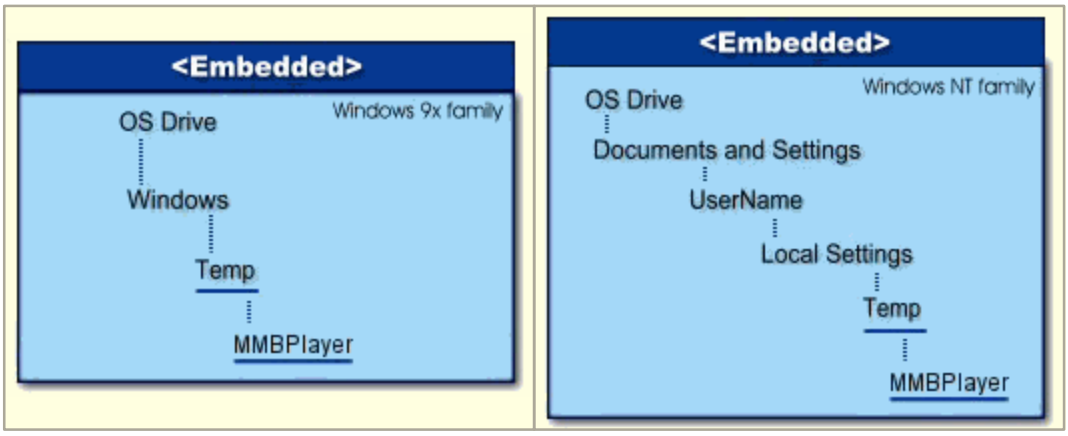
If first available CD drive is placed on E letter, <CD> path macro will return:

E:\

This macro is also suitable for ROM-oriented media projects (run from CD-ROMs, DVDs) that must reach data folders starting from the root folder of media. Also works well for multimedia players for locating default Audio CD / DVD drive. Downside of this macro is - it cannot enumerate all drives, but only the first one.

<Embedded>

Returns folder used by your application for placing of *embedded* files.



As you can see on images above, locations of embedded folders are Window's folders for temporary files and they differ on Windows OS families.

For example. If you have *embedded* sound file:

MySound.wav

It's location with path macro will be:

<Embedded>\MySound.wav

When you run application, MMB will translate that path (for itself) either to:

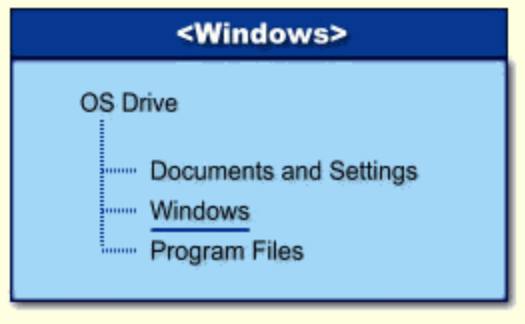
c:\Windows\Temp\MMBPlayer\MySound.wav (for Windows 9x OS family)

...or to something like:

c:\Documents and Settings\UserName\Local Settings\Temp\MMBPlayer\MySound.wav
(for Windows NT OS family, where folders for temporary files are individual for users)

<Windows>

Returns Windows root folder.



For example, if Windows is installed on the C drive and you want to find win.ini file, location with path macro will look like this:

<Windows>\win.ini

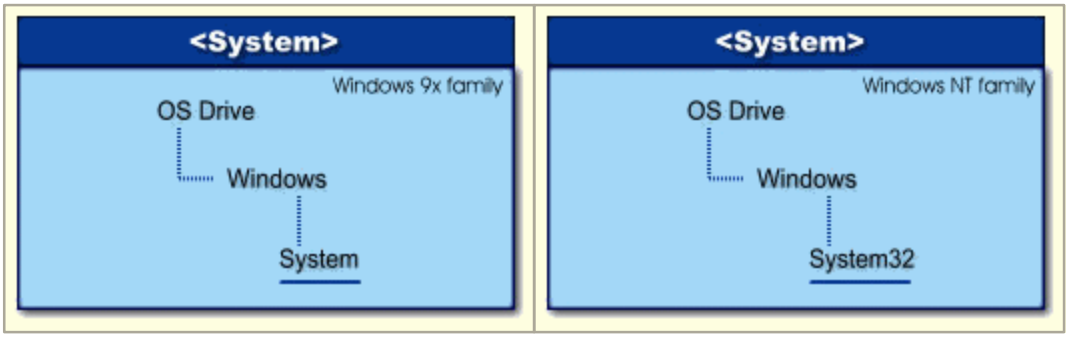
When you run application, MMB will translate that path (for itself) to:

C:\Windows\win.ini

In MMB, <Windows> path macro is usually called to locate system INI files.

<System>

Returns Windows System folder.



If you're trying to locate program:

dxdiag.exe

It's location with path macro will be:

<System>\dxdiag.exe

When you run application, MMB will translate that path (for itself) either to:

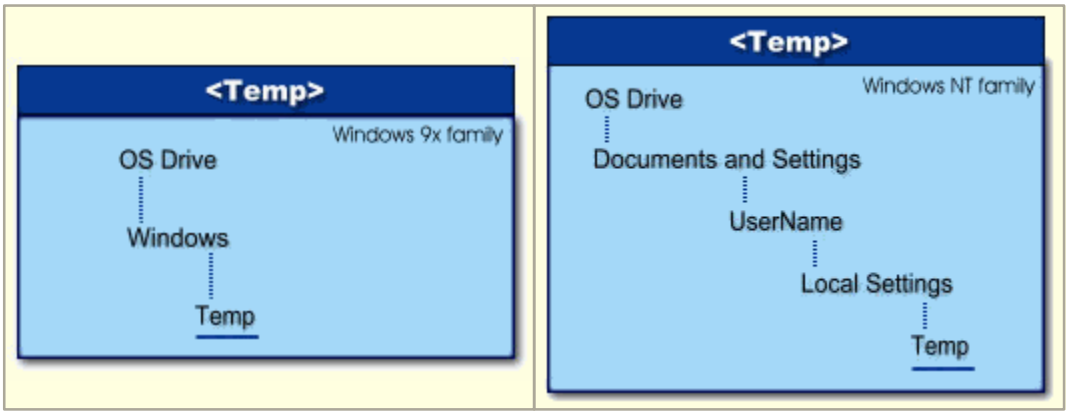
c:\Windows\System\dxdiag.exe (for Windows 9x OS family)

...or to something like:

c:\Windows\System32\dxdiag.exe (for Windows NT OS family)

<Temp>

Returns Windows Temp folder used for placing of Windows temporary files.



If you have some temporary file:

TempList.txt

It's location with path macro will be:

<Temp>\TempList.txt

When you run application, MMB will translate that path (for itself) either to:

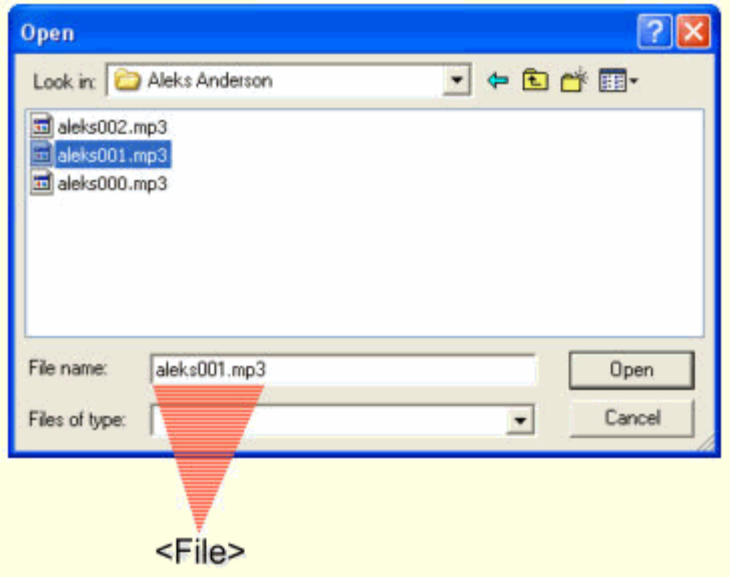
c:\Windows\Temp\TempList.txt (for Windows 9x OS family)

...or to something like:

c:\Documents and Settings\UserName\Local Settings\Temp\TempList.txt (for Windows NT OS family, where folders for temporary files are individual for users)

<File>

Returns full path & name of file opened using MMB's Open File dialogbox.



If you open file:

c:\windows\regedit.exe

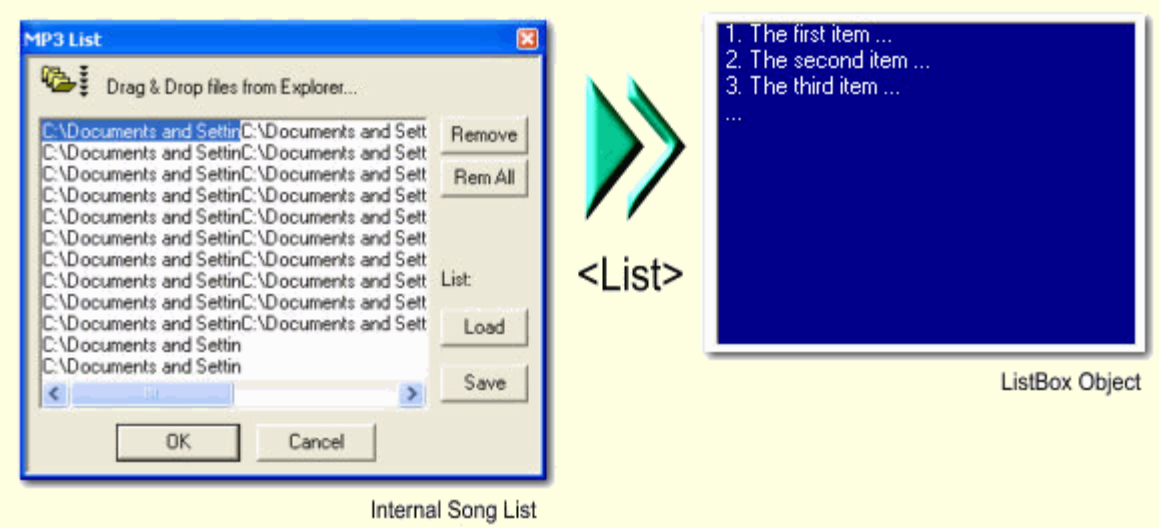
that path can be retrieved later by simply pointing to:

<File>

Content of this path macro is changed every time user selects & opens file, so it's recommended to use string variables or arrays for multiple path storage.

<List>

Enables operations specific to MMB's internal play list and ListBox object.



<List> constant holds items present in MMB's Internal Song List, so you can easily transfer all it's items to MMB's ListBox Object.

Script example:
```
ListBoxAddItem("SongList","<List>")
```

In real-code example above, content of <List> will be transfered to ListBox object labeled "SongList" . For more info on ListBoxAddItem command, refer to ListBox Commands section.

<This>

<This> is a special macro useful only in MCI command. It will tell the device the MMB window will be the parent.

Here is a small sample how to play MPG movie inside the mbd project in the position (100,50,100,100):

```
MCICommand ("open <SrcDir>\sample.mpg alias MPEG style child parent <This
>")
MCICommand ("put MPEG window at 100 50 200 200")
MCICommand ("window MPEG state hide")
MCICommand ("play MPEG")
```

12.8 Variables

12.8.1 Introduction

Changes, changes & more changes ! There are more of these every day. Everything started with 'em and they'll be around to the very end.

Changes = Variables

When you go for shopping and spend all your money there - variable called "Wallet" is changed from state "Full" to "Empty".

Travelling from one country to another makes change to your variable location.

Eating ice-cream reduces it's quantity in the box.

Computers couldn't operate without streams of flip-flopping zeroes and ones.

It's hard to imagine how dull everything would be without variables. Especially computers. That's why computer software uses 'em a lot. Being multimedia authoring tool (intended for entertainment stuff as opposite to The Dull World) - variables could prove to be quite important for MMB, couldn't they ? Your application depends on variable data.

Data put into variables is meant to be directly changed by your program.

The glass on images above represents constant **label** (container).

Content in glass is variable. It's changing as time passes.

Variables in MMB are used for every segment of data your program can get or put.

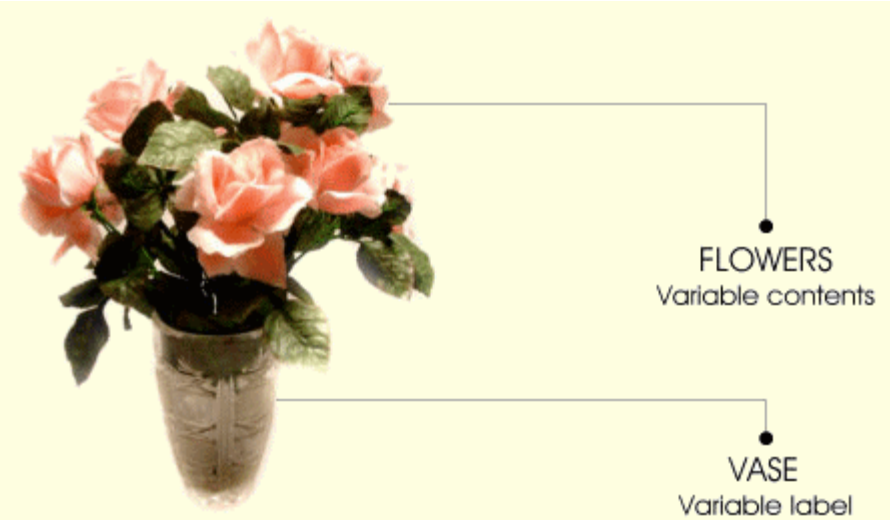
There are two sorts of variables MMB uses:

- **Integer (number) variables** - used for storing numbers, these variables are important for MMB's script math operations
- **String (text) variables** - store all kinds of characters, words, sentences, paragraphs...

Variables consist of:

- **variable labels**: names you'll refer to when using 'em. Variables are represented with certain labels, used as value descriptors and are composed of alphanumerical characters (for e.g. *vase*, *date, name, address, email1, email2*...)

- **variable contents**: numbers for numerical variables, letters for string variables



Labels and content of variables are specific to variable type and are discussed below.

Notice for MMB users with experience in programming: MMB doesn't require "declaring" of variables prior to their use. It takes care of 'em automatically, so your only job is to

use them.

12.8.2 Numerical Variables

Some tasks in scripting require math operations. To perform these, you need - numbers. That's where numerical variables take place. You'll use 'em as containers for numbers.

There are two kinds of numbers you'll store in numerical variables:

- real (floating point) numbers, like: 43.92, 1.2938, 441.1, 881824.2
- integer (round) numbers, for example: 43, 21, 1, 4932, 6884

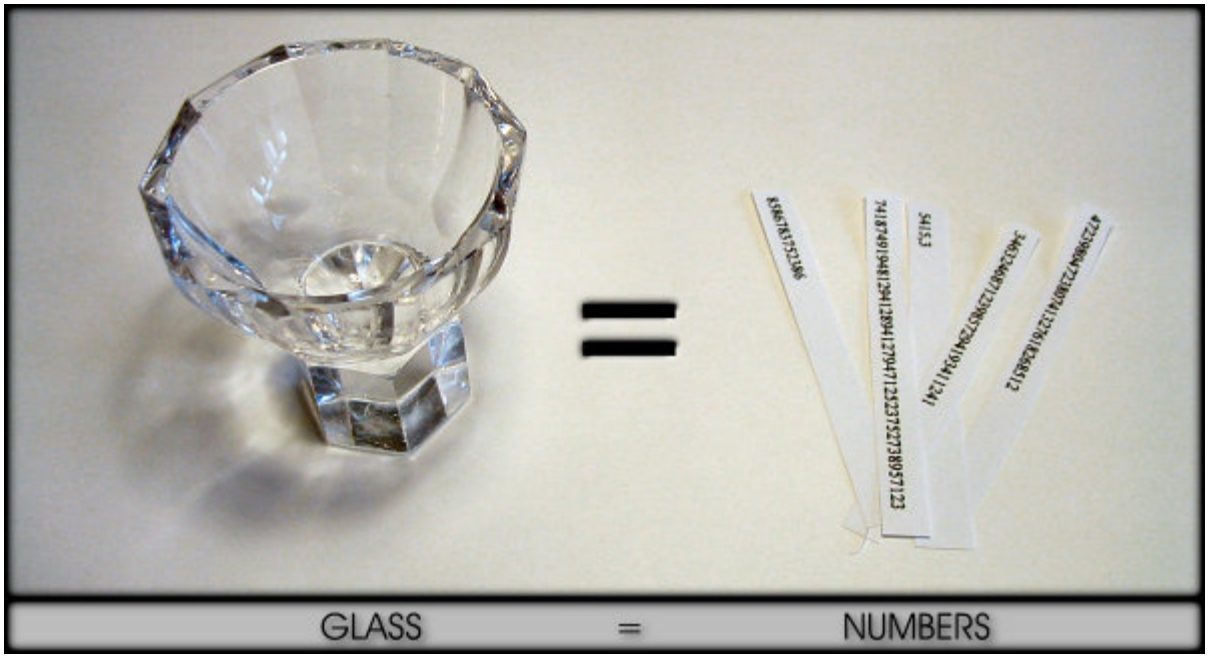
If we take this crystal glass and put numbers in it - we get **numerical variable** !

Just like that glass on image above, important part of numerical variable in MMB is **label**.
Here are some examples of valid labels:

i	n	month	1st_one
click_count	NoOfWords	5thUser	glass

It is recommended to use meaningful & descriptive labels, not too short, not too long - just enough to describe it's contents. If you'll use numerical variable to store number of mouse clicks, label of that variable should be MouseClicks or NoOfMouseClicks.

The most important part of this story is - **assigning of variable contents**.



In MMB script code lines, it is being done by:

a) writing label name:

glass

b) followed by equal sign:

=

c) and writing content (value) of variable at the end, behind label and equal sign:

5415

All together, MMB code line for assigning of number to numerical variable looks like this:

glass=5415

There ! With this line you instructed MMB: "Take a **glass** and **put** number **5415** in it."

Where will you give that instruction ? In MMB's Script Editor, of course. That's where you'll write your very first MMB code line (if you didn't cheat and skipped *all this boring stuff*) ;)

Very nice. Now, is this all you can do ? Assign fixed numbers and never change 'em again ?

Nope !

It is important to have ability of **assigning value of one variable to the value of another variable**. In other words: if you draw one caricature and give it to your kid:

	*...what will you do if other kid wants one too ?* **You will copy it** - or draw it again !	
face1	**=**	**face2**

In MMB script code lines, it is being done by:

a) writing label of **destination** numerical variable (one that **accepts** value):

```
face2
```

b) followed by equal sign:

```
=
```

c) and writing label of **source** numerical variable (one that **gives** value):

```
face1
```

All together, MMB code line for assigning of content of one numerical variable to another numerical variable looks like this::

```
face2=face1
```

If variable **face1** contains number 328, code line above will assign the same value to **face2** .

Usage of variable-to-variable approach is recommended when you're planning some math operations on numerical variable and don't want to loose original value. In real life, you would practice on piece of paper before doing anything on original. When scripting in MMB, you don't have to worry about corrupting the original - you simply take another variable and perform changes on it ! Now that's a neat advantage, isn't it !

Talking about math, here are operations you can use in MMB script language:

Operator	Explanation	Examples
+	**Addition** of two or more numbers or numerical variables. Result of addition is assigned to	a=12+43 a=b+c Weekend=Saturday+Sunday

	destination numerical variable.	DaysOfWeek=1+2+3+4+5+6+7
-	**Subtraction** of two or more numbers or numerical variables. Result of subtraction is assigned to destination numerical variable.	a=43-12 a=c-b WorkDays=Week-Weekend Tax=100-11-65
*	**Multiplication** of two or more numbers or numerical variables. Result is assigned to destination numerical variable.	a=43*12 a=c*b Month=Week*4 Sales=24.99*buyers
/	**Division** of two or more numbers or numerical variables. Result of division is assigned to destination numerical variable.	a=43/12 a=c/b Month=Year/12 SpecWeight=quantity/30

Notice how many combinations are available here...

Math operations with two or more numbers:

```
a=432/21
SixPack=4+1+1
```

Math operations with two or more numerical variables:

```
a=b*c
Price=Quantity+ItemPrice- Taxes
```

Math operations with combinations of numbers and numerical variables:

```
a=b+42*NumberOfClicks
Price=2032*Quantity+83/ NumberOfMembers
```

All these are valid MMB script code lines ! You will find this combinations very useful in practice.

Once calculations on the right side are done, result is assigned to numerical variable on the left and you can use calculated value wherever you want, for example to show it as message:

```
Price=2032*Quantity+83/ NumberOfMembers
Message("Price of your order is: ","Price")
```

These two code lines will a) calculate order price, and b) display it using MMB's message box

Here's an example of multiple code lines with math operations:

```
days =365
```

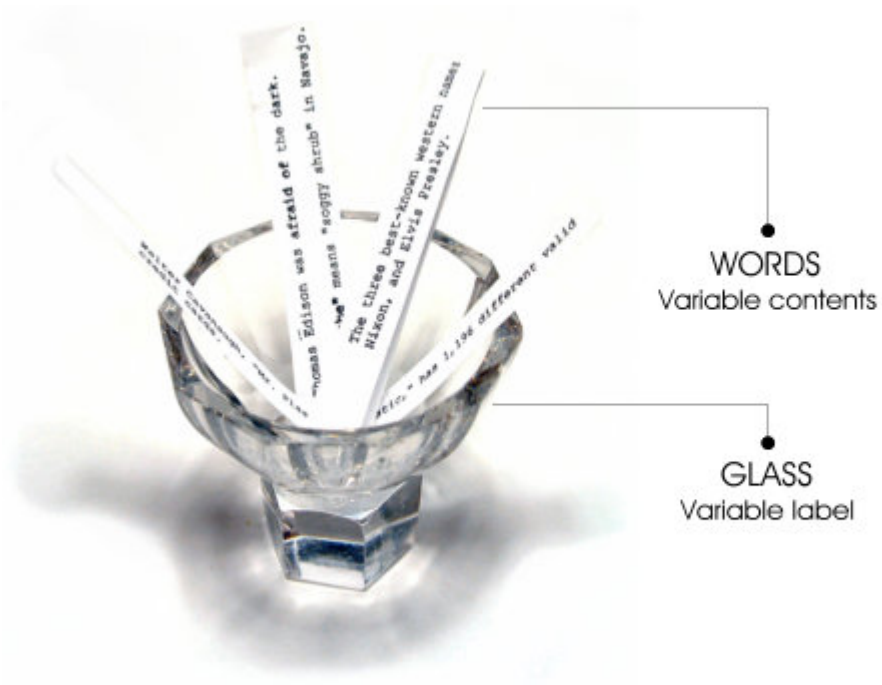
```
years=100
millenium=1000
NumberOfDays=days*years* millenium
Message("Number of days in one millenium: ","NumberOfDays")
```

The result is 36500000 and it's displayed in MMB's message box.

Examples mentioned above already show you what numerical variables could be used for. And there are hundreds of other uses, especially when dealing with multimedia subject.

12.8.3 String Variables

For everything else except numbers used for math operations - you will use string variables.
Letters, words, sentences, paragraphs - are natural contents of string variables !



If we take this crystal glass and put words in it - we get **string variable.**

Just like that glass on image above, important part of string variable in MMB is **label**.

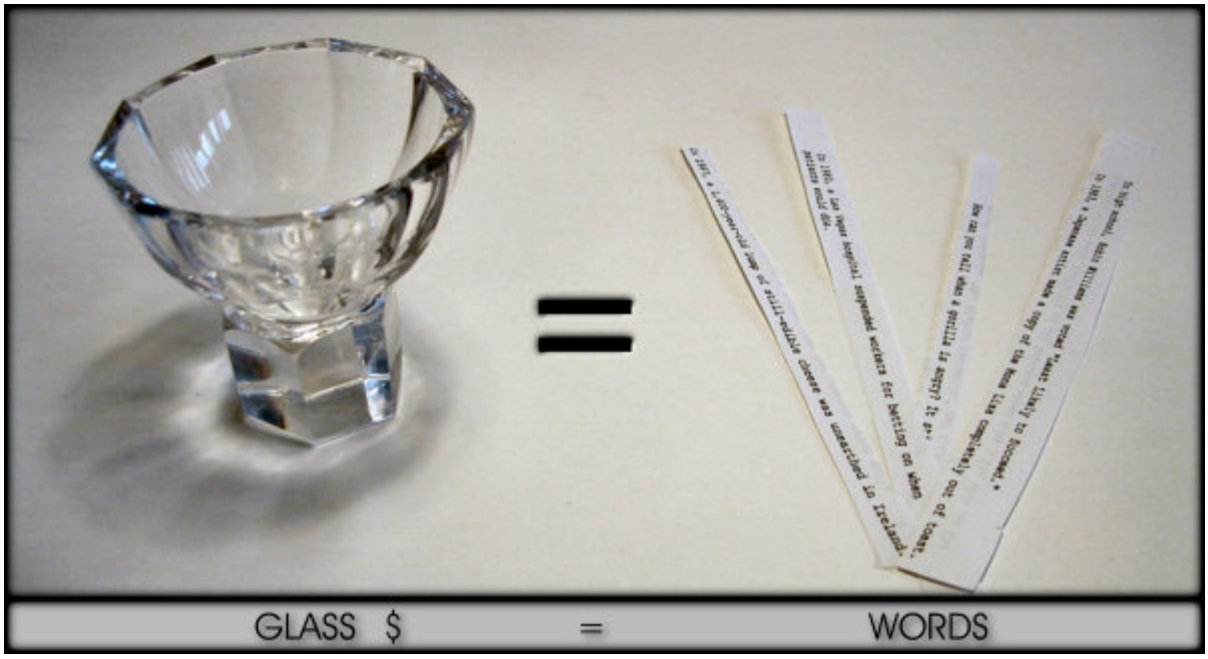
String variable label consits of **name** and **$** sign.
Here are some examples of valid labels:

i$	n$	name$	1st_user$
Main_Title$	TextContent$	RegKey$	glass$

Never forget to attach $ after label name. That suffix tells MMB "This is string variable".

It is recommended to use meaningful & descriptive labels, not too short, not too long - just enough to describe it's content. If you'll use string variable to store user's answer to displayed question, label of that variable should be UserAnswer or AnswerNo1 .

Of course, important part of this story is - **assigning of variable content**.



In MMB script code lines, it is being done by:

a) writing label name:

sentence

b) adding suffix:

$

c) followed by equal sign:

=

d) and writing content (value) of variable at the end, behind label, $ suffix and equal sign.
Contents is enclosed in single quotes:

```
'My first sentence in MMB string variable !'
```

All together, MMB code line for assigning of sentece to string variable looks like this:

```
sentence$= 'My first sentence in MMB string variable !'
```

There ! With this line you instructed MMB: "Take a string variable **sentence** and **put** sentence **My first sentence in MMB string variable !** in it."

Where will you give that instruction ? In MMB's Script Editor, of course.

Now, you're probably wondering: "What should I do with strings that contain **'** character ?"
If you have a string:

```
What's this ?
```

...in MMB's string variable it will look like:

```
sentence$= 'What\'s this ?'
```

You use **backslash ** to tell MMB "here ya go, I'm using single quote in my string variable..."

While backslash is obviously being used for this case in string variables, we must take care of the case when backslash is used for path to some directory (folder). String variable containing path to folder looks like this:

```
path$= 'c:\windows\\'
```

Example above contains **double backslash**, needed at the end of the string.
Explanation is very simple - MMB uses one backslash before **'** character so it can display that **'** character.
If you have path in string, last backslash must be understood as "hey, we've got the end of a string, not **'** character" , and the only way to do this is by using - double backslash.

Very nice. And just like in numerical variable case, the question is: can you only assign fixed words and never change 'em again ?

Nope !

It is important to have ability of **assigning value of one variable to the value of another variable**. Here's an example. If you have one bowl with words in it:

	*...what will you do to fill another bowl with words ?* **You will copy words from the first bowl**	
bowl1	**=**	**bowl2**

In MMB script code lines, it is being done by:

a) writing label of **destination** string variable (one that **accepts** value) with **$** suffix:

```
bowl2$
```

b) followed by equal sign:

=

c) and writing label of **source** string variable (one that **gives** value) with **$** suffix:

```
bowl1$
```

All together, MMB code line for assigning content of one string variable to another string variable looks like this::

```
bowl2$=bowl1$
```

If variable **bowl1$** contains word ***Hello***, code line above will assign the same value to **bowl2$** .

Usage of variable-to-variable approach is recommended when you're planning some changes on string variable and don't want to loose original value.
When scripting in MMB, you don't have to worry about corrupting the original - just simply take another variable and perform changes on it.

To merge strings and put them into one string variable, use + operator.

For example, if definition of first string variable looks like this:

```
FirstPart$='To be - or not to be...'
```

And definition of second string variable is:

```
SecondPart$='the question is now.'
```

Merging of these strings into one looks like this:

```
CompleteSentence$=FirstPart$+SecondPart$
```

Merged strings are contents of CompleteSentence$:

To be - or not to be...the question is now

The same goal would be achieved by directly assigning text (but it's fixed then, can't be changed) to CompleteSentence$:

```
CompleteSentence$= 'To be - or not to be...' + 'the question is now.'
```

Here are most common combinations of string variable assignments:

Definition	Example
Text to variable: fixed (non-changable)	name$='Johnny Be Good'

text is being assigned to string variable.	
Variable to variable: value of one string variable is being assigned to another string variable (for example, to retrieve value of MMB's EditBox object).	UserPassword$=EditBoxInput$
Variable+Text (or vice versa) to variable: using **+** for appending, this combination 1) merges contents of source string variable and source (fixed) text, 2) assigns result to destination string variable	a$=UserName$+' is hungry'

Contents (or results) of right-side strings are assigned to string variable on the left and you can use given values wherever you want, for example to show 'em as message:

```
a$=UserName$+' is hungry'
Message("Guess what: ","a$")
```

These two code lines will

a) assign strings to UserName$, and
b) display it using MMB's message box

Here's an example of multiple code lines with append operations:

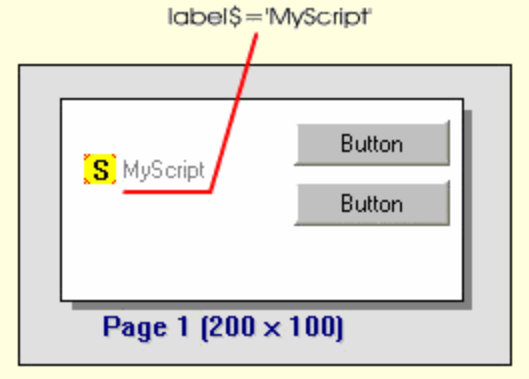
```
Player1$='Doug'
Player2$= 'Steve'
MatchPlayers$= Player1$+' vs '+ Player2$
Message("And now, match: ","MatchPlayers$")
```

String displayed in MMB's message box would be:

And now, match: Doug vs Steve

String variables as object labels

Flexibility of MMB enables using content of string variable as a source for object labels, used for various script commands:



As an image above shows, object label "MyScript" is assigned to string variable (label$). Once label is there, every script command that refers to some object label can use string variable instead of (fixed) label. So first you assign object label to a string variable:

```
ObjectLabel$= 'MyScript'
```

...and then execute script command that requires object label. For example, **RunScript** command uses one parameter - script label. Here we set string variable instead of fixed script label:

```
RunScript("ObjectLabel$")
```

This feature is very useful when objects have uniformed labels (Text1, Text2, Text3...) and performed commands should refer to variable labels - usually in **for..next loops**.

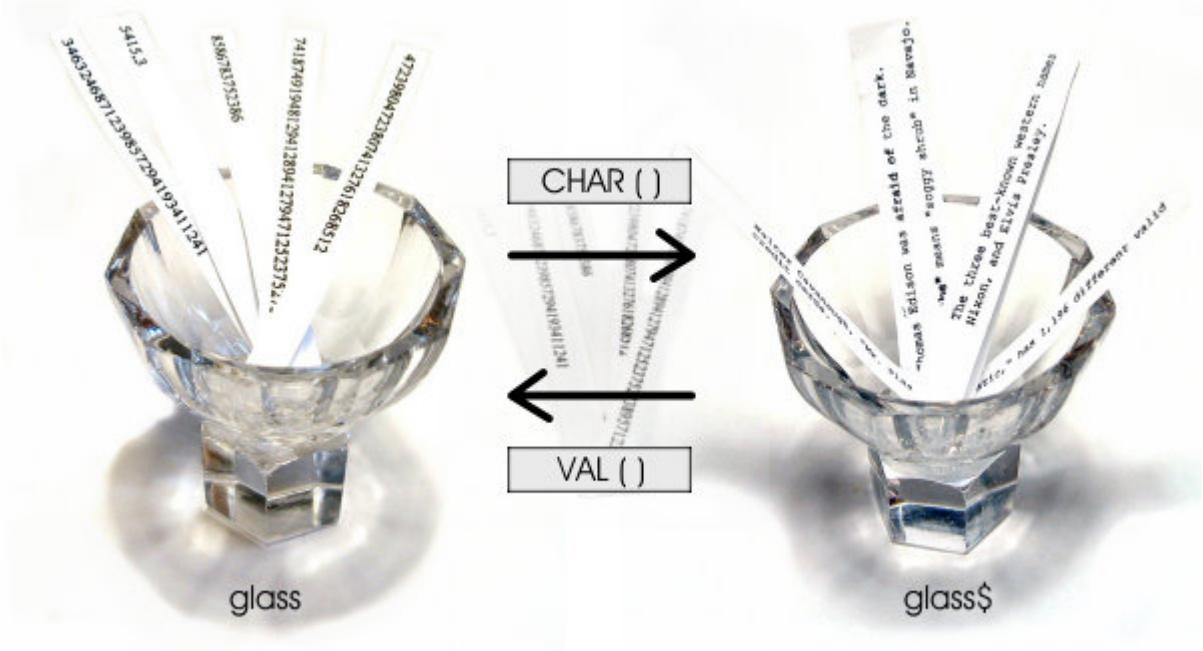
Examples mentioned above show you some uses of string variables. There are numerous purposes & combinations, so we leave you here to experiment and enjoy the power of strings!

12.8.4 Variable-related Functions

Travelling ? It's a common practice to learn basics of foreign language to make your visit more comfortable. You can, of course, pay translator to do the job. Doing educational courses ? You'll give your best to translate knowledge to acceptable form, comprehendable by the course group.
Using computer ? By time you've read this paragraph, it performed millions of translations.

We're obviously talking about MMB, using some translations in script language too ! In sections above we have mentioned string and numerical variables with all their differences.

Using of variable-related functions makes variable differences disappear.



And to achieve that goal, we'll need a kind of translation. In MMB case, we'll perform **translations between numerical and string variables**.

CHAR(NumVariable)

converts *numerical variable* to a *string variable*

Specifying of numerical variable we want to convert to string is done by using **source** numerical variable label inside parenthesis:

```
CHAR(NumOfYears)
```

This would hardly make sense without specifying **destination** string variable, where converted numerical variable will be put. So we also need one string variable for the left

side of code line:

```
Years$
```

Put together, code line for numerical -> string conversion looks like this:

```
Years$ =CHAR(NumOfYears)
```

Script line above will convert content of NumOfYears numerical variable to string and assign result to string variable Years$.

This type of conversion (numbers to text) is used most often. Here's an example of displaying advanced string with data retrieved from numerical variable:

```
NumOfYears=36
UserAge$= CHAR(NumOfYears)+' years old'
Message("Our user is ","UserAge$")
```

Result of these code lines will be displayed in message box:

Our user is 36 years old

From this example is visible for what you'll use CHAR conversion - when you want to display numbers from numerical variable in string / text content.

VAL(StringVariable$)

converts *string variable* to *numerical variable*

Specifying of string variable we want to convert to number is done by using **source** string variable label inside parenthesis:

```
VAL(Year$)
```

This function requires specifying of **destination** numerical variable, where converted string variable will be put. So we also need one numerical variable for the left side of code line:

```
Year
```

Put together, code line for string -> number conversion looks like this:

```
Year =VAL(Year$)
```

Script line above will convert content of Year$ string variable to a number and assign result to numerical variable Year .

Here's an example of using VAL function to get year, perform math-related operation and display result in the message box:

```
Year$='2003'
Year=VAL(Year$)
Year=Year+10
Message("For ten years it will be: ","Year")
```

Result of this code lines will be displayed in message box:

For ten years it will be: 2013

From this example is visible for what you'll use VAL conversion - when you want to perform math operations on some number that was previously a string (text).

INT(NumVariable)

rounds floating point number to lower integer value

Specifying of numerical variable we want to round value for is done using **source** numerical variable label inside parenthesis:

```
INT(CarTax)
```

This function requires specifying **destination** numerical variable, where integer number will be put. So we also need one numerical variable for the left side of code line:

```
RoundCarTax
```

Put together, code line for floating point -> integer conversion looks like this:

```
RoundCarTax =INT(CarTax)
```

Script line above will round floating point content of CarTax numerical variable to integer and assign result to numerical variable RoundCarTax .

Here's an example of using INT function in practice:

```
CarTax=14.32
RoundCarTax=INT(CarTax)
Message("Rounded tax for car is: ","RoundCarTax")
```

Result of this code lines will be displayed in the message box:

Rounded tax for car is: 14

Notice how variable is rounded to lower value. This is being done in all cases:

```
CarTax=14.99
RoundCarTax=INT(CarTax)
Message("Rounded tax for car is: ","RoundCarTax")
```

Result of INT function in this case will not be rounded to a higher value:

Rounded tax for car is: 14

INT function is in MMB mostly used for percentage counting (progress bars, gauges).

ABS(NumVariable)

rounds floating point number to a lower integer & absolute value

What "absolute" means ? Number can't be negative. If numerical variable content is lower than zero, negative prefix is removed and number becomes positive (above zero).

Specifying numerical variable we want to round & absolute value for is performed using **source** numerical variable label inside parenthesis:

```
ABS(CupsOfCoffee)
```

This function requires specifying **destination** numerical variable, where round & absolute number will be put. So we also need one numerical variable for the left side of the code line:

```
CoffeeCups
```

Put together, code line for real number -> absolute number conversion looks like this:

```
CoffeeCups =ABS(CupsOfCoffee)
```

Script line above will (optionally) round floating point content of CupsOfCoffee numerical variable to integer, check if number is negative so it can be converted to positive value and assign result to the numerical variable CoffeeCups .

Here's an example of using ABS function in practice:

```
CupsOfCoffee=-5.5
CoffeeCups=ABS(CupsOfCoffee)
Message("Number of coffee cups to order: ","CoffeeCups")
```

Result of these code lines will be displayed in message box:

Number of coffee cups to order: 5

ABS function is mostly used for prevention of negative numerical values.

RND(NumVariable)

generates random number in the range specified using numerical variable

This is specific variable-related function. It does not convert input variable, but uses it as a range limiter when generating random numbers.

Specifying numerical variable that is upper limit for random number range is done by using **source** numerical variable label or fixed number inside parenthesis:

```
RND(UpLimit)
```

...or...

```
RND(48)
```

This function requires specifying **destination** numerical variable, where generated random number will be put. So we also need one numerical variable for the left side of a code line:

```
LottoNumber
```

Put together, code line for random number generator looks like this:

```
LottoNumber =RND(UpLimit)
```

...or...

```
LottoNumber =RND(48)
```

Script lines above will use content of numerical variable (UpLimit) or fixed number (48) as upper limit of generated random number and assign result to numerical variable LottoNumber . Generated number will be within range *0 - upper limit* .

Here's an example of using RND function in practice:

```
LottoNumber=RND(48)
Message("Today's lucky number is: ","LottoNumber")
```

Result of this code lines will display message box text...

Today's lucky number is:

...and append random number (RND function result) to it.

Calling RND function repeatedly will generate numbers that differ from previous ones. Of course, having limited range, numbers will eventually repeat.

12.8.5 Advanced String Functions

VAL(string$)

Description

Convert a string variable to numerical variable.

Code Examples

```
** Convert mystring$ variable to integer value and store it in RetVal
mystring$='1.222'
RetVal=VAL(mystring$)

** Convert string variables and count them
mystring_A$='20'
mystring_B$='10'
RetVal=VAL(mystring_A$) + VAL(mystring_B$)
```

Additional Info

For additional information about VAL check this link.

CHAR(number)

Description

Convert a numerical variable to string

This type of conversion (numbers to text) is used most often. Here's an example of displaying advanced string with data retrieved from numerical variable:

Code Examples

```
NumOfYears=36
UserAge$= CHAR(NumOfYears)+' years old'
Message("Our user is ","UserAge$")
```

Result of these code lines will be displayed in message box:

Our user is 36 years old

Additional Info

For additional information about CHAR check this link.

CHR(number)

Description

Returns the character (from ASCII table) with the specified ordinal (decimal) value (0-255).

Complete ASCII character table can be found here.

Code Examples

```
string$=CHR(169) + ' Odklizec 2003'
LoadText("Text","string$")
```

Result of these code lines will be displayed in Text object:

© Odklizec 2003

```
string$= 'First line' + CHR(13) + CHR(10) + 'Second line'
LoadText("Text","string$")
```

Result of these code lines will be text divided into two lines:

First line
Second line

ORD(character$)

Description

Returns the ordinal (decimal) value of the specified character.

Complete ASCII character table can be found here.

Code Examples

```
character$='#'
RetVal=ORD(character$)
```

Result of this code will be 35

LEN(string$)

Description

Returns the length of the specified string.

Code Examples

```
LoadText("string$","c:\test.txt")
RetVal=LEN(string$)
```

Result of this code will be length of the loaded file.

LOW(string$) / UPP(String$)

Description

Returns a string with the same text as the string passed in **String$** variable, but with all letters converted to LOWercase/UPPercase.

Code Examples

```
string$='ThIs TeXt WiLL bE CoNvErtED to LoWeRcAsE'
RetString$=LOW(string$)
```

Result of this code will be text converted to lowercase.

this text will be converted to lowercase

```
string$='ThIs TeXt WiLL bE CoNvErtED to UppErCasE'
RetString$=UPP(string$)
```

Result of this code will be text converted to lowercase.

THIS TEXT WILL BE CONVERTED TO UPPERCASE

POS(SubString$, String$)

Description

Searches for **SubString** within **String** and returns an integer value (starting from

1) that is the index of the first character of **SubString** within **String.**
If Substr is not found, POS returns zero.

Code Examples

```
string$='Have a nice day!'
substring$='nice'
RetVal=POS(substring$,string$)
```

Result of the above code will be 8.

Additional Info

Because POS function (as well as other string functions) is case sensitive, we recommend you to convert the source string to upper or lowercase before starting POS search.

```
string$='Have a nice day!'
substring$='have'
RetString$=LOW(string$)
RetVal=POS(substring$,RetString$)
```

If you don't convert the source string$ to Lowercase, then POS returns 0, because 'Have'<>'have'. But after converting string$ to Lowercase POS returns 1.

NOL(FileName$)

Description

This function will return number of lines from FileName$.
If file doesn't exist NOL returns zero.

Code Examples

```
OpenFile("txt Files (*.txt)|*.txt|All Files|*.*||","*.txt")
If (OpenFile$<>'') Then
  RetVal=NOL(OpenFile$)
End
```

Result of the above code will be number of lines in selected file. If none file will be selected then NOL function will not be performed.

StrCopy(String$, Indx, Count)

Description

Returns a string containing Count characters starting with at **String$**[Index].
If **Index** is larger than the length of **String$**, Copy returns an empty string.
If Count specifies more characters than are available, then only the characters from String$[Index] to the end of String$ are returned.

Code Examples

```
string$='Have a nice day!'
ReturnExt$=StrCopy(string$,1,4)
     Returns Have
```

This more advanced StrCopy example will show you how to make a simple text typing animation. Simply insert this code to a button and insert new Text object into project.

```
string$='Have a nice day!'
** count number of letters
count=LEN(string$)
** create loop from 1 to number of letters
For i=1 to count
** copy one character per loop
  ReturnExt$=StrCopy(string$,i,1)
** pause loop
  Pause("100")
** merge letters (this will produce animation effect)
  newstring$=newstring$ + ReturnExt$
** load merged string to a Text object
  LoadText("Text","newstring$")
Next i
```

StrDel(String$, Indx, Count)

Description

Removes a substring of Count characters from string **String** starting at **String**[Index].
If Index is larger than the length of **String**, no characters are deleted.
If Count specifies more characters than remain starting at the **String**[Index], StrDel removes the rest of the string and Returns a modified string.

Code Examples

```
string$='Have a nice day!'
ReturnExt$=StrDel(string$,1,5)
```

```
    Returns a nice day!
```

This more advanced StrDel example will show you how to separate Total and Free memory obtained as a single string from GetMemory function.

```
RetMem$=GetMemory()
separ$='/'
**Get Total memory
n=POS(separ$,RetMem$)-1
TotalMem$=StrCopy(RetMem$,0,n)
n=n+1
**Get free memory
FreeMem$=StrDel(RetMem$,0,n)
```

StrIns(SourceStr$, DestStrs$, Indx)

Description

Merges **DestStr$** into **SourceStr$** at the position **S**[index]. Returns a modified string.

Code Examples

```
sourcestr$='Have a nice day!'
deststr$=' too'
** count number of letters
count=LEN(sourcestr$)
ReturnStr$=StrIns(sourcestr$,deststr$,count)
    Returns Have a nice day too!
```

StrGet(String$,Int)

Description

Returns the I-th character in **String$**.

Code Examples

```
string$='Have a nice day!'
count=LEN(string$)
ReturnStr$=StrGet(string$,count)
```

Returns "!" (without the quotes).

StrSet(String$, Int,C$)

Description

Set the I-th character in **String$** to character **C$**. Returns a modified string.

Code Examples

```
string$='Have a nice day!'
count=LEN(string$)
c$='!!!'
ReturnStr$=StrSet(string$,count,c$)
    Returns Have a nice day!!!
```

StrOfChar(C$, Int)

Description

Returns a string of length **I** with all characters set to character **C$**.

Code Examples

```
c$='#'
ReturnStr$=StrOfChar(c$,5)
    Returns #####

** use CHR for obtaining special characters from ASCII table
c$=CHR(177)
ReturnStr$=StrOfChar(c$,5)
    Returns ±±±±±
```

StrChange(String$, FromStr$, ToStr$)

Description

Change all occurrences of **FromStr$** in **String$** to **ToStr$**. Returns a modified string.

Code Examples

```
string$='Lord of the Rings'
fromstr$='Rings'
tostr$='Beer'
ReturnStr$=StrChange(string$,fromstr$,tostr$)
     Returns Lord of the Beer
```

StrToFile(FileName$, String$, Append, LineFeed)

Description

Saves or appends the specified **String$** to the specified file. Returns **1** if successful, **0** otherwise.
If **Append** is True (or 1) and the specified file does not exist, a new file is created.
If **Append** is True (or 1) and the specified file exist, the **String$** will append right after last character in the specified file.
If **LineFeed** and **Append** are True (or 1), then the specified string will append on next line.
If **LineFeed** and **Append** are False (or 0), then the specified string will saved in new file or overwrite the existing one.

Code Examples

```
file$='c:\temp\test.txt'
string$= 'this string will be append to text file'
ReturnVal=StrToFile(file$,string$,TRUE,FALSE)
     This save string$ to test.txt
file$='c:\temp\test.txt'
For i=1 To 10
  string$= 'Line ' + CHAR(i)
  ReturnVal=StrToFile(file$,string$,TRUE,TRUE)
Next i
     This save 10 lines (Line 1...Line 10) to test.txt
```

StrToLine(FileName$, String$, ToLine, Overwrite)

Description

Add or replace the line inside text file. Returns **1** if successful, **0** otherwise.

FileaName$ = path to text file (not binary files!)
String$ = string to add/replace to line
ToLine = number of line to add/replace
Overwrite = TRUE/FALSE - if true, the original line is replaced. In case of false, the line is added to ToLine and the rest of file is moved down.

Code Examples

```
file$='c:\temp\test.txt'
string$= 'this string will be added to text file'
** this adds the string$ to 5th line of test.txt and move the rest of file down
ReturnVal=StrToLine(file$,string$,5,FALSE)

** this replace the 5th line of test.txt with string$
ReturnVal=StrToLine(file$,string$,5,TRUE)
```

StrFromFile(FileName$, FromLine, NumOfLines)

Description

This will load the entire file **FileName$** (or just single line) to string variable. **FromLine** is the number of line from which will be file loaded and **NumOfLines** is a number of lines that will be loaded. If **NumOfLines**=-1, it will load entire file. If **NumOfLines**=-1 and **FromLine**>0 then it will load rest of file from the line determined by **FromLine**

Code Examples

```
file$='c:\temp\test.txt'
fromline= 1
numoflines=10
ReturnStr$=StrFromFile(file$,fromline,numoflines)
    Returns first 10 lines from test.txt
```

ExtractExt(FileName$)
ExtractDir(FileName$)
ExtractName(FileName$)
ExtractDrive(FileName$)

Description

ExtractExt(FileName$)
Extracts the extension part of the given file name (path).

ExtractDir(FileName$)
Extracts the drive and directory parts of the given path.

ExtractName(FileName$)
Extracts the name and extension parts of the given file name (path).

ExtractDrive(FileName$)
Returns a string containing the 'drive' portion of a fully qualified path name for the file passed in the FileName (path).

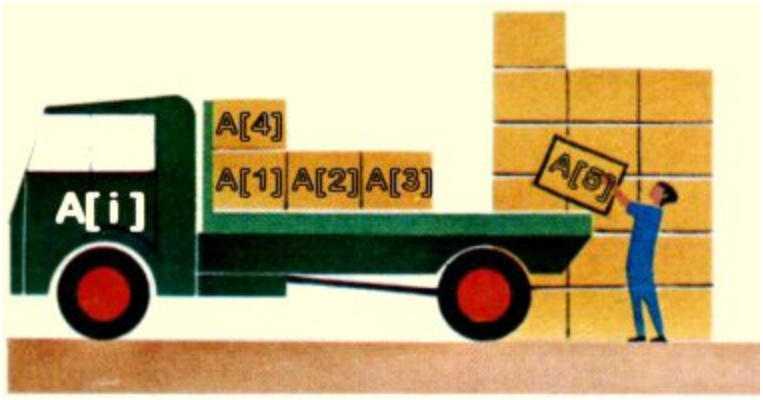
Code Examples

```
Path$='c:\MyFiles\text.txt'
ReturnExt$=ExtractExt(Path$)
     Returns .txt

ReturnDir$=ExtractDir(Path$)
     Returns \MyFiles\

ReturnName$=ExtractName(Path$)
     Returns text

ReturnDrive$=ExtractDrive(Path$)
     Returns c:
```

12.8.6 Arrays

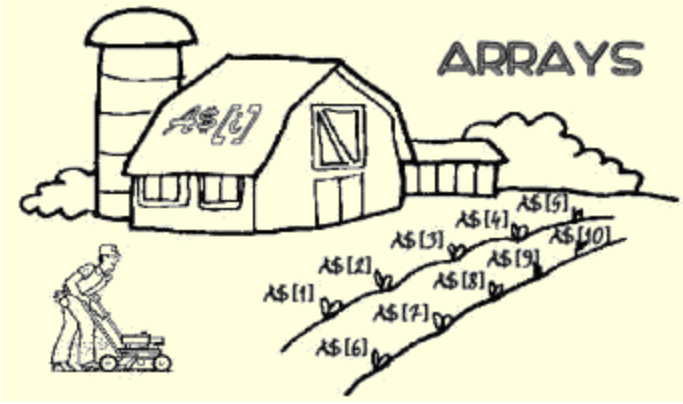
Remember those tool shelves and drawers in garage ? They have function of arrays - you put similar tools on the same shelve or drawer. Screws of different size go to different drawers.

Array is also shipment of some stocks in the truck:



On this truck you can see how entire shipment contains similar items (as the label on truck doors suggests). Every item has it's own number.

That's a reason why one would want to use arrays in program - they help programmer to hold similar data in one *container*. It's easier to load all similar items in one truck, than requesting many trucks with different names.



MMB uses arrays too. We call them *1-dimensional arrays*, because items are stored in one "line" (dimension) - when your program puts or requests data from this kind of array, it uses only one coordinate (ordinal number of item) to go through array and perform operation on desired item.

Array in MMB is a group of variables under the same name, with addresses, so we can reach every item in array. In MMB script language, arrays are used just like variables.

Let's compare them...

Variable	Array
a$	a$[n]
name$	name$[n]
clicks	clicks[n]
user_address$	user_address$[n]
user_counter	user_counter[n]

On the left side are variables (**string and numerical**), and on the right are those same variables but in array form - extended form of variables, while we can store more items in arrays.

Let's take one array:

user_counter[n]

Here we have **label of array** (**user_counter**), two square brackets **[ ]** and item index (**n**). This is basic form of array. Array label can be specified just like in variable case. Square brackets enclose **index number** - either numerical variable or plain number that instructs MMB what item we want to retrieve from array.

If you've read section of manual about integer and string variables, you know they can be used almost anywhere in MMB. One of variable roles is to specify address of an item in array. These addresses are numbers, so we use numerical variables.

Here are some examples:

Arrays where item address is specified through ordinal numbers:

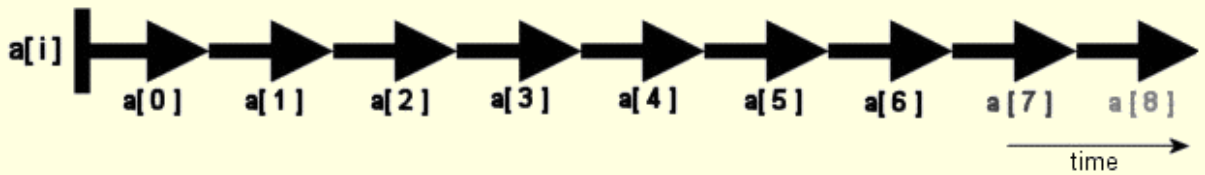
```
a[1]
twain$[72]
my_finger[534]
tool$[102]
```

Arrays where item address is specified through numerical variable:

```
a[i]
twain$[chapter_no]
my_finger[finger_number]
tool$[tool_drawer]
```

Arrays are frequently used in for...next loops, where variables are better solution for items addresses.

When you fill array, newest item gets highest address in array:



To specify item address you can even use math operations:

```
a[i + 1]
twain$[chapter_no - 4]
my_finger[finger_number * 8]
tool$[ tool_drawer / 2 ]
```

And how will you fill arrays ? Depends, what kind of array you're using:

Numerical arrays

Like numerical variables, MMB can use numerical arrays to **store numbers**. We use them for math operations. Both floating point and round numbers can be stored (83.21, 10, 294.1).

Let's fill some numerical arrays:

```
a[1] = c
my_finger[i] = 15
electricity[f + 1] = variable_volts
```

First example: array called " **a[]** " is filled with variable " **c** ", on item address **1**.

Second example: array called " **my_finger** " is filled with number " **15** ", on item address received from variable " **i** " .

Third example: array called " **electricity[]** " is filled with variable " **variable_volts** ", on item address received from calculation: **variable " f " + 1** .

Reading of items is similar:

```
c = a[1]
finger = my_finger[i]
power = electricity[f + 1]
```

First example: item on address " **1** ", from array " **a[]** ", is copied to numerical variable " **c** " .

Second example: item on address received from variable " **i** " , in array called " **my_finger** ", is copied to variable " **finger** " .

Third example: item on address calculated from variable " **f** " **+ 1**, in array called " **electricity[]**", is copied to variable " **power** " .

String arrays

Like MMB string variable, this kind of array - string array - can store words, sentences, paragraphs of text and numbers. Basic difference in comparison with variables is possibility of storing many individual items.

String array looks like this:

```
a$[i]
```

It contains **name of array** (**a**), **$ suffix** that tells MMB we're using string array, square brackets **[ ]** and item address (numerical variable or ordinal number).

Filling string arrays looks like this:

```
a$[1] = c$
finger_names$[i] = name$
library$[f + 1] = book$
```

First example: array called " **a$[]** " is filled with content of variable " **c$** ", on item address **1**.

Second example: array called " **finger_names$[]** " is filled with content of variable " **name$** ", on item address received from variable " **i** " .

Third example: array called " **library$** " is filled with content of variable " **book$** ", on item address received from calculation: numerical variable **" f " + 1** .

Reading items from string arrays:

```
c$ = a$[1]
name$ = finger_names$[i]
book$ = library$[f + 1]
```

First example: item on address " **1** ", from array " **a$[]** ", is copied to string variable " **c $** " .

Second example: item on address received from numerical variable " **i** " , in array called "**finger_names$[]**", is copied to string variable " **name$** " .

Third example: item on address calculated from numerical variable " **f** " **+ 1**, in array called " **library$[]** ", is copied to string variable " **books$** " .

Examples of arrays in loop

As already mentioned, arrays are frequently used in loops. Here's one example:

```
for i=1 to 50
  a[i] = i
```

```
next i
```

This code will fill 50 items of numerical array called " **a[]** " with numbers from 1 to 50. Address of item is received from for...next loop.

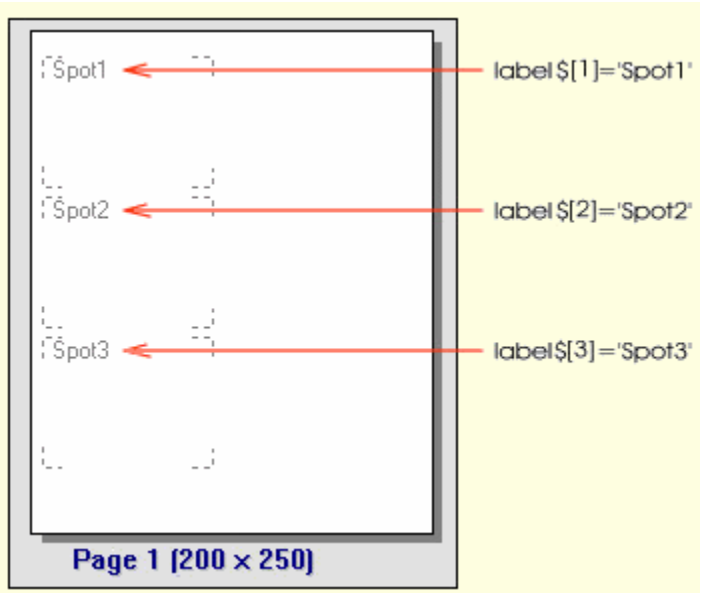
Reading values after filling array can be done using loop too:

```
for i=1 to 50
  a = a [i]
  Message("Array item contents: ","a")
  Pause("200")
next i
```

Address of item is received from *for...next* loop. Every item is copied to numerical variable " **a** ", and then displayed in MMB's message box.

String arrays as object labels

Flexibility of MMB enables using content of string arrays as a source for object labels, used for various script commands:



As an image above shows, HotSpot object labels Spot1, Spot2 and Spot3 have been assigned to string array label$[], having items label$[1], label$[2] and label$[3] .

Once labels are there, every script command that refers to some object label can use string array item instead of (fixed) label. So first you assign object label to string array item:

```
label$[1]= 'Spot1'
```

...and then execute script command that requires object label. For example, **RunScript** command uses one parameter - script label. Here we set string array item instead of

fixed object label:

```
RunScript("label$[1]")
```

This feature is very useful when objects have uniformed labels (Text1, Text2, Text3...) and performed commands should refer to variable labels - usually in **for..next loops** :

```
for i=1 to 10
  text$ = 'Hello no. '+CHAR(i)
  label$[i]='Text'+CHAR(i)
  LoadText("label$[i]","text$")
next i
```

For full understanding of MMB arrays it is highly recommended to read manual sections on variables and loop subjects.

12.8.7 Array Functions

GetArrayItem(Array$,SeparatorChar$, Position)

Description

This function returns item from **Array$** at defined **Position**. **SeparatorChar$** is an array delimiter character.

Remember that each array item must be delimited by separator character. In fact, it can be any character from ASCII table (0-255), but we recommend to use less usual characters (#, *, /,|...).
Extended characters (127-255) can be added by CHR(number) code.

Code Examples

This code show item1..item5 in message box.

```
Items$ = 'item1#item2#item3#item4#item5#'
For i = 1 To 5
  ArrayItem$ = GetArrayItem(Items$,#,i)
  Message("Obtained array item: ","ArrayItem$")
Next i
```

Here is more advanced example:
This code returns selected item(s) from ListBox object

```
ListBoxGetSelectedItems("ListBox","Items$,ItemsNum$,#,NumItems")
For i=NumItems To 1
  ArrayItem$ = GetArrayItem(Items$,#,i)
  Message("Selected item: ","ArrayItem$")
Next i
```

Additional Info

Coma (,) character cannot be (by default) used as an array delimiter. However, there is a trick, how to use it ;)

Instead of this..
Name$=GetArrayItem(String$,',',1)
or this..
Name$=GetArrayItem(String$,,,1)

use this..
Name$=GetArrayItem(String$,'**0x2C**',1)

..where the 0x2C code is just the hexadecimal representation of the coma character (in ASCII table).

GetArrayNum(Array$,SeparatorChar$)

Description

This function returns a number of Items in the **Array$** with predefined delimiter (**SeparatorChar$**).

This function is useful in cases if you don't know the number of items in the array.

Code Examples

This code show item1..item5 in message box.

```
Items$ = 'item1#item2#item3#item4#item5#'
NumOfItems=GetArrayNum(Items$,#)
For i = 1 To NumOfItems
  ArrayItem$ = GetArrayItem(Items$,#,i)
  Message("Obtained array item: ","ArrayItem$")
Next i
```

12.9 Script Flow Functions

12.9.1 Introduction

Imagine how strange would it be to drive a car that can move only straight forward, without steering wheel to turn it either left or right. Strange would also be to have a phone that can connect only to one phone. And what about a TV you can't change channel on ?

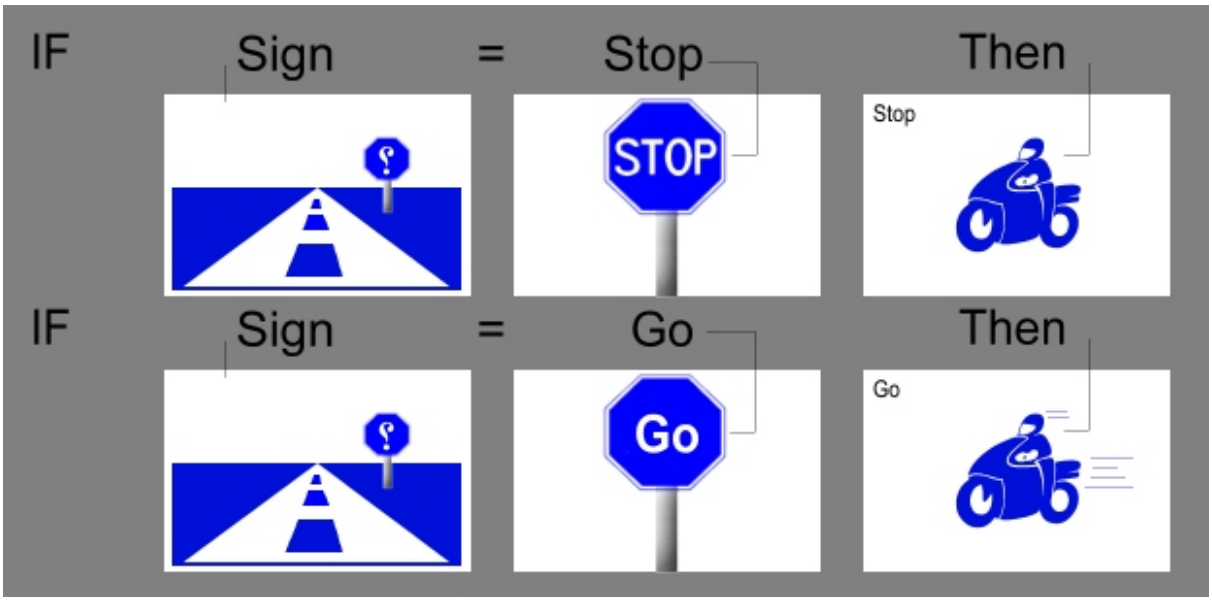
We're lucky to only read about these funny cases where the lack of choice could lead to a not very happy end. While computer software likes happy ends, it contains variety of functions, and some of them deal with **flow control**. Giving ability of choice, programmers make software interactive. Meaning, user is not watching a movie, but participating in interactive sessions between software and it's user.

The car will take you almost wherever you want, thanks to the steering wheel. To make MMB application go at desired path, you'll use steering wheel too - flow control functions !

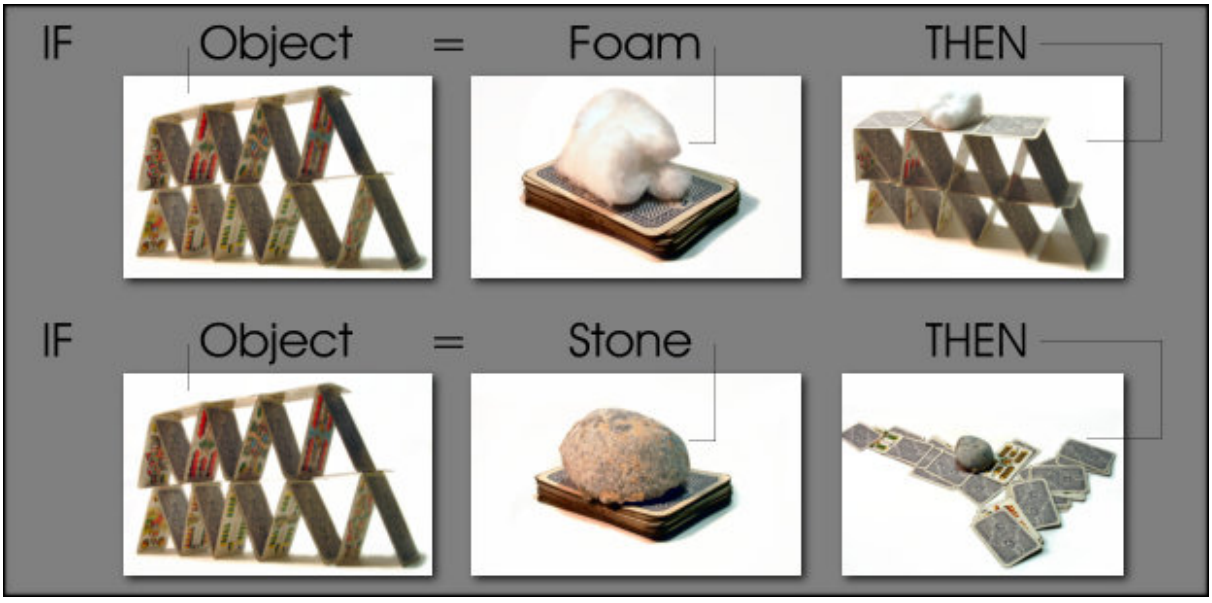
MMB uses two ways of controlling script flow: **If statements** and **For..Next loops.**

12.9.2 If..Then Statements

Introduction



Making decisions is what if statements are all about. By reading road signs you'll adjust the speed or direction of your vehicle. Hitting the ball in the right direction will make a goal for your team. Adding extra pepper will make your meal spicey.



As you can see from the image above, depending on what you put on the house of playing cards - that house will either keep standing or collapse.

MMB uses variable label as platform for variable value (object you put on the platform).

Basic If statement syntax

Script code lines consist of:

a) writing **If** clause:

```
If
```

b) opening parenthesis with variable label:

```
(object$
```

c) followed by equal sign:

```
=
```

d) then writing contents (value) of variable at the end, behind label and equal sign, followed by closing parenthesis:

```
'rock')
```

e) and adding of then clause at the end of the first line:

```
Then
```

Put together, first line of *if statement* will look like this:

```
If (object$='rock') Then
```

This line instructs MMB to perform certain actions if content of string variable **object$** matches exactly the string specified inside single quotes (in this case: **rock**).

And what if variable really matches specified content ? After the line above, you will continue writing script lines that will be executed in the case of content match:

```
MyText$='Object put to platform is: '+object$
Message("","MyText$")
```

When you're done writing code lines that should be performed in match case, you can use another clause:

```
Else
```

...and continue writing lines that should be performed if variable doesn't match specified content. For example:

```
MyText$='Unknown object has been put to platform'
Message("","MyText$")
```

Adding of else clause and script lines of else case are optional. But, whatever you decide, you must tell MMB when you're done with if statement, by adding the **end**

clause:

```
End
```

Let's see now how **entire if statement code** looks like !

```
If (object$='rock') Then
   MyText$='Object put to platform is: '+object$
    Message("","MyText$")
Else
    MyText$='Unknown object has been put to platform'
    Message("","MyText$")
End
```

Translated: if content of string variable **object$** is the rock, text assigned to **MyText$** string variable will be "Object put to platform is: rock" and it'll be displayed in MMB's message box.
But if content of string variable object$ is something else, text assigned to **MyText$** string variable will be "Unknown object has been put to platform" and it'll be displayed in MMB's message box. All lines put after **Else** clause will be executed only when **If** clause is false.

Of course, you don't have to use **Else** if you don't want to:

```
If (object$='rock') Then
   MyText$='House of cards has collapsed !'
    Message("","MyText$")
   Return()
End
```

These code lines will display message box with text "House of cards has collapsed !" if **object$** variable contents is **rock** , and after that it'll call Return() command to stop going any further in script code lines (both in & out of if statement code lines).

Multiple If statements

Another option is to use **multiple If statements**, to make more decisions:

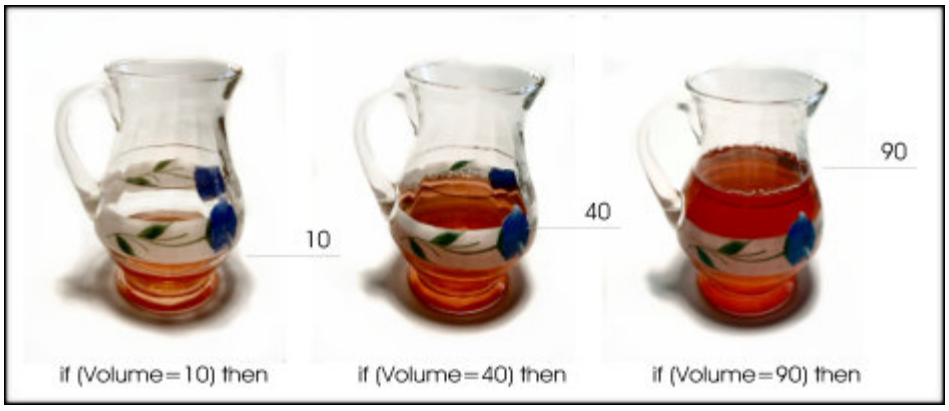
```
If (TankVolume=10) Then
    Message(" Please add some more fuel ","")
   Return()
End
If (TankVolume=50) Then
    Message(" Tank is half full ","")
   Return()
End
If (TankVolume=100) Then
```

```
    Message(" Tank is full ! ","")
   Else
   Message("Tank status unknown","")
End
```

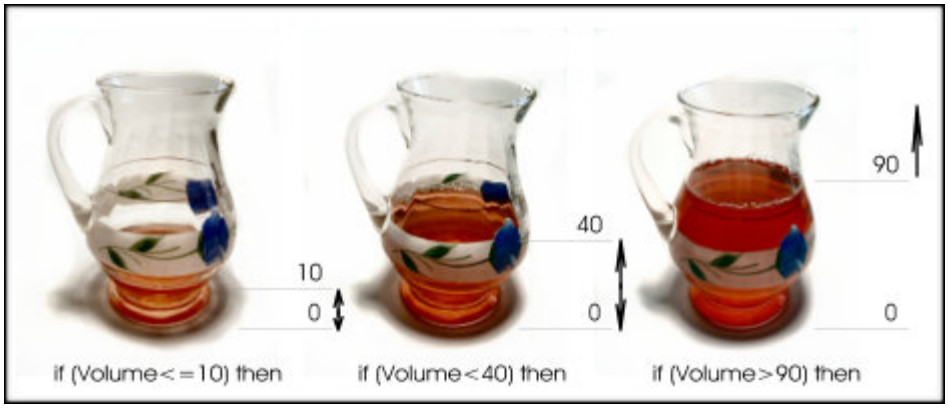
The first thing you'll notice above is - we're using numerical variables. The principle is completely the same as for string variables. Here we take numerical variable **TankVolume** that contains some value. Once it goes through multiple if statements, we'll have message box pop up even when neither of *if cases* is **true**, because last *if statement* also includes **Else** clause, containing the code that will be executed if code lines didn't go in any other direction.

Using value range in If statements

It wouldn't be very handy to have only fixed values in if statements:



We would need too many if cases to handle all sub-values. This looks better:



So, not only equal operator is available, but variety of others:

Operator	Name
=	equals
<>	different than
<	less than
>	greater than
<=	less than or equal
=>	more than or equal

Put into code lines, here's how these operators look like:

```
If (Volume<10) Then
    Message("Volume is less than 10 !","")
End
```

(displays message box if content of numerical variable Volume is under 10)

```
If (Volume<=49) Then
    Message("Volume is under 50.","")
End
```

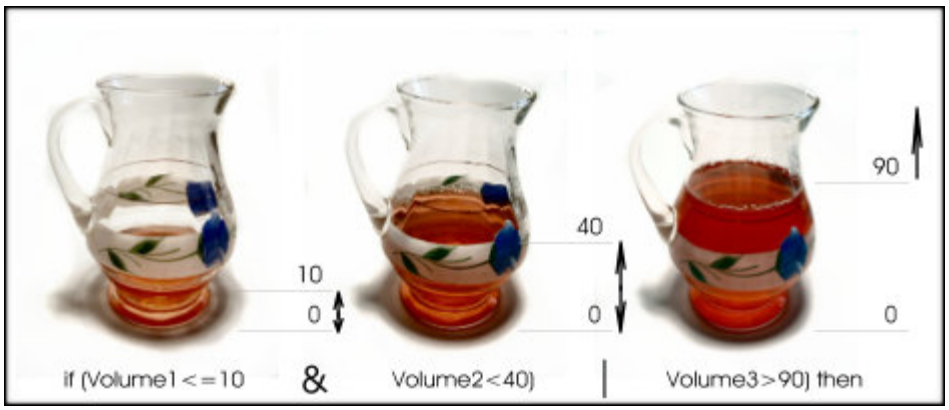
(displays message box if content of numerical variable is 49 or under that number)

```
If (Volume<>100) Then
    Message("Volume is not 100.","")
End
```

(displays message box if content of numerical variable is different than 100)

Multiple cases in If statements

Another neat feature of if statements in MMB is ability to use more conditions in one line.



There are two available operators:

Operator	Name
&	**and**
|	**or**

To use these, you'll start by writing an **If** clause:

```
If
```

...opening parenthesis with first variable label:

```
(OldUser$
```

...followed by equal sign:

```
=
```

...then writing content (value) of variable at the end, behind label and equal sign:

```
'John'
```

...and adding either **&** or **|** operator:

```
&
```

After writing this, you'll start adding the next case, without opening new parenthesis, but continuing writing variable name inside current one:

```
NewUser$
```

...followed by equal sign:

=

...and writing contents (value) of variable at the end, behind label and equal sign:

'Peter'

Now you can either continue adding another case, or finish the if statement code line by closing parenthesis:

)

... and adding **Then** clause at the end of the first line:

Then

Put together, first line of *If statement* with more conditions looks like this:

```
If (OldUser$='John' & NewUser$='Peter') Then
```

And what will this code line do ? Check value of **OldUser$** for a match with name **John**. If the match exists, it'll go and check if value of **NewUser$** matches name **Peter**. If that's the case, MMB will perform code lines entered after **Then** clause.

Here's an example with **|** (or) operator:

```
If (OldUser$='John' | NewUser$='Peter') Then
```

Case will be positive if either OldUser$ or NewUser$ matches the case, and MMB will perform code lines entered after **Then** clause. If neither match, if statement code lines are ignored.

And there are even more combinations here ! Variables can be compared with other variables, multiple cases can be combined between numerical and string variables, using of <> operator is available not only for numerical, but also for string variables. Nested if statements (one in another) are also available here.

Let's see how these features work, one by one...

Variable vs. Variable

In many occasions, values are stored in variables. It's possible to compare two variables in the if statements and check do they match:

```
If (User1$=User2$) Then
    Message("Users are identical !","")
End
```

Example above compares 2 string variables (User1$, User2$) and if they match, message box is displayed, saying "Users are identical !". Here's one example that uses numerical variables:

```
If (CurrentLevel>Recommended) Then
    Message("Level is higher than recommended !","")
End
```

Example checks if value of numerical variable CurrentLevel is higher than value of numerical variable **Recommended**. If it is, message box is displayed, saying "Level is higher than recommended !"

If statements with mixed variables

Another advantage of if statements with multiple cases is ability to look for matches on both string and numerical variables inside the same if statement. Here's an example:

```
If (UserName$='Bundy' & UserHeight>197) Then
    Message("Bundy is higher than 197 cm","")
End
```

Code above checks match on 2 variables - string variable **UserName$** and numerical variable **UserHeight**. If string variable contains name Bundy, and if numerical variable content is higher than 197, message box will display: "Bundy is higher than 197 cm".

Checking match with <>

In some cases you won't be interested in checking value range or match, but only see if either numerical or string variable differs from specified value. One of usual examples is checking of passwords, where your program only wants to know if user input matches (either variable or fixed) password. So let's see how this works:

```
If (UserInput$<>'SoftwareRegKey') Then
    Message("Entered software reg key is not valid !","")
End
```

You remember <> operator from previous paragraphs - and it's by default used for numerical variables. But in MMB you can use <> to search for a difference on string variables too.
Just for the record, here's an example with numerical variable:

```
If (MyHeight<>YourHeight) Then
    Message("We have different height !","")
End
```

Here we check the difference between two numerical variables. If difference exists, message box is displayed, saying: "We have different height !".

Nested If statements

In addition to multiple if statements, here's another feature that is actually longer version of multiple case if statement. It's possible to enter second if statement inside of the first one and have parts of the code executed depending on case match.

```
If (UserName$='Bundy') Then
   If (UserHeight>197) Then
       Message("Bundy is higher than 197 cm","")
   End
End
```

Although you could replace this example with shorter version:

```
If (UserName$='Bundy' & UserHeight>197) Then
    Message("Bundy is higher than 197 cm","")
End
```

...by using nested If statements it's possible to execute some other code inside of the first statement, that doesn't depend on the second if statement match case:

```
If (UserName$='Bundy') Then
 Message("Hello Bundy !","")
   If (UserHeight>197) Then
       Message("Bundy is higher than 197 cm","")
   End
End
```

Code lines above will check if string variable UserName$ matches name "Bundy". If it does, message box with "Hello Bundy !" text is displayed. After that, second (nested) if statement is checked with numerical variable UserHeight. If number is higher than 197, second message box will be displayed, saying: "Bundy is higher than 197 cm".
So, even if Bundy is not higher than 197 cm, first (Hello Bundy !) message box will be displayed. The same principle goes for code lines entered below entire second if statement.

Notice where nested if statements are entered - **End** clause of previous if statement is not put above 'em, but below, encapsulating nested statements. You can add as many

nested statements as you like and brake further execution with Return() command:

```
If (UserName$='Bundy') Then
 Message("Hello Bundy !","")
    If (UserHeight>197) Then
       Message("Bundy is higher than 197 cm","")
         If (UserWeight>110) Then
           Message("Bundy weighs more than 110 kilos","")
            Return()
         End
       Message("Bundy weighs less than/or 110 kilos","")
    End
End
```

If statements are marked with different colors so you can see which **End** clause belongs to which statement. First message box is displayed if **UserName$** matches name "Bundy". Second message box is displayed if numerical variable **UserHeight** contains number higher than 197. Now, third if statement (marked red) is interesting - if numerical variable **UserWeight** is higher than 110, message box will display "Bundy weighs more than 100 kilos" and that's where execution of all if statements is stopped with **Return()** . But if **UserWeight** is not greater than 110, message box will display "Bundy weighs less than/or 110 kilos".

12.9.3 For..Next Loops

Introduction

Keep walking.

Could you repeat that ?

Make 10 circles around the stadium !

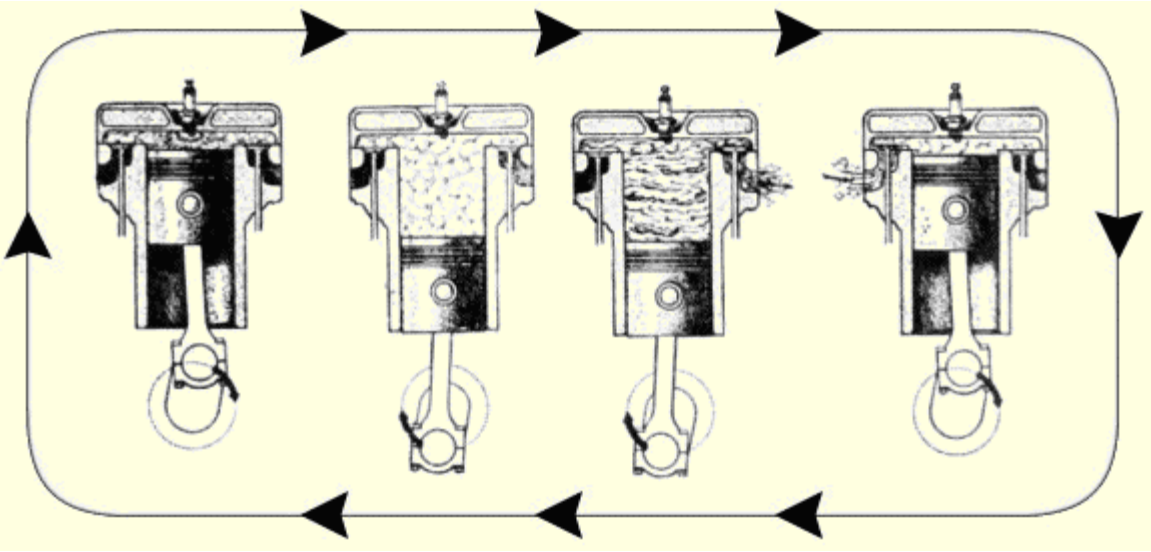
Above mentioned expressions are used every day, and nobody is used to say:

Keep making loops with your legs.

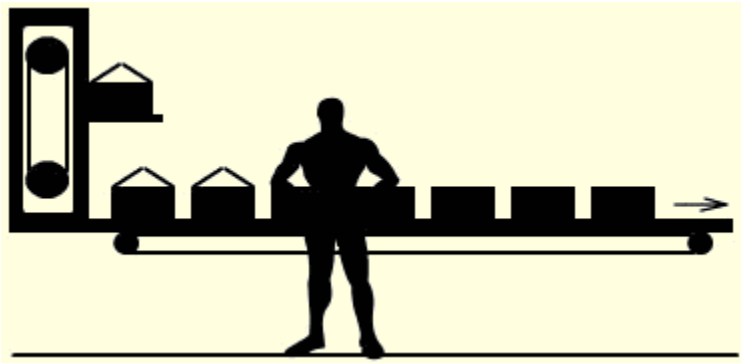
Could you loop what you just said ?

Loop around the stadium 10 times !

Although it sounds funny, at the same time it's true - moving one leg in front of another repeatedly is a loop, just as circulating around the stadium. Engines use loops too:

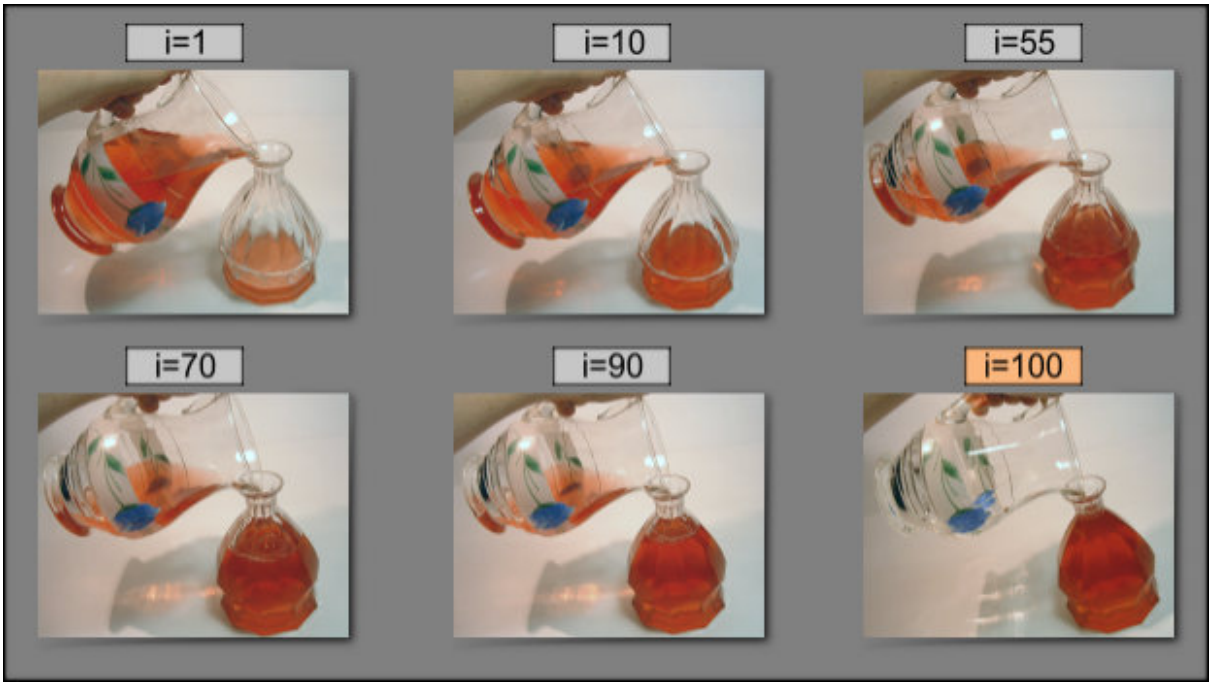


Millions of people work on factory lines, having loops as their work basics:



And in all these cases there's no help - actions must be repeated manually, sometimes with help from automatized machines. And talking about machines, did computers borrow this process too ? Yep, they did ! Programmers had enough of hard manual work, repeating the same routines again and again... so they added looping !

Loops & MMB



That's how it's done in MMB ! Pour commands, just like syrup, until you have enough !

To use loops, you'll start by writing a **For** clause:

```
For
```

...adding a numerical variable that will be used as a counter:

```
Counter
```

...followed by equal sign:

```
=
```

...then writing starting value for a range of **Counter** variable :

```
1
```

...and adding **To** clause:

```
To
```

After this, you'll write an ending value of range for **Counter** variable:

```
100
```

Let's see now how the first *for..next* loop line looks like:

```
For Counter=1 To 100
```

Line above tells MMB that everything written below should be repeated. And how many times ? From the starting value of numerical variable, to an ending value of that variable.

After setting loop properties with code line above, you can start writing code lines that will be performed in the loop. In code sample above, commands would be executed 100 times (as specified through Counter numerical variable). So, let's write some already-known commands:

```
var$='Hello from loop, this is first message box'
Message("","var$")
var$='Hello from loop, this is second message box'
Message("","var$")
```

These lines will display two message boxes. One saying *Hello from loop, this is first message box*, and the second one saying *Hello from loop, this is second message box* . If these lines were executed without loop, message boxes would appear only once each. But ! Specifying loop using **For Counter=1 To 100**, we said to MMB: *repeat showing these message boxes hundred times each !* Every var$ would be assigned 100 times. Every message box would appear 100 times.

Once you're done writing commands that should be executed in loop, it's necessary to mark the end of such code block, otherwise MMB wouldn't know what's inside and outside of the for..next loop. And running entire program in one big loop surely ain't something you want.
Marking the end of for..next loop code block is done using **Next** clause. Remember that numerical variable **Counter**, used for setting number of loop repeats ? You'll write label of that variable behind the **Next** clause:

```
Next Counter
```

It's like saying to MMB: *once you're done performing lines in for..next block, update value of numerical variable used as a counter* (it goes either +1 or -1, more about it below) *and start performing block lines all over again !*

Here's colorized summary of for..next loop code lines:

```
For Counter= 1 To 100
```

```
   var$='Hello from loop, this is first message box'
    Message("","var$")
    var$='Hello from loop, this is second message box'
    Message("","var$")
Next Counter
```

Loop clauses **For, To, Next** are marked blue.
Numerical variable that serves as a loop counter is labeled **Counter** and is marked red.
Range for loop counter is marked green, in this case **0 - 100**.
All code lines that are executed repeatedly in loop are pink - **var$, Message("","")**

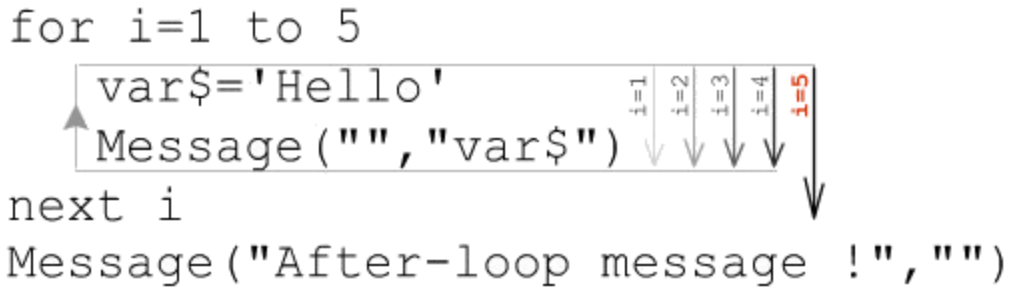
Loop counter

If you stand by the road and count cars passing by, you'll start from the first car and think of number "1" . When another car passes by, you'll add 1 to previous number and keep waiting for the next car. Until you get bored and say: *Hey, I had enough of this loop !*

Another case is a stopwatch in one hand, a gun in another hand and dozen of racers on track at some big stadium. You count down numbers...

10...9...8...7...6...5...4...3...2...1...BANG !

Both cases are being used in MMB for..next loops for counting purposes. Already mentioned **next** clause does exactly what people do when counting things - it either adds 1 or substracts 1 to/from some number (represented as numerical variable). That way, MMB's numerical variable "remembers" number of repeats it performed on code block inside loop and once it reaches ending value of range - loop is over, and BANG is heard (just kidding).



Sample above uses numerical variable **i** as a counter. Loop code block is performed 5 times, and after that, MMB continues executing code lines below For..Next loop block. It means that message box saying "After-loop message! !" will not be displayed before loop block is performed **n** times (in this case - 5).

Counting cases above mention both positive (+1) and negative (-1) counting. Examples of positive counting have been used before, but here's another one:

```
For Love=1 To 10
  var$='I love MMB'
```

```
  Message("","var$")
Next Love
```

That's when you start with lower value (**Love=1**) and end with higher value (**To 10**).

Counting in opposite (negative, -1) direction can also be used:

```
For Seconds=10 To 1
  var$='Woof !'
  Message("","var$")
Next Seconds
```

Here you start with higher value (**Seconds=10**) and end with lower value (**to 1**).

MMB compares starting and ending value, automatically deciding in what direction should it go. Of course, it's **not necessary** to use number 1 as either starting or ending point:

```
For MileAge=90 To 100
  var$='Almost there!'
  Message("","var$")
Next MileAge
```

This example uses 90 as a loop starting value and 100 as an ending value. So how many times will loop code be repeated ? Yep ! **10 !**

And it's not like you'll just use for..next loop to execute fixed code ! While MMB uses numerical variables for counters, and numerical variables can be used wherever you want in your code, current value of loop counter can be used for some dynamic actions.

It's like walking from one runner to another before the race, giving 'em t-shirt labels saying "You're number 1", "You're number 2", "You're number 23473"...

Let's take a look at loop code sample:

```
For Runner=1 To 10
  var$='You are runner no. '+CHAR(Runner)
  Message("","var$")
Next Runner
```

Loop above uses numerical variable **Runner** to execute some code lines 10 times. Assigning of text to string variable **var$** is interesting - it first sets text "You are runner no." and then uses CHAR function to convert current value of numerical variable **Runner** (used as a loop counter, remember ?) to act as a string, when **+** operator assigns it to **var$** .

The result ? Loop will show message box 10 times, but contents will be adjusted according to current value of loop counter. So, message box will every time read different numerical value:

You are runner no. 1
You are runner no. 2
You are runner no. 3

You are runner no. 4
You are runner no. 5
You are runner no. 6
You are runner no. 7
You are runner no. 8
You are runner no. 9
You are runner no. 10

This ability is very useful for intensive processing of items, names, records, files... you only change index number and everything else goes through the same block of commands inside the loop. A true time saver - imagine writing & maintaining the same block of commands for every item you want to process ! When you get into using loops, you will never throw 'em away ;)

TIP! There is also an option to dynamically change the **To** ending value from within the For..Next loop counter. Thanks to this you can prematurely terminate the For..Next loop in a certain conditions and without the need to call **Return**() or **Break**() function.

EXAMPLE OF USAGE:
The below script script is useless, but it can show you how to dynamically change the **To** value in **For...Next** loop. Initially, the loop should be repeated 5 times, but after the third loop, the loop range is changed to 3 and therefore the For..Next is terminated.

```
**This loop is terminated after 3 loops...
maxloop=5
For i=1 To maxloop
  Message("in loop","")
  If (i=3) Then
    ** maxloop is now 3!!
    maxloop=i
  End
Next i
Message("outside loop","")
```

Loop utilities

Once in a while it's possible to have a need for an endless loop that will, for example, check some things (system info, file existence, user input...) occasionally and without specific number of repeats. That's when **Infinite loops** come into action:

```
For check=1 To Infinity
  check=check+1
  DisplayValue("Text","check")
Next check
```

Code sample above uses **Infinity** clause as an ending value of numerical variable counter, thus making definition of loop saying "Start with 1 and never stop" . Of course, that *never* is hypothetical - loop is repeated while your program is running and project page is not changed. If either user exits program, changes page or computer shuts program down, loop will reach the very end of it's *infinity* ;)

Issue emerging from using this kind of loops is - program functioning while such loop is

being performed. Will everything be blocked while infinity loop keeps repeating ?

Not really, but to allow other objects & scripts to work properly it is recommended to use flow commands:

Refresh() - gives time to other threads being used by program to perform their actions
Pause("") - stops execution of code block for specified amount of time (in msecs)

Read more about these in command descriptions, here's how they work for loops:

```
For check=1 To Infinity
  check=check+1
  DisplayValue("Text","check")
  Refresh()
Next check
```

Put after loop code block, **Refresh()** gives time to other program threads before starting another repetition of code block.

```
For check=1 To Infinity
  check=check+1
  DisplayValue("Text","check")
  Pause("500")
Next check
```

Put after loop code block, **Pause("500")** makes delay (in this case 500 miliseconds - half a second) before repeating code block. It means that loop code will be executed every half a second. Even higher delay periods are recommended for tasks where real-time check-ups are not of a critical importance.

Another issue related to infinity loops in service of check-ups is a case when some condition has been fulfilled and loop should be stopped. In that case **Return()** command is a great help, while it stops further execution of entire script:

```
For check=1 To Infinity
  check=check+1
  DisplayValue("Text","check")
  If (UserInput$='GO') Then
    Return()
  End
  Pause("500")
Next check
```

In case you don't want to stop execution of entire script, but only jump from the current For..Next loop, use **Break()** command instead. The execution of code will continue after "***Next n"*** line of the For..Next loop where the Break() function was used.

Loop above is "smart" enough to perform repetition as long as value of string variable UserInput$ doesn't match fixed value "GO" . When UserInput$ contains correct value, if statement will call Return() command and loop (together with any code lines below the loop block) will be stopped. This routine is oftenly being used to call some other script with RunScript and ScriptTimer commands:

```
For check=1 To Infinity
  check=check+1
  DisplayValue("Text","check")
  If (UserInput$='GO') Then
    RunScript("InputCorrect")
    Return()
  End
  Pause("500")
Next check
```

Read more about **RunScript** and **ScriptTimer** in their command descriptions.

That's about everything to tell on the loop subject ! If you're a beginner in MMB scripting, loops probably won't be the first subject you'll get into, and instead you'll use manual code repetitions. But as you make progress in MMB scripting learning curve, loops will come up as a very comfortable solution for tasks with intensive scripting.

12.10 MMB Script Commands

12.10.1 Create and Delete Objects in a Runtime

Summary

With CreateObject functions you can create MMB objects directly from script! Previously, you had to design everything in designer and without an option to prepare some dynamic actions, like drawing lines, creating dynamic forms based on selected options, etc.

Sure, there was always an option to pre-made all needed objects, hide them from users and then show them at proper time. But it was not very useful solution, especially for games or heavily scripted projects. So here we are. From MMB 4.9.8 you can dynamically create most of basics MMB objects and even delete them (to save memory) using a DeleteObject function. To change the appearance of newly created objects use SetObjectParam function.

Here is the list of available commands, allowing you to create objects in a runtime:

- **Text Button** - CreateTextButton("inlabel","outlabel$,x,y,w,h,text")
- **Text Label** - CreateText("inlabel","outlabel$,x,y,text")
- **Paragraph** - CreateParagraph("inlabel","outlabel$,x,y,w,h,text")
- **Circle** - CreateCircle("inlabel","outlabel$,x,y,w,h,r,g,b")
- **Rectangle** - CreateRectangle("inlabel","outlabel$,x,y,w,h,r,g,b")
- **Line** (relative positioning) - CreateLine("inlabel","outlabel$,x,y,w,h,r,g,b")
- **Line** (absolute positioning) - CreateLineAB("inlabel","outlabel$,x1,y1,x2,y2,r,g,b")
- **HotSpot** - CreateHotSpot("inlabel","outlabel$,x,y,w,h")
- **Script object** - CreateScript("inlabel","outlabel$")

Where:

x is object's x position

y is object's y position

w is object's width

h is object's height

r,**g**,**b** is object's Red, Green, Blue color (in range 0..255)

text is object's text string

inlabel is a user-defined objet's name, let's say Text. If there already is an object with such name in the project, there will be added a number after the suggested name and this new name will be passed to outlabel$ variable (or whatever this variable will be called in your project). As you may know, you can't have two objects of the same name on the same page.

In short, if the object already exists (let's say **Text**), there will be used first available name based of suggested inlabel string (let's say **Text13)**. This name will be then passed to **outlabel$** variable and used by MMB. If you would like to delete or hide this object, you will have to use the **outlabel$** name in the appropriate function.

All parameters (except **outlabel$**) are IN parameters and can be defined as a numerical or string variable.

All parameters are optional, but it's recommended to define at least the text/position ;)

If there is no **inlabel** defined, the Create function uses default object names (like TextBTN, Circle,...). If there is no position/size defined, the object is created at 0,0 and with default (hardcoded) size.

You can skip some parameters, but you still have to use the appropriate number of comas, as a parameters separator. So let's say, you want to create Text Button with defined position and Text, but leave on MMB to use the default size. The code should appear like this..

CreateTextButton("inlabel","outlabel$,x,y,,,text")

CreateTextButton("inlabel","outlabel$,x,y,w,h,text")

Description

With this function you can create **Text Button** object. This will probably be the most used Create Object function. Available parameters:
in/out label
x,y - position
w,h - width/height
text - default button's text

To define other Text Button parameters (like BG color, text color, font, etc..) use SetObjectParam function right after CreateTextButton.

Code Examples

```
** this creates Text Button object called "BTN"
CreateTextButton("BTN","outlabel$,10,10,50,25,OK")


** or like this via script variables..
inlabel$='BTN'
x=10
y=10
w=50
h=25
text$='OK'
CreateTextButton("inlabel$","outlabel$,x,y,w,h,text$")
** and to change the button color use this..
```

```
rgbcol$='255,0,255'
SetObjectParam("outlabel$","BGCOLOR=rgbcol$")

** you can also use expressions in Create Object functions..
n=n+10
CreateTextButton("inlabel$","outlabel$,x,y,w+n,h+n,text$")
```

CreateText("inlabel","outlabel$,x,y,text")

Description

With this function you can create **Text** object. Available parameters:
in/out label
x,y - position
text - default text

To define other Text parameters (like text color, font, etc..) use SetObjectParam function right after CreateText function.

Code Examples

```
** this creates Text object called "TXT"..
  CreateText("TXT","outlabel$,10,10,Short Text")

** or like this via script variables..
inlabel$='TXT'
x=10
y=10
text$='Short Text'
CreateText("inlabel$","outlabel$,x,y,text$")
** and to change the text color use this..
rgbcol$='255,0,255'
SetObjectParam("outlabel$","TEXTCOLOR=rgbcol$")
```

CreateParagraph("inlabel","outlabel$,x,y,w,h,text")

Description

With this function you can create **Paragraph** object. Available parameters:
in/out label
x,y - position
w,h - width/height

text - default paragraph text

To define other Paragraph parameters (like text color, font, etc..) use SetObjectParam function right after CreateParagraph function.

Code Examples

```
** this creates Paragraph object called "PARA"..
CreateParagraph("PARA","outlabel$,10,10,320,200,Long text in Paragraph")

** or like this via script variables..
inlabel$='PARA'
x=10
y=10
w=320
h=200
text$='Lorem ipsum dolor sit amet et condimentum lectus sodales litora
parturient.'
CreateParagraph("inlabel$","outlabel$,x,y,w,h,text$")
** and to change the paragraph text color use this..
rgbcol$='255,0,255'
SetObjectParam("outlabel$","TEXTCOLOR=rgbcol$")
```

CreateCircle("inlabel","outlabel$,x,y,w,h,r,g,b")
CreateRectangle("inlabel","outlabel$,x,y,w,h,r,g,b")

Description

With these two functions you can create **Circle** or **Rectangle** object. Available parameters:
in/out label
x,y - position
w,h - width/height
r,g,b - default fill color

To define (or change) Circle/Rectangle parameters (like fill color, border type or color) use SetObjectParam function right after CreateCircle or CreateRectangle function.

Code Examples

```
** this creates Circle object called "CIRC"..
CreateCircle("CIRC","outlabel$,10,10,50,50,128,0,128")

** or like this via script variables..
inlabel$='CIRC'
```

```
x=10
y=10
w=50
h=50
** generate random color..
r=RND(254)+1
g=RND(254)+1
b=RND(254)+1
CreateCircle("inlabel$","outlabel$,x,y,w,h,r,g,b")
** and to change the circle fill color and border color use this code..
r=RND(254)+1
g=RND(254)+1
b=RND(254)+1
SetObjectParam("outlabel$","BORDERCOLOR=r,g,b")
```

CreateLine("inlabel","outlabel$,x,y,w,h,r,g,b")
CreateLineAB("inlabel","outlabel$,x1,y1,x2,y2,r,g,b")

Description

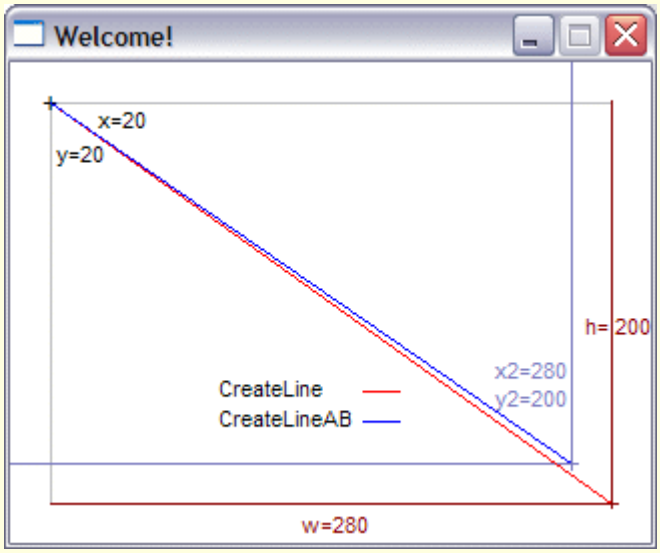
With **CreateLine** and **CreateLineAB** function you can create **Line** object. The difference between these two functions is in entering the position of the second (ending) point of the line.

Available parameters for **CreateLine**:
in/out label
x,y - position
w,h - width/height
r,g,b - default fill color

Available parameters for **CreateLineAB**:
in/out label
x1,y1 - position
x2,y2 - width/height
r,g,b - default fill color

While the Line created with **CreateLineAB** is defined by two points with absolute position on the project window, the line created with **CreateLine** is defined by the width/height of the line bounding box. In other words, both functions with exactly the same "position" values may produce different results, because of different meaning of the second pair of position values.

Please see the following image:



Here is the code used in the above screenshot:

```
CreateLine("LINE","outlabel$,20,20,280,200,255,0,0")
CreateLineAB("LINE","outlabel$,20,20,280,200,0,0,255")
```

As you may know, most of MMB objects are defined by x/y position of their left-top corner and width/height, which in fact represents the bottom-right corner of the object. This is also the case of **CreateLine** function. Although it may sound useless (to define the line with width/height) it's very useful in cases when you need to create multiple lines and you want to use relative positioning.

Code Examples

```
** this creates Line object called "LINE" and using the absolute position values..
CreateLineAB("LINE","outlabel$,20,20,280,200,0,0,255")

** this creates Line object called "LINE" but this time using the width/height values..
CreateLine("LINE","outlabel$,20,20,280,200,255,0,0")
```

CreateHotSpot("inlabel","outlabel$,x,y,w,h")

Description

With **CreateHotSpot** function you can create hotspot object. It's useful for creating clickable active areas in your project. But as you may (or may not?;) know, the HotSpot can also be used as an image container! You can load an image into the HotSpot using the **ReplaceImage** function. And then the HotSpot object can be used and manipulated as Bitmap object (using the available Bitmap related commands).

Available parameters for **CreateHotSpot**:
in/out label
x,y - position
w,h - width/height

Code Examples

```
In this example you can learn how fill the project window with hotspots
containing images and how to calculate the maximum number of images for
project width/height.

pw=PubWidth()
ph=PubHeight()
** size of space between the images
coef=10
w=50
h=50
** count the number of images for current width..
n_imgs_w=pw/(w+coef)
RoundNum1=INT(n_imgs_w)
RoundNum2=n_imgs_w-RoundNum1
If (RoundNum2>=0.5) Then
  n_imgs_w=RoundNum1 + 1
End
** and the number of images (image columns) is..
n_imgs_w=n_imgs_w-1
** n of images for current height
n_imgs_h=ph/(h+coef)
RoundNum1=INT(n_imgs_h)
RoundNum2=n_imgs_h-RoundNum1
If (RoundNum2>=0.5) Then
  n_imgs_h=RoundNum1 + 1
End
** and the number of images (image rows) is..
n_imgs_h=n_imgs_h-1
********************
x=coef
y=coef
For j=1 To n_imgs_h
  For i=1 To n_imgs_w
    Pause("50")
    CreateHotSpot("HSpot","object[i]$,x,y,w,h")
```

```
    ** set new x (column) postition
    x=x+w+coef
    ** set script to HotSpot
    script$='CurObj$=CurrentObject()'+CHR(13)+CHR(10)
    script$= script$ + 'Message("Selected Object","CurObj$")' +CHR(13)+
CHR(10)
    SetObjectParam("object[i]$","MOUSEUPSCRIPT:1=script$")
    ** set repeating of images. If you for example have only 3 images in
the folder
    ** and the number of added hotspots is higher, you will have to
repeat the images.
    ** In this sample we have only 3 images in Images folder, so if the
actual image count is
    ** higher or equal 3, reset the image count..
    If (ni>=3) Then
      ni=1
    Else
      ** else add 1 to the image count..
      ni=ni+1
    End
    imgpath$='<SrcDir>\Images\\'+CHAR(ni)+'.jpg'
    ** set image to hotspot object
    ReplaceImage("object[i]$","imgpath$")
    ** move object to back (change object Z-order)
    ReorderObject("object[i]$","BACK")
    Refresh("")
  Next i
  ** reset x (column) position
  x=coef
  ** new y (line) position
  y=y+h+coef
Next j
```

CreateScript("inlabel","outlabel$")

Description	

With this function you can create **Script** object. Available parameters:
in/out label

The Script object is the small yellow rectangle holding the scripts, which can be called via RunScript or ScriptTimer. It's some kind of procedure, which can be called from another, processed (via RunScript) or run simultaneously (using ScriptTimer) with another part of your project. Usually, Script objects are created at a design time. But thanks to CreateScript you can now create new Script object also in a runtime and fill it with script using the SetObjectParam, which now allows (from 4.9.8) to replace the actual object's script with new content. The script can be generated via script, loaded from predefined file, or simply typed in EditBox ;)

Code Examples

```
** this creates Script object called "SCRPT"..
CreateScript("SCRPT","outlabel$")
** set new script..
ScriptParam$='SCRIPT:1=' + 'Message("Hello World","")'
SetObjectParam("outlabel$","ScriptParam$")
** this runs the newly created script object
RunScript("outlabel$")

** create new Script object, this time with multiline script..
CreateScript("SCRPT","outlabel$")
** set new script..
script$='SCRIPT:1=' + 'For i=1 To 10'+CHR(13)+CHR(10)
script$= script$ + 'loop$=CHAR(i)'+CHR(13)+CHR(10)
script$= script$ + 'loop$=CHAR(i)'+CHR(13)+CHR(10)
script$= script$ + 'Message("Loop:","loop$")'+CHR(13)+CHR(10)
script$= script$ + 'Next i'+CHR(13)+CHR(10)
SetObjectParam("outlabel$","script$")
** this runs the newly created script object
RunScript("outlabel$")
```

DeleteObject("ObjectLabel")

Description

DeleteObject("ObjectLabel")

With this function you can delete both objects created in a designer or dynamically in a runtime. All you need to do is to apply the correct (existing) object label into the function parameter. Be careful with using this function! Once you delete the object, the only way how to get it back is to restart the project or create the object with runtime object creation functions.

Code Examples

```
** this deletes the "TextBTN"
DeleteObject("TextBTN")


** this deletes all text buttons from TextBTN1 to TextBTN10
For i=1 To 10
  DeleteObject("TextBTN[i]")
Next i
```

12.10.2 Global Object and Project Functions

Summary

SetObjectParam("ObjectLabel, "Parameters")
this command will allow you to set/change some of common aspects of basic MMB objects in a runtime.

SetProjectParam("ProjectParameterName", "Parameters")
this command will allow you to set/change some of the project parameters in a runtime.

Both commands along with their parameters (and other 4.9.7 related commands), can be seen in action in 497_test_project.mbd samples project.

SetObjectParam("ObjectLabel, "Parameters")

Description

SetObjectParam("ObjectLabel, "Parameters")
Usually, you may set object attributes in a design time via object properties panel. But with this magic function you can set various aspects of some MMB objects in a runtime.

Here is the list of available parameters including their applicability to MMB objects...

Parameter	Value	Description|Usage	Applicability
MOUSEDOWNSCRIPT= MOUSEUPSCRIPT= MOUSEDOWNSCRIPT:0 or :1 = MOUSEUPSCRIPT:0 or :1=	script code/ variable $	Sets the object's Mouse UP/DOWN script. The subparam :0 or :1 is for quiet/loud syntax parsing. In case of some script errors (e.g. wrong script syntax) and subparam :1 you will be notified about incorrect script via error message box. **Be careful about using quiet parsing!** With quite parsing you can easily miss some syntax errors!	Circle HotSpot, Rectangle Text Text Button Bitmap Bitmap Button
SCRIPT= SCRIPT:0 or :1=	script code/ variable $	Sets the Script object's script. As in case of above MOUSEDOWNSCRIPT/ MOUSEUPSCRIPT, the subparam :0 or :1 is for quiet/loud syntax parsing.	Script
BG=	TRUE/ FALSE	Enables/disables EditBox background. SetObjectParam("object","BG=TRUE")	EditBox
BGCOLOR=	R,G,B	Sets the object background color. The	EditBox

		letters R, G and B stands for Red, Green and Blue integer values in range 0..255. Instead of separated R,G,B values can be used also a string variable containing that RGB comma separated values or **CBK_SelColor** returned from **ColorPicker** dialog.	Primitive TextBTN ListBox Line
BORDERCOLOR=	R,G,B	Sets the border color of supported objects. The R,G,B parameter settings are the same as in above BGCOLOR. NOTE: The border color can be changed only for LINE border.	EditBox Primitive
BORDERTYPE=	LINE WINDOWS SUNKEN NONE	Sets the border type of supported objects. The currently supported parameters are the same as used in design-time mode and similar to Windows border types. LINE - just one-pixel line around the object WINDOWS - raised border around the object SUNKEN - sunked border around the object NONE - surprisingly, no border around the object ;)	EditBox Primitive
TEXTCOLOR=	R,G,B	Sets the text color of supported objects. The R,G,B parameter settings are the same as in above BGCOLOR and BORDERCOLOR.	EditBox Text Paragraph Button ListBox
FONTNAME=	fontname string	Sets the font in supported objects with Text label. The fontname value is just the name of font, but not the filename as you may think! The name of font is just the name under which is the font registered in windows. It's the same name as you may get from the font properties (by double-clicking the font file) or which you can see in Font selection dialog.	EditBox Text Paragraph Button ListBox
FONTSTYLE=	REGULAR ITALIC BOLD BOLDITA	Sets the font style in supported objects with Text label.	EditBox Text Paragraph Button

<table>
<tr><td></td><td>LIC</td><td></td><td>ListBox</td></tr>
<tr><td>FONTSIZE=</td><td>number</td><td>Sets the font size in supported objects with Text label.</td><td>EditBox
Text
Paragraph
Button
ListBox</td></tr>
<tr><td>FONTEFFECT=</td><td>NONE
STRIKEOUT
UNDERLINE
STRIKEUNDER</td><td>Sets the font effect in supported objects with Text label.</td><td>EditBox
Text
Paragraph
Button
ListBox</td></tr>
<tr><td>FONTSCRIPT=</td><td>number</td><td>Sets the font script (national character set) in supported objects with Text label.

The script value can be any number from the below table..

<table><tr><th>script num.</th><th>description</th></tr><tr><td>0</td><td>western</td></tr><tr><td>1</td><td>other</td></tr><tr><td>2</td><td>symbol</td></tr><tr><td>77</td><td>MAC</td></tr><tr><td>128</td><td>japanese</td></tr><tr><td>129</td><td>korean (Wansung)</td></tr><tr><td>130</td><td>korean (Johab)</td></tr><tr><td>134</td><td>simplified chinese</td></tr><tr><td>136</td><td>traditional chinese</td></tr><tr><td>161</td><td>greek</td></tr><tr><td>162</td><td>turkish</td></tr><tr><td>163</td><td>vietnamese</td></tr><tr><td>177</td><td>hebrew</td></tr><tr><td>178</td><td>arabic</td></tr><tr><td>186</td><td>baltic</td></tr><tr><td>204</td><td>cyrillic</td></tr><tr><td>222</td><td>thai</td></tr><tr><td>238</td><td>central european</td></tr></table></td><td>EditBox
Text
Paragraph
Button
ListBox</td></tr>
</table>

FONTALL=	CBK_FONT or String array	This "font" parameter simply sets all font parameters in a single line of code. The parameter can be either CBK_FONT obtained from **FontPicker** command or a string array in this format.. FONTNAME|FONTSTYLE|FONTSIZE|FONTSCRIPT|FONTEFFECT|	EditBox Text Paragraph Button ListBox

Code Examples

```
** this adds (replaces the actual object's script with..) "Message" function to "TextBTN" object (with loud script parsing)
SetObjectParam("TextBTN","MOUSEUPSCRIPT:1=Message("Hello there!","")")

** this adds (replaces the actual object's script with..) "Message" function, this time with using the string variable (& quite script parsing)
code$='Message("","a")'
SetObjectParam("Script","MOUSEDOWNSCRIPT:0=code$")

** this sets the object BG color to Red
SetObjectParam("object","BGCOLOR=255,0,0")

** this sets the object BG with color obtained from ColorPicker dialog
ColorPicker()
SetObjectParam("object","BGCOLOR=CBK_SelColor")

** An example of SetObjectParam with FONTALL param:
SetObjectParam("object","FONTALL=arial|regular|12|0|none|")
** or like this...
params$='arial|regular|12|0|none'
SetObjectParam("object","FONTALL=params$")
** or this...
SetObjectParam("object","FONTALL=CBK_FONT")
```

SetProjectParam("ProjectParameterName, "Parameters")

Description

SetProjectParam("ProjectParameterName, "Parameters")
This function sets some aspects (for now, mainly the Page attributes) of MMB project.

Here is the list of available parameters...

1st Parameter	2nd Parameter	Value	Description|Usage
PAGEBG	IMAGE=	path$	Sets an image on the Page BG. If the "IMAGE=" is an empty string, the Page BG image is removed.
PAGEBG	DISPLAY=	TILE NORMAL	Sets the Page BG image tiling. At the moment, only TILE and NORMAL parameters..sorry, no STRETCH yet :)
PAGEBG	COLOR=	R,G,B	Sets the Page BG color. The same syntax as in above SetObjectParam.
PAGEBG	FROMMASTER=	TRUE/ FALSE	Enable/Disable obtaining the Page BG settings from Master Page.
MASTERPAGEBG	IMAGE=	path$	Sets an image on the MasterPage BG. If the "IMAGE=" is an empty string, the MasterPage BG image is removed.
MASTERPAGEBG	DISPLAY=	TILE NORMAL	Sets the MasterPage BG image tiling. At the moment, only TILE and NORMAL parameters..sorry, no STRETCH yet :)
MASTERPAGEBG	COLOR=	R,G,B	Set the MasterPage BG color. The same syntax as in above SetObjectParam.
FULLSCREENBG	IMAGE=	path$	Sets an image on the FullScreen BG. If the "IMAGE=" is an empty string, the FullScreen BG image is removed.
FULLSCREENBG	DISPLAY=	TILE STRETCH NORMAL	Sets the FullScreen BG image tiling.
FULLSCREENBG	COLOR=	R,G,B	Set the FullScreen BG color. The same syntax as in above SetObjectParam.

Code Examples

```
** this sets the Page BG image
SetProjectParam("PAGEBG","IMAGE=<SrcDir>\images\image.jpg")

** this stretch the image loaded on FullScreen BG
```

```
SetProjectParam("FULLSCREENBG","DISPLAY=STRETCH")

** this sets the Master Page color
ColorPicker()
SetProjectParam("MASTERPAGEBG","COLOR=CBK_SelColor")
```

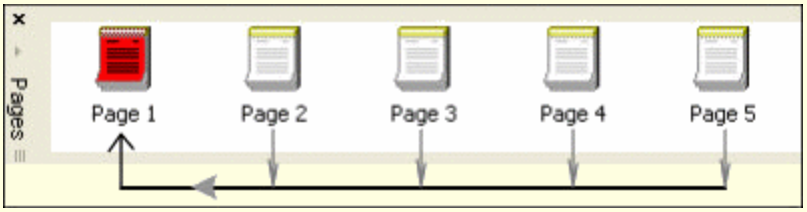
12.10.3 Project Commands

The purpose of project commands is handling of application pages, their appearance, project loading and exiting. Pages in MMB are displayed at the editor's bottom toolbar:

FirstPage()

Description

Moves to first page of project.



If there's page startup script, it will be executed.

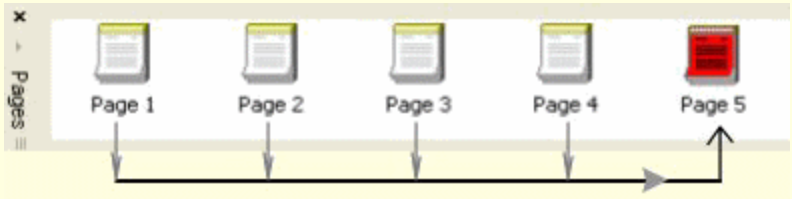
Code Example

```
FirstPage()
```

Additional Info

If you're using visible PlugIns, it is recommended to use their (PlugIn-specific) commands for hiding before changing project page.

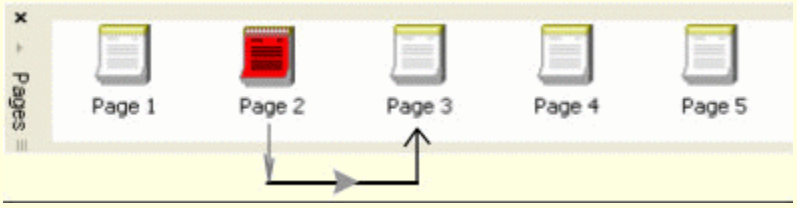
LastPage()

Description
Moves to last page of project.    If there's page startup script, it will be executed.

Code Example
LastPage()

Additional Info
If you're using visible PlugIns, it is recommended to use their (PlugIn-specific) commands for hiding before changing project page.

NextPage()

Description
Moves to next page of project.    If there's page startup script, it will be executed.

Code Example
NextPage()

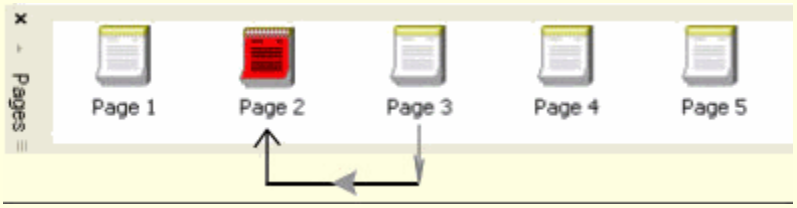
Additional Info

If you're using visible PlugIns, it is recommended to use their (PlugIn-specific) commands for hiding before changing project page.

PrevPage()

Description

Moves to previous page of project.



If there's page startup script, it will be executed.

Code Example

```
PrevPage()
```

Additional Info

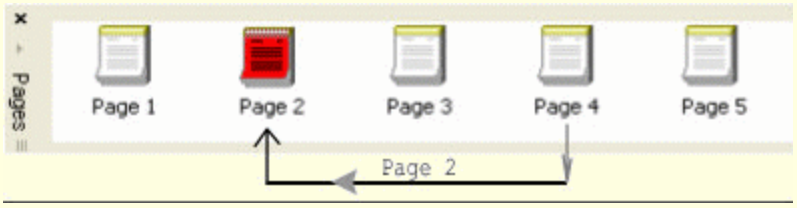
If you're using visible PlugIns, it is recommended to use their (PlugIn-specific) commands for hiding before changing project page.

Page("Page")

Description

Moves to page set by label parameter inside parenthesis and quotes.

Using LASTPAGE as parameter will load last visited page.



If there's page startup script, it will be executed.

Code Examples

```
Page("Page 4")
**(loads page labeled Page 4)

Page("LASTPAGE")
**(loads last visited page)
```

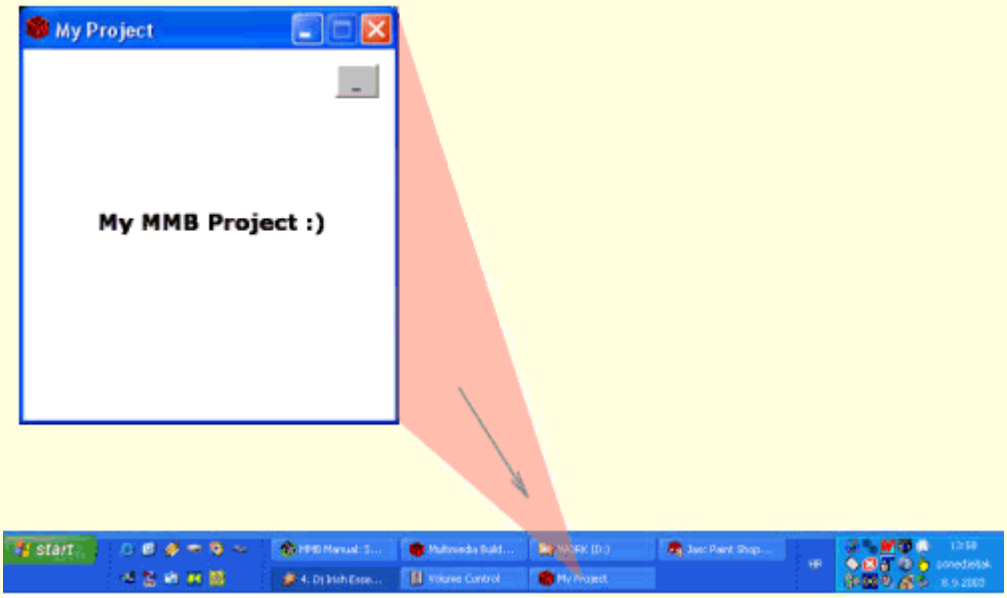
Additional Info

If you're using visible PlugIns, it is recommended to use their (PlugIn-specific) commands for hiding before changing project page.

Minimize()

Description

Minimizes project window to Windows TaskBar.



Clicking associated taskbar button will restore project window.

Code Examples	

```
Minimize()
```

Additional Info	

Minimizing to tray icon is available through PlugIns like SmallPlugin, MMBMisc or Tweak.

Maximize()

Description	

Maximizes project window on Desktop. This function is available only for standard project window with title bar!

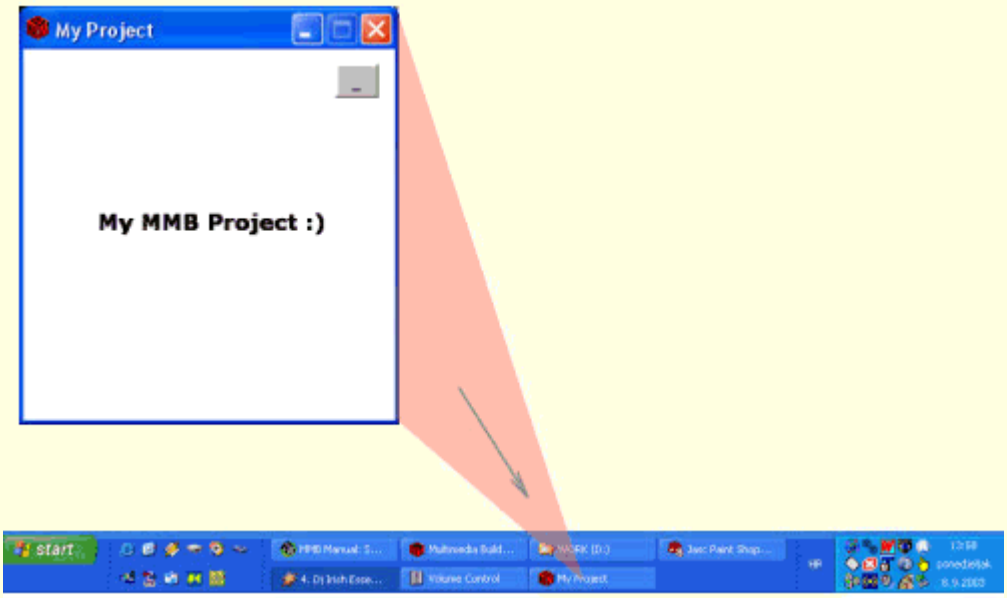
Code Examples

```
Maximize()
```

Restore()

Description

Restores project window Minimized to Windows TaskBar or Maximized to its full size.



Code Example

```
Restore()
```

Run("Path","CmdLineParams")

Description	

Using this command, you can run any program (.exe), batch file (.bat) or call file with any registered extension that can be viewed by some external program. For example, HTML files will be started in an associated web browser (typically IE) or for example MP3 files will be started in associated audio player (e.g. WinAmp).

Command has one required and one optional parameter:
`Run("FolderPath-FileName","Parameters-Arguments")`

1) program folder path & file name is required

2) Run command parameters or program arguments are optional

Program folder path & file name - either relative (using **file path macros**) or fixed path with name of file you want to run.

Run command parameters - handle behaviour of executed file and your project when that file is run. 6 comma separated (but without an empty space!) parameters are available:

TOP - executed program will always start on top of the source application (your project), but clicking on another window will move program's window to the background (behind selected window). Setting delay value (in milliseconds) inside parenthesis will pause putting program's window on top. For example: *TOP(2000)* will wait 2 seconds before setting top position for program's window.

TOPMOST - executed program will be kept on top of all windows (until some other application takes TOPMOST position).
Setting delay value (in milliseconds) inside parenthesis will pause putting program's window on top of all others. For example: *TOPMOST(2000)* will wait 2 seconds before setting top position for program's window.

WARNING!If you run any application with TOP/TOPMOST flags, MMB always wait until the application initialize (an infinite time). It can be potentially dangerous in cases when the external application cannot start (from any reason), because it can freeze the MMB completely. Therefore, there is an option to set a timeout value to continue with MMB application after a number of milliseconds, no matter if the external application is already initialized or not.

Using the timeout value is required if you use TOP/TOPMOST flags on some nonstandard applications without the "message queue" (for example command.com, cmd.exe or just the bat files, which in fact runs command.com/cmd.exe system apps).

WAIT - your MMB project will wait for run application to finish (close) before any further code line is performed. Useful for calling external encoders, compressors, encryptors and similar programs that require unpredictable amount of time for data processing. Using WAIT parameter, your project won't go out of synchronization with external program.

MAXIMIZE - Activates application's window and displays it as a maximized window.

MINIMIZE - Displays application's window in a minimized mode (i.e application is minimized in

Windows taskbar). The project window (MMBuilder) remains active, but only without WAIT parameter.

HIDE - Hides application's window and activates another (main project) window. This parameter is useful if you don't want to display the application window even in minimized mode (for example if you run the bat files). This will start the application with completely hidden main window.

WARNING!!! Be careful with using HIDE parameter, mainly together with WAIT parameter! If application with HIDE and WAIT parameters will not be automatically finished (i.e. without user's input) it will freeze the MMB application! For example, don't use "Pause" command in your bat files if you want to process these files with HIDE&WAIT parameters.

You can use all the above parameters together, separated by comma (without an empty space), but some combinations are useless (e.g. HIDE/MINIMIZE&TOP/TOPMOST):
Run("<SrcDir>\MyApp.exe","TOP(2000),WAIT")

Arguments - various parameters specific to program you're running. For example, running of Notepad can be argumented with name of file you want to run, making it automatically opened upon Notepad run:

Run("<Windows>\notepad.exe","<Windows>\win.ini")

If you want to run Notepad with a text file as a parameter, and with TOPMOST and WAIT, you will have to use this notation...

Run("<Windows>\notepad.exe","WAIT,TOPMOST <Windows>\win.ini")

In other words, first empty space used in second Run command parameter is used as a separator of MMB internal parameters (like WAIT, TOPMOST, etc.) and the program arguments (in this case path to **<Windows>\win.ini**).

Code Examples

```
** Running Notepad without parameters/arguments:
Run("<Windows>\notepad.exe","")

** Running Notepad with file name as argument:
Run("<Windows>\notepad.exe","<SrcDir>\MyLog.txt")

** Running Notepad using TOP parameter and 2-second delay:
Run("<Windows>\notepad.exe","TOP(2000)")

** Running Calculator on top of all windows:
Run("<System>\calc.exe","TOPMOST")

** Running BladeEnc on top with waiting & sound files in <SrcDir> folder:
Run("<SrcDir>\BladeEnc.exe","TOP,WAIT -128 file.wav file.mp3")

** Running BAT file in HIDE and WAIT mode :
Run("<Embedded>\test.bat","HIDE,WAIT param1 param2")
```

```
An advanced example of using WAIT and HIDE with Windows command prompt...
** on WinNT/XP
If (UsingWinNT()) Then
   ** this will list all directories from a given drive and save the obtained list to txt file
  Run("<System>\cmd.exe","WAIT,HIDE /C dir d:\ /A:D /B /O:N /s > c:\outputfile.txt")
** on Win9x/ME
Else
  Run("c:\command.com","WAIT,HIDE /C dir d:\ /A:D /B /O:N /s > c:\outputfile.txt")
End
** and finally, load the saved list to ListBox
ListBoxAddItem("ListBox","c:\outputfile.txt")
```

Additional Info

It is recommended to, whenever possible, run programs/files using relative paths. Read more about **path macros here**.

This command sends return values from the running application (if return codes were implemented in application) to **CBK_ReturnVal** constant.

Known Limitations

- TOP/TOPMOST parameters don't support child windows (only the first (main) application window can be set as TOP/TOPMOST).
- The applications without PID (Process ID) number may not work with TOP/TOPMOST flags.
- The external application will not be automatically closed if you close the source MMB application
- Timout value is required for TOP/TOPMOST parameters used for console applications (command com, cmd.exe, bat files) or any nonstandard applications without the "message queue". Otherwise TOP/TOPMOST parameter will freeze MMB application.

RunMBD("Path","Parameters")

Description

Runs another MMB project file with MBD extension.

To avoid large distributions of numerous compiled EXE files, using MMB you can compile only one project file and then run all others using that same compiled EXE. This is possible because project file is only attached to interpreter that represents your compiled EXE project. If it can run attached MBD, why not make it possible to run external ones too ?

This is very neat feature that reduces requirements for both RAM and distribution memory capacity, while MBD files are much smaller than executable files.

Using Compress & Export dialogbox:

...every MBD file can be:

- Compressed - using one of 8 compression levels
- Password-protected - editing of files in MMB editor will require password input
- Encrypted using Secure Layer - attended to project distribution purposes, encrypted files won't be editable again

Thus you don't have to worry about distributing MBD project files.

Command has one required and one optional parameter:
RunMBD("FolderPath-FileName","Page Label")

1) folder path & MBD file name - required

2) Label of displayed page in loaded MBD - optional

Program folder path & file name - either relative (using **file path macros**) or fixed path with name of MBD file you want to run

Parameters:

NEW_WINDOW - (optional parameter) starts MBD project in a new window.

Page Label - (optional parameter) once MBD file is loaded, MMB can automatically display the page specified using the Page Label parameter.

Code Examples

```
** Running Program2.mbd from <SrcDir> without page label:
RunMBD("<SrcDir>\Program2.mbd","")

** Running Program2.mbd from <SrcDir> with page label:
RunMBD("<SrcDir>\Program2.mbd","Page 2")

** Running Program2.mbd from <SrcDir> with page label and in new window:
RunMBD("<SrcDir>\Program2.mbd","NEW_WINDOW,Page 2")
```

Additional Info

NEW_WINDOW can be used together with **Page Label** parameter (separated by comma) or individually.

NEW_WINDOW must be used in front of PageLabel parameter

Page Label now can be used also with "Run another project" trigger action in "External Commands and Page Actions".
It is recommended to, whenever possible, run programs/project files using relative paths. Read more about **path macros here**.

Exit()

Description

Closes application.

If your project allows user input that becomes permanent part of application, make

sure to save inputed data before exiting.

Code Example	

```
Exit()
```

Additional Info	

Closing of application is by default also available using ESC (Escape) key on the keyboard. To disable this, use CBK_EXIT object on Master Top Layer. Read about **CBK constants** for more info.

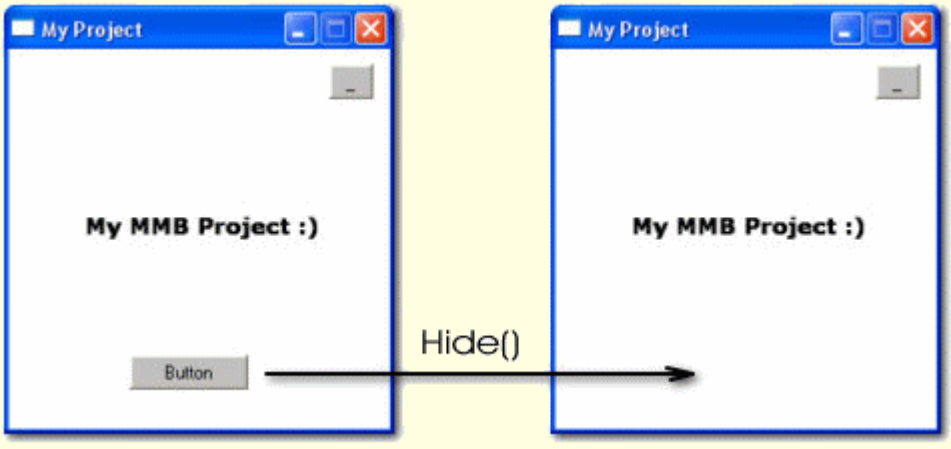
12.10.4 Object Commands

With object-related commands you can handle display status and position of objects in your project. There are also two object-related commands for text loading/changing.

Hide("ObjectLabel")

Description

Hides object specified using label as a command parameter.



Both individual objects and object groups can be hidden using this command.

Object or Group label that should be hidden is specified as parameter inside parenthesis and double quotes.

Code Examples

```
** Hide object labeled "Button" :
Hide("Button")

** Hide group of objects labeled "GroupOfButtons" :
Hide("GroupOfButtons")
```

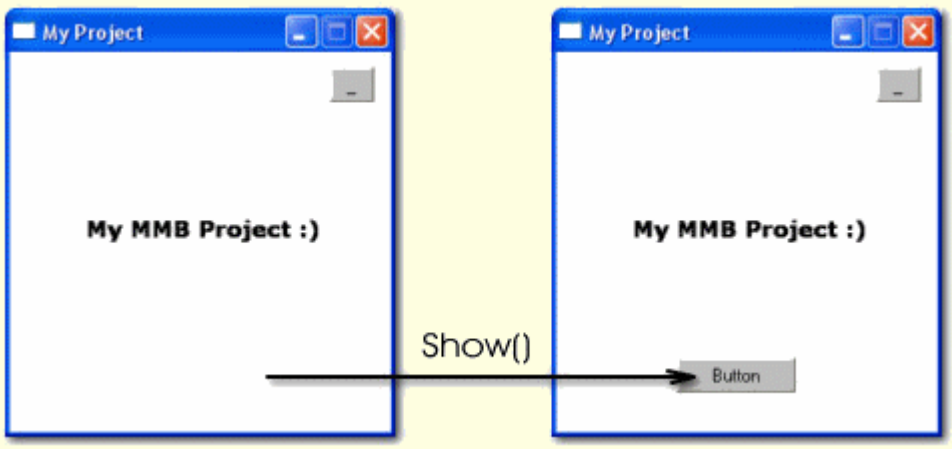
Additional Info

Visible PlugIns cannot be hidden using this command. To hide visual PlugIns, use their (PlugIn-specific) commands for window hiding.

Show("ObjectLabel")

Description

Shows object specified using label as a command parameter.



Both individual objects and object groups can be displayed using this command. Object or Group label that should be displayed is specified as parameter inside parenthesis and double quotes.

Code Examples

```
** Show object labeled "Button" :
Show("Button")

** Show group of objects labeled "GroupOfButtons" :
Show("GroupOfButtons")
```

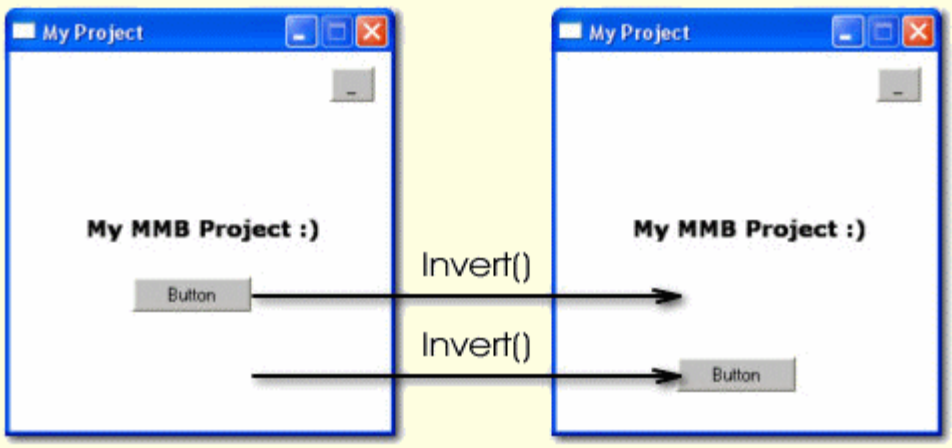
Additional Info

If hidden, visible PlugIns cannot be displayed using this command, so use their (PlugIn-specific) commands for window hiding.

Invert("ObjectLabel")

Description

Inverts display status of object specified using label as a command parameter. If object was hidden, it will be displayed. If object was displayed, it'll be hided.



Both individual objects and object groups can be inverted using this command. Object or Group label that should be inverted is specified as parameter inside parenthesis and double quotes.

Code Examples

```
** Invert object labeled "Button" :
Invert("Button")

** Invert group of objects labeled "GroupOfButtons" :
Invert("GroupOfButtons"
```

Additional Info

Display status of visible PlugIns cannot be inverted using this command, so use their (PlugIn-specific) commands for display status.

MoveObject("ObjectLabel","x,y[,w,h]")

Description	

Moves and resizes object specified using label as a first command parameter to position and size set through coordinates & size values as second command parameter.

```
MoveObject("ObjectLabel","x,y,w,h")
```

Second parameter consits of 4 values:

x - horizontal position of object in pixels (required)
y - vertical position of object in pixels (required)
w - width of object in pixels (optional)
h - width of object in pixels (optional)

Both individual objects and object groups can be moved using this command. Object or Group label that should be moved is specified as parameter inside parenthesis and double quotes. Only individual objects can be resized using MoveObject command.

Second parameter can either have only size values:

```
MoveObject("Bitmap","139,130")
```

...or both size and dimension values:

```
MoveObject("Bitmap","139,130,97,95")
```

Code Examples	

```
** Move object labeled "Bitmap" without resizing:
MoveObject("Bitmap","201,325")

** Move object labeled "Bitmap" with resizing:
MoveObject("Bitmap","201,325,38,11")

**Move group of objects labeled "GroupOfBitmaps" :
MoveObject("GroupOfBitmaps","150,50")
** Move object labeled "Bitmap" with resizing using numerical variables:
MoveObject("Bitmap","xpos,ypos,width,height")
```

Additional Info

Combining MoveObject command with **MouseX() and MouseY() constants** and **for..next loops** can give interesting results - objects can be moved real-time, as user moves mouse pointer.

MoveTo("ObjectLabel","Parameters")

Description

Moves object specified using label as a first command parameter to position set through coordinates and movement animation properties as second command parameter.

```
MoveTo("ObjectLabel","x,y,Steps,AnimType")
```

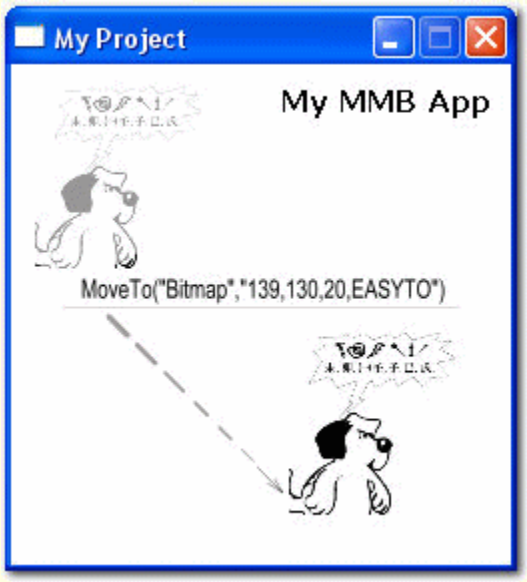
Second parameter consits of 4 values:

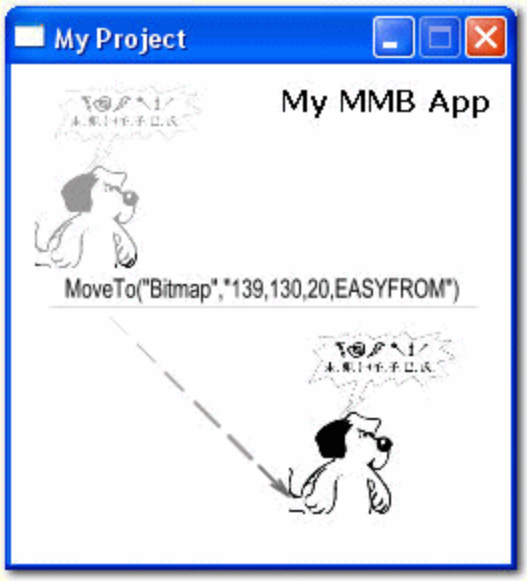
x - horizontal position of object in pixels (required)
y - vertical position of object in pixels (required)

Steps - number of steps that should be used for movement animation - lower value makes movement faster, higher value makes animation smoother and slower)

AnimType - sets type of movement animation. Available types are:

- EASYTO (starts with faster movement and slows down as it reaches the destination coordinates),
- EASYFROM (starts with slower movement and speeds up as it reaches destination coordinates)





Both individual objects and object groups can be moved using this command. Object or Group label that should be moved is specified as parameter inside parenthesis and double quotes.

Code Examples	

```
** Move object labeled "Bitmap" with 40 steps (slow) and
**slower animation on start of movement:
MoveTo("Bitmap","201,325,40,EASYFROM")

** Move object labeled "Bitmap" with 10 steps (fast) and
**slower animation on end of movement:
MoveTo("Bitmap","201,325,40,EASYTO")

** Move object labeled "Bitmap" using numerical variables and
**slower animation on end of movement:
MoveTo("Bitmap","xpos,ypos,steps,EASYTO")

** Move group of objects labeled "BitmapGroup" with 20 steps (medium) **and
slower animation on start of movement:
MoveTo("BitmapGroup","201,325,20,EASYFROM")
```

Additional Info

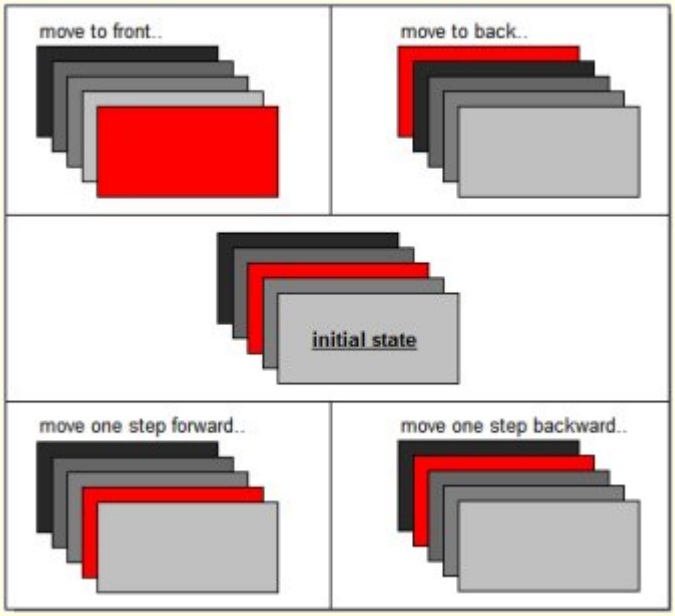
Combining MoveTo command with **MouseX() and MouseY() constants** and **for.. next loops** can give interesting results - objects can be moved real-time, as user moves mouse pointer.

ReorderObject("ObjectLabel","Parameters")

Description

Changes the current position of a given object in Z-axis. The applicable parameters are:

FRONT - move object to Front
BACK - move object to Back
FORWARD - move object one step up
BACKWARD - move object one step back



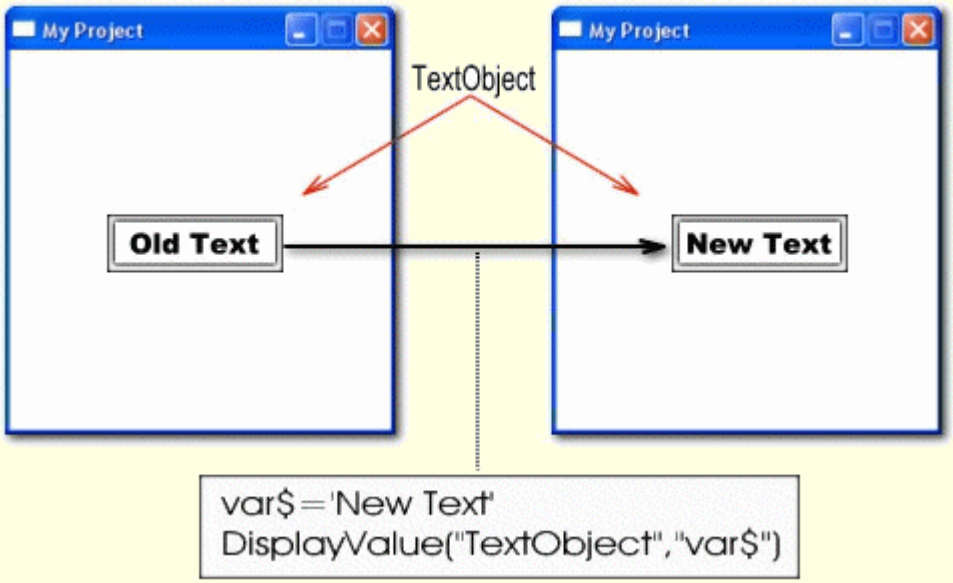
Code Examples

```
**Set object on top of other objects..
ReorderObject("ObjectLabel","FRONT")

**Move object one step back in Z-Axis..
ReorderObject("ObjectLabel","BACKWARD")
```

DisplayValue("ObjectLabel","Value")

Description

Loads content of string or numerical variable into either text object or button object. First parameter sets object label where content of the variable will be displayed. Second parameter sets source variable where content comes from.



Code Examples

```
** Displaying content of string variable in text object NameLabel:
name$='Little Joe'
DisplayValue("NameLabel","name$")

** Displaying content of numerical variable in text object AgeLabel:
age=47
DisplayValue("AgeLabel","age")

** Displaying content of string variable in button object TextBTN:
Item$='File'
DisplayValue("TextBTN","Item$")
```

Additional Info

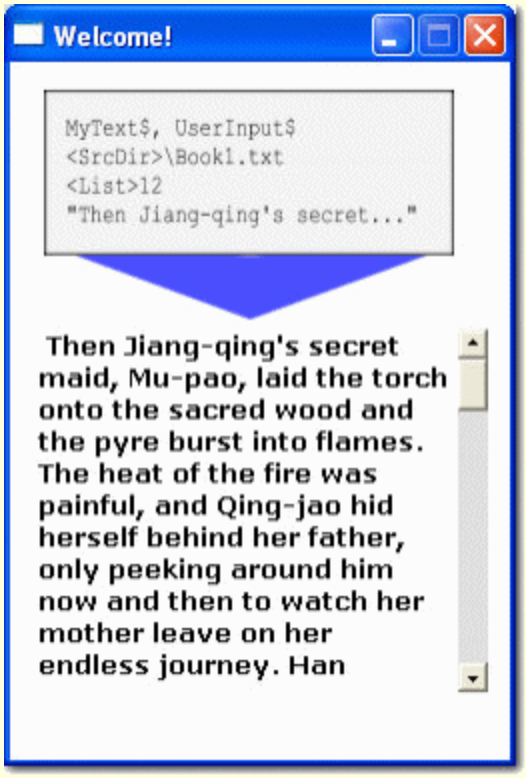
This command should be used for short strings only. To load text paragraphs, use LoadText command.

LoadText("ObjectLabel","String")

Description

Loads content of string variable, external file or text set as second parameter into any text, button, editbox or paragraph object.

LoadText("ObjectName","TextSource")



First parameter sets object label where content will be displayed. Second parameter sets one of multiple text sources:

- string variable (txt$, var$)
- text file (<SrcDir>\MyStory.txt)
- Song List (<List>5)
- fixed text as second parameter

Code Examples

```
** Loading text to Paragraph from text$ string variable:
LoadText("Paragraph","text$")

** Loading text to Paragraph from text file story.txt in <SrcDir>:
LoadText("Paragraph","<SrcDir>\story.txt")

** Loading text to Paragraph from MMB's Song List, item 7 :
LoadText("Paragraph","<List>7")

** Loading text to Paragraph from fixed text in second parameter:
LoadText("Paragraph","Hello MMB")

** Loading text to EditBox from input$ string variable:
LoadText("EditBox","input$")
```

Additional Info

MMB recognizes folder path & file name in second parameter as a source file for text that should be loaded. Sometimes all you want to do is display a path in the text object, not load it's content. To control LoadText behaviour in such case, use these parameters in front of the variable:

STRING - instructs command to handle source contents as a string:

```
LoadText("Paragraph","STRING:File$")
```

FILE - instructs command to handle source contents as a file:

```
LoadText("Paragraph","FILE:File$")
```

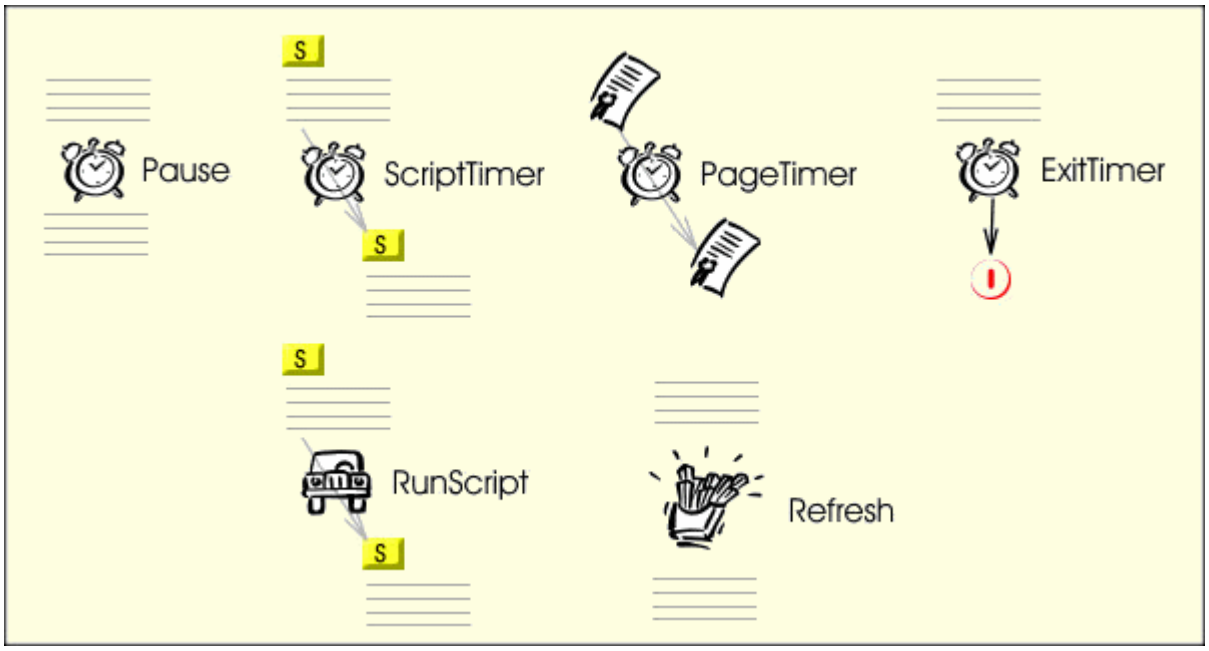
Examples above use File$ variable that contains a file path.
In the first case, path will only be displayed in destination text object.
In the second case, path will be used as a source file for text.

12.10.5 Time & Script Run Commands

There are cases when executed code requires delay before continuing to the next step (line of code). Either it's about moving to another page, delaying execution of current script or another script, time & script run commands serve as creators of pauses / delays.

Strange thing to do, indeed.

In reality we try to avoid delays and here we make artificial ones.



You'll use commands like Pause and ScriptTimer to delay execution of scripts either for data processing purposes or waiting for some event like user input. There are two specific timer-related commands - with PageTimer you set delay before moving to some other project page, and with ExitTimer command MMB waits specified amount of time before it exits the project.

RunScript command won't make any delays - it will immediately run specified script.

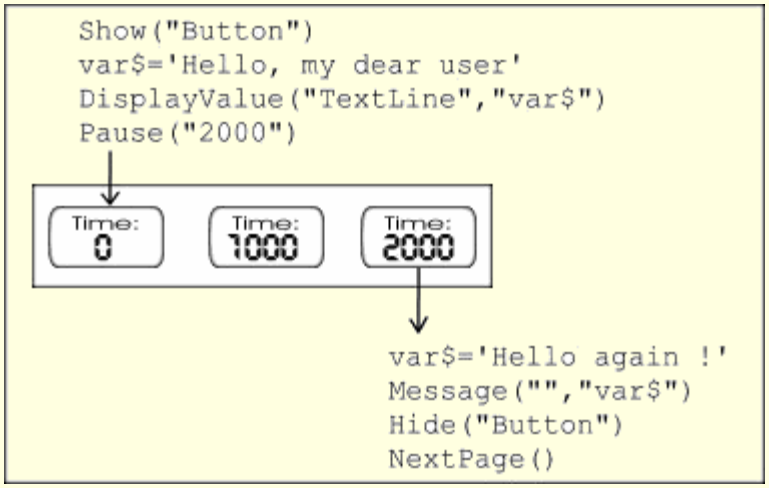
Both ScriptTimer and RunScript don't interrupt running of current script when calling another one - ScriptTimer will immediately continue with code lines of it's current script, RunScript will wait for called script to finish and then continue with code lines (if any) below.

All timer-related commands use millisecond as a time unit. One second contains 1000 milliseconds, and you'll often use *n x 1000* to manage delays in seconds.

Pause("ms")

Description

Delays further running of current script for specified amount of time.



Code Examples

```
**Pause script for 5 seconds (5000 ms) :
Pause("5000")

** Pause script for 12 seconds (12000 ms) using numerical variable:
delay=12000
Pause("delay")
```

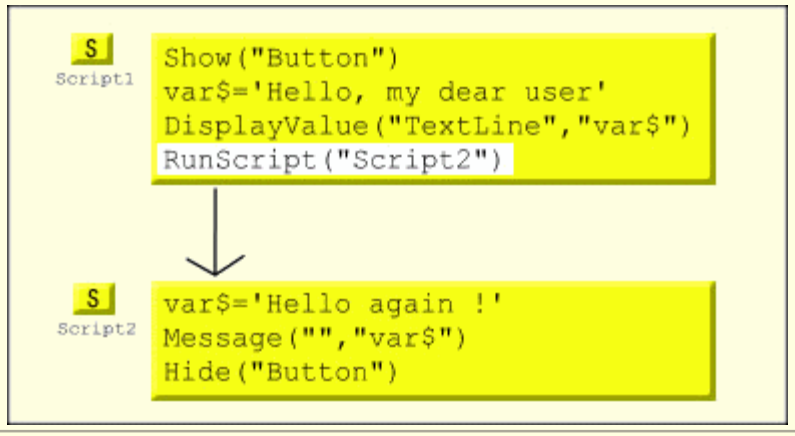
Additional Info

This command is occasionally being used to make delay for user input or wait for some external task (like data processing, copying) to finish.

RunScript("ObjectLabel")

Description	

Runs script specified as command parameter.



Code Example	

```
**Run script labeled "Calculate"  :
RunScript("Calculate")
```

Additional Info	

Running of other scripts saves you from re-writing code lines - it's recommended to put frequently used routines under separate script objects and run 'em whenever they're needed. Code lines below RunScript line will in most cases wait for called script to finish it's work, so put and retrieve processing results using string and numerical variables.

This command runs default (Mouse Up) script on objects with both *Mouse Down* and *Mouse Up* events (Buttons, Bitmaps, Rectangles, etc).

WARNING! Do not use RunScript command to call the same Script object/ MouseUp event multiple times and from the same Script object! This will cause an infinite recursion, because RunScript wait for its end. After using RunScript in that type of code you can see "Recursion in Script reached 50 levels" error message. You can use ScriptTimer instead, which doesn't wait for end.

RunScriptCode("parameter1","parameter2")

Description	

RunScriptCode runs external scripts form string or string variable. It means that

you can write any script you want in plain txt file, load entire file to string variable (using for example **LoadText** function) and run it over RunScriptCode. It's easy as that ;)

Available parameters:
paremeter1 string or string variable holding the script source code
paremeter2 0=quiet parsing, 1=loud parsing (1 = runtime errors are displayed)

Code Example

```
** Here is how to load and run an external file with script
LoadText("ExtScript$","<Embedded>\pagestart.txt")
RunScriptCode("ExtScript$","")

** Here is how to define a script code directly in other script
script$='For i=1 To 1000' + CHR(13) + CHR(10)
script$=script$ + 'm=m+1' + CHR(13) + CHR(10)
script$=script$ + 'Next i'
RunScriptCode("script$","")
```

Additional Info

Running of other scripts saves you from re-writing code lines - it's recommended to put frequently used routines in external or even embedded files and run them whenever they're needed. Code lines below RunScriptCode line will wait for called script to finish it's work.

WARNING! Do not use RunScriptCommand command to call the same Script multiple times and from the same Script! This will cause an infinite recursion, because RunScriptCode wait for its end. After using RunScriptCode in that type of code you can see "Recursion in Script reached 50 levels" error message. You can use ScriptTimer instead, which doesn't wait for end.

ScriptTimer("ObjectLabel","ms")

Description	

Waits for specified amount of time (number or numerical variable as second parameter) and then starts execution of another script, specified as first command parameter:

```
ScriptTimer("ScriptLabel","TimeDelay")
```

To give you extra power, MMB introduces 1000 independent timers inside ScriptTimer command called Timer1..Timer1000. Using this principle, you can call ScriptTimer command 1000 times, using different timers and make 1000 independent *threads* - every thread runs it's own script and won't care much about others.

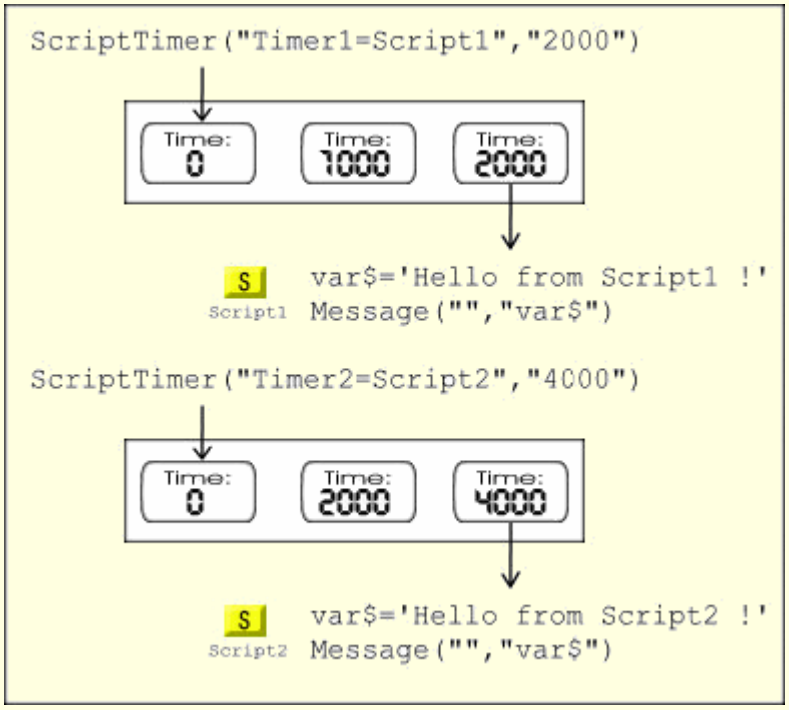
NOTE: In a previous versions, there were only three timers - TimerA, TimerB and TimerC. These timers are still active and you can see them in old projects and scripts. But we do not recommend to use them in new projects!

Using multiple timers starts by writing their labels...

Timer1, **Timer2,**..., **Timer1000**

...together with equal sign as a script label prefix:

```
ScriptTimer("Timer1=ScriptLabel1","TimeDelay")

ScriptTimer("Timer2=ScriptLabel2","TimeDelay")

ScriptTimer("Timer3=ScriptLabel3","TimeDelay")
```



Code Examples

** Run Script1 with delay of 8 seconds :
```
ScriptTimer("Script1","8000")
```

** Run MyScript with delay specified using numerical variable :
```
delay=5500
ScriptTimer("Script1","delay")
```

** Run ThreadScript with 5 second delay and using Timer1:
```
ScriptTimer("Timer1=ThreadScript","5000")
```

Additional Info

This command is occasionally being used to make delay for user input or wait for some external task (like data processing, copying) to finish.

ScriptTimer runs script only once, so you have to call ScriptTimer command again to repeat script running.

In addition to this, running of ScriptTimer before previously set timer delay has elapsed will reset previous setting and use the new one.

For example, if you run:

```
ScriptTimer("Script1","8000")
```

...and before elapsed 8 seconds you call ScriptTimer again:

```
ScriptTimer("Script2","12000")
```

...previous setting will be substituted with the new one, running Script2 after 12 seconds. This is also the case for individual timers Timers1, Timers2, etc...

ScriptTimer command runs default (Mouse Up) script on objects with both *Mouse Down* and *Mouse Up* events (Buttons, Bitmaps, Rectangles, etc).

All running scripts (including ScriptTimers) running from ordinary Page are terminated by jump to another Page. If you want to preserve run of a ScripTimer even if user jumps between pages, place the Script object on Master Page/Layer and then call it with **Master Page::** or **Master Layer::** prefix (as described **here**).

Then the script should appear like this:
```
ScriptTimer("Master Page::Script","100")
```
or this if you want to use multiple timers.
```
ScriptTimer("Timer1=Master Page::Script","100")
```

PageTimer("ms","PageLabel")

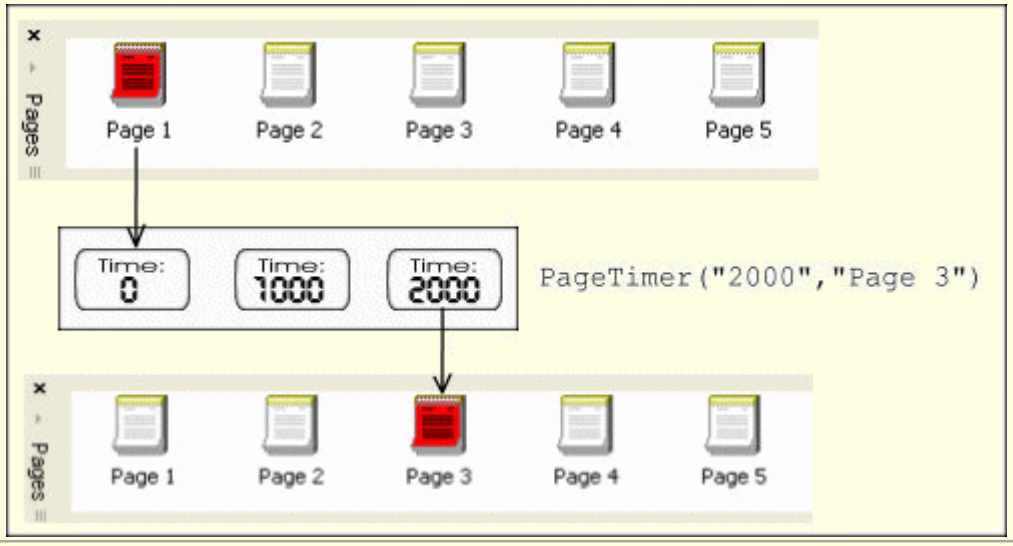
Description	

Waits for specified amount of time (number or numerical variable as the first parameter) and then either moves to the next page (if second parameter is empty), page specified as the second parameter, or performs one of the following actions specified as the second command parameter:

THIS_PAGE - restarts current project page (object positions, background music) and runs it's startup script

THIS_SCRIPT - runs only startup script of current project page

IF_IDLE - moves to next page if there's no keyboard or mouse click event for specified amount of time



Code Examples

```
** Move to next page after 8 seconds:
PageTimer("8000","")

**Move to Page 7 after 20 seconds :
PageTimer("20000","Page 7")

**Run startup script of current page after 10 seconds :
PageTimer("10000","THIS_SCRIPT")

**Restart current page after 5 seconds :
PageTimer("5000","THIS_PAGE")

**Move to next page if no user interaction for 60 seconds :
PageTimer("60000","IF_IDLE")
```

Additional Info

Running of PageTimer before previously set timer delay has elapsed will reset previous setting and use the new one.

For example, if you run:

```
PageTimer("120000","")
```

...and before elapsed 120 seconds you call PageTimer again:

```
PageTimer("180000","")
```

...previous setting will be substituted with the new one and timer will move to next

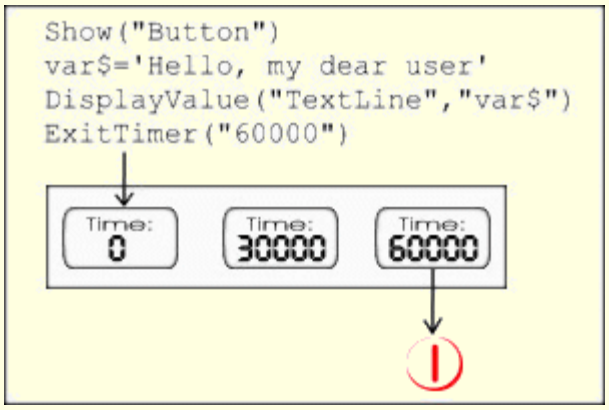
page after 180 seconds.

Using ExitTimer after PageTimer will cancel PageTimer event.

ExitTimer("ms")

Description

Waits for specified amount of time (number or numerical variable as command parameter) and then exits (closes) project. If keyboard or mouse click event occurs during delay, ExitTimer is canceled.



Code Example

```
**Exit from project after 120 seconds of user inactivity:
PageTimer("120000")
```

Additional Info

Running of PageTimer before ExitTimer delay has elapsed will cancel ExitTimer event.

For example, if you run:

```
ExitTimer("120000","")
```

...and before elapsed 120 seconds you call PageTimer:

```
PageTimer("12000","")
```

...PageTimer will cancel ExitTimer event.

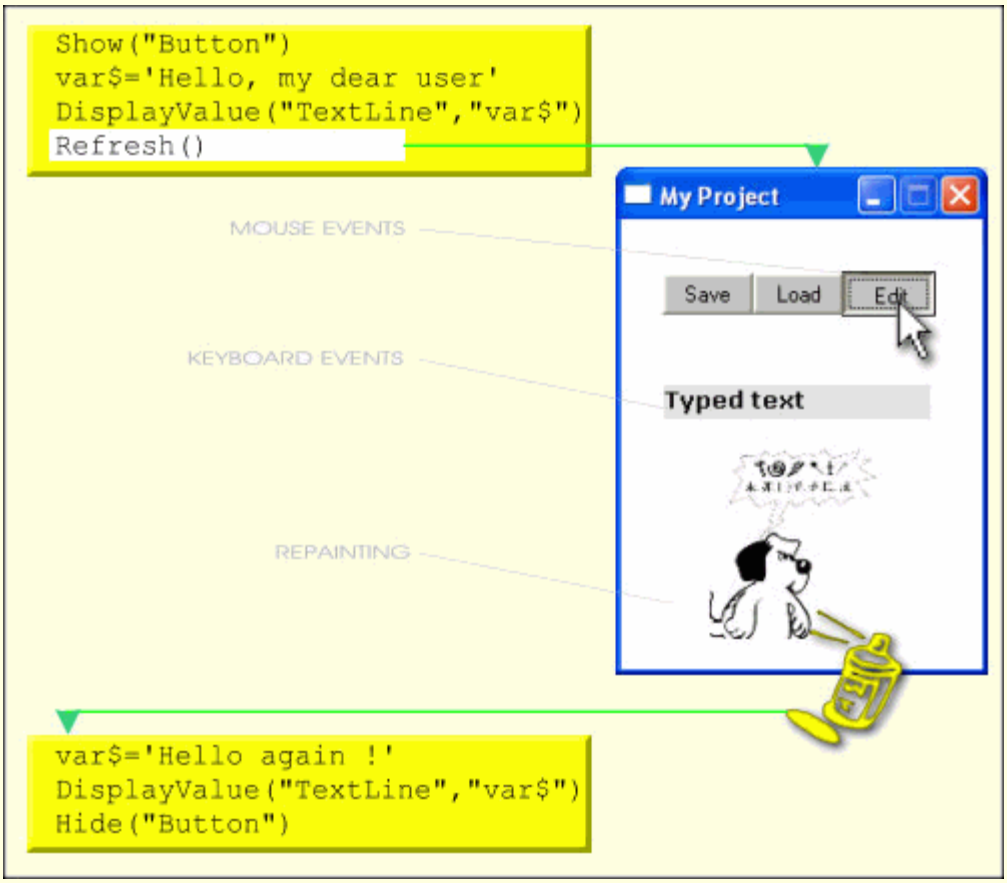
Refresh("FORCED")

Description

Redraws project's window and allows processing of other events in project (mouse & keyboard input, running of scripts, etc).

Refresh without FORCED parameter is just the standard refresh. It should be good enough in mos cases.

FORCED - Forced refresh of entire project window. This option is useful in cases, when the standard Refresh becomes insufficient (some graphically intensive operations). We would recommend to use this parameter only in case of problems with standard refresh.



Code Example

```
** Perform standard refresh
Refresh("")

** Perform FORCED refresh. Use it carefully! It can slowdown your program.
Refresh("FORCED")
```

Additional Info

Calling this command is highly recommended in for..next loops, to allow processing of other threads (running routines also present in project) and avoid 'freezing' of application window.

12.10.6 Dialog Box Commands

Authors make adjustments to graphic interface of their applications, but there are some standard Windows dialog boxes that are used by all applications.
MMB introduces this standard dialog boxes, available through it's script language:

- Message Box
- Message Box Ex
- Open File dialog box
- Save File dialog box
- Browse for Folder dialog box
- Color Picker dialog box
- Font Picker dialog

Message("String","Variable")

Description

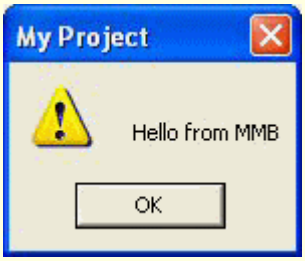
Displays Message box with OK button and exclamation mark.
Use first parameter to set message you want to display:

```
Hello from MMB
```

Second parameter can be used to show a value of any string or numerical variable. This is especially useful when debugging a program, to check current variable value:

```
UserFile$
```

Here's an example of a Message box:



Code Examples

```
**Show Message box without variables:
Message("Hello from MMB", "")

**Show Message box with numerical variable as 2nd parameter:
year=2004
Message("Selected year: ", "year")
```

```
**Show Message box with string variable as 2nd parameter:
name$='Sam'
Message("Welcome, ","name$")

**Show Message box with 2 string variables as parameters:
greeting$='Hello,'
name$='Peter'
Message("greeting$","name$")
```

Additional Info

Message box is displayed on top and project's window can't be used until user closes message box.

MessageEx("Title","text, flag[, timeout]")

Description

Displays fully configurable Message box.
The first parameter defines the Message box title bar. The second parameter is a comma separated list of parameters defining the rest of Message box parameters like a number of buttons, displayed icon, default button, etc..

Description:

title:	title bar text
text:	text in message box
flag:	indicates the type of message box and the possible button combinations
timeout:	[optional parameter] Timeout in milliseconds. The message box will be automatically closed after timeout expiration

Return value after Message box confirmation/expiration is stored in **CBK_MsgEx** where..

Success => returns the **ID** of pressed button.

Timeout => returns **-1** if the message box timed out.

message buttons return ID values	
OK	1
CANCEL	2

ABORT	3
RETRY	4
IGNORE	5
YES	6
NO	7
TRY AGAIN	10 **
CONTINUE	11 **
** only on Windows 2000/XP and above.	

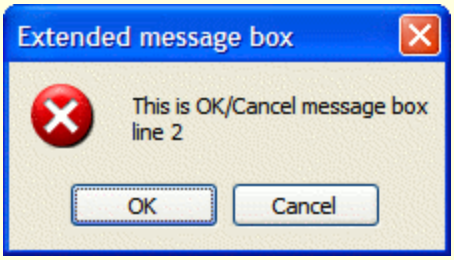
The message **Flag** parameter can be a combination of these values:

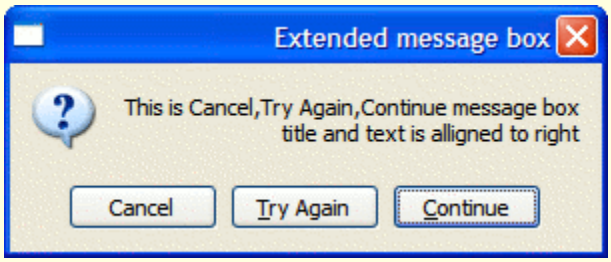
button related flags	
OK button	0
OK and Cancel	1
Abort, Retry, and Ignore	2
Yes, No, and Cancel	3
Yes and No	4
Retry and Cancel	5
Cancel, Try Again, Continue	6 **
** only on Windows 2000/XP and above.	
icon related flags	
No icon (default)	0
Stop sign	16
Question mark	32
Exclamation mark	48
'i' Icon consisting of an 'i' in a circle	64
default button related flags	
First button is default button (default)	0
Second button is default button	256
Third button is default button	512
message box modality related flags	
Application (default)	0

System modal (dialog has an icon in title bar)	4096
Task modal	8192
miscellaneous flags	
nothing special (default)	0
MsgBox has top-most attribute set	262144
title and text are right justified	524288

How to use the above flags? Actually, it's pretty simple! Just select the flags you would like to see in the message box and count them up. So for example, OK/ Cancel buttons + Stop sign + 2nd button as default will require flags 1 + 16 + 256 = **273** and this value should be used as the above mentioned **Flag** parameter.

Here you can see two examples of extended message box:





Code Examples

```
** here is an example of message box with "Abort/Retry/Ignore" buttons
(2) + Question mark (32) + 3rd button as default (512)
Title$ = 'Extended message box'
Text$ = 'This is Abort/Retry/Ignore message box.'
Flag= 2 + 32 + 512
Params$ = Text$+','+CHAR(Flag)
MessageEx("Title$","Params$")

** here is an example of message box with comma character inside text
Title$ = 'Extended message box'
```

```
Text$ = '"This is, message box"'
Params$ = Text$ + ',64'
MessageEx("Title$","Params$")
```

Additional Info

IMPORTANT REMARKS!

A message box appears centered on the application screen.

The size of message box may vary according to the text it contains.

The title could get truncated if too long and the SYSTEMMODAL flag is used.

The title bar icon in case of SYSTEMMODAL cannot by changed :(

If you want to use comma character inside the "text" flag, you have to enclose entire text in an additional pair of double quotes..like this..

```
Text$ = '"This is, message box"'
```

And NO, there is no way to change/replace the button labels!

OpenFile("Filter","DefaulExt")

Description

Displays Open File dialog box. First parameter sets extension filter, default extension is specified as a second parameter.

IMPORTANT! OpenFile command does not open anything! It's just a dialog box, which fills the MMB path macros with selected path!

Extension filter - sets what file formats will be displayed in a dialog box. MMB supports multiple masks, selectable through "Files of type" combobox inside Open File dialog box. Filter is set in a string as the first command parameter. Elements and their parts are separated using **|** character.

Mask element is comprised of label and extension:

```
MPEG Files (*.mpg) | *.mpg
```

Blue part is extension element label, green part is extension itself. They're separated with | character. You'll write another extension element behind this one, separating it again using | character. So extension filter will now look like this:

```
MPEG Files (*.mpg) | *.mpg| AVI Files (*.avi)| *.avi
```

When you're done adding extension elements, | character is needed at the end of parameter to signal MMB about it:

```
MPEG Files (*.mpg) | *.mpg| AVI Files (*.avi)| *.avi |
```

There. Line above represents first command parameter. It is also common to specify "All files" as the last extension element, giving users ability to see all files (turn filtering off). Here's how such version of first command parameter looks like:

```
MPEG Files (*.mpg) | *.mpg| AVI Files (*.avi)| *.avi | All Files (*.*)
| *.* |
```

It is also possible to specify multiple extensions in one element, giving users ability to choose among similar file extensions (.mpg & .mpeg, .htm & html, .jpg & .jpeg...) . Use **;** as an extension delimiter:

```
Movie Files (*.mpg,*.avi) | *.mpg ;*.avi| All Files (*.*)| *.* |
```

Default extension element, the one that will be used for file filtering when Open File dialog box appears, is set using second command parameter.

It's one of element extensions specified as the first parameter, for e.g.

```
*.mpg
```

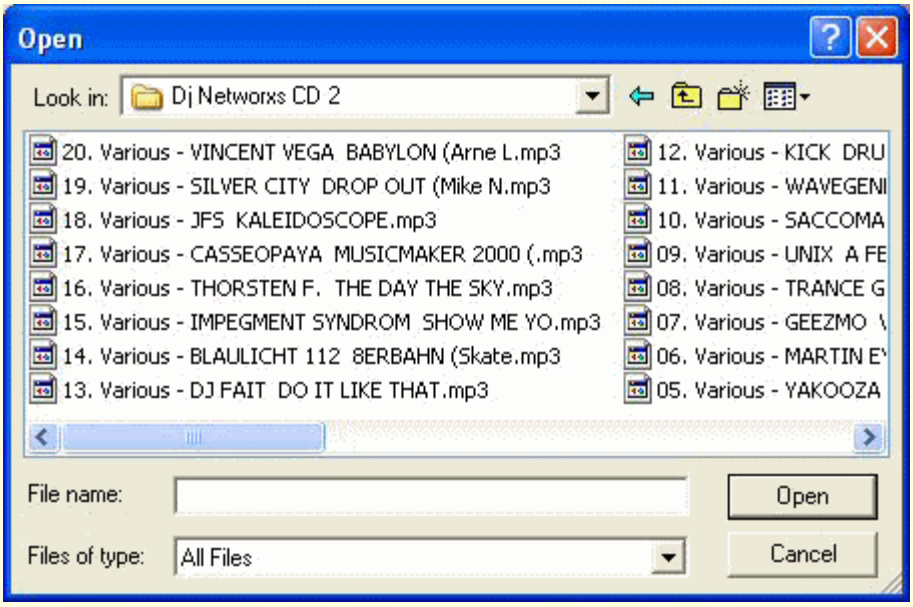
Using multiple extensions in one element affects default extension too, so you will also use ; as extension delimiter:

```
*.mpg ;*.avi
```

Put into OpenFile command, parameters look like this:

```
OpenFile(" Movie Files (*.mpg,*.avi) | *.mpg ;*.avi| All Files (*.*)|
*.* | "," *.mpg ;*.avi ")
```

Here's example of Open File dialog box, showing all files:



Once user selects file and clicks Open button (or double-clicks selected file), MMB fills **OpenFile$** string variable with full folder path and file name of selected file.

Three constants also contain dialog box result:

<File> - contains the same data as OpenFile$ variable
CBK_OpenFile - contains selected file name without folder path
CBK_OpenDir - contains folder path without name of selected file

Code Examples

```
**Show Open File dialog box with filters for MP3, WAV, MID and MOD
**files with WAV as default filter
OpenFile(" MP3 Files (*.mp3)|*.mp3|WAVE Files (*.wav)|*.wav|MIDI Files
(*.mid)|*.mid|MOD Files (*.mod)|*.mod|","*.wav")

**Show Open File dialog box with one filter for MP3, WAV, MID and MOD
**files, labeled "Sound Files"
OpenFile(" Sound Files (*.mp3,*.wav,*.mid,*.mod) |*.mp3;*.wav;*.mid;*.
mod|","*.mp3;*.wav;*.mid;*.mod")

**Show Open File dialog box with filters for EXE and All files
**with EXE as default filter
OpenFile(" Executable Files (*.exe)|*.exe|All Files (*.*)|*.*|","*.
exe")

**Show Open File dialog box with filters for HTML and all files
```

```
**with HTML as default filter. Result is displayed in message boxes
OpenFile(" HTML Files (*.htm)|*.htm|All Files (*.*)|*.*|","*.htm")
Message("Selected file in OpenFile$ variable: ","OpenFile$")
file$=<File>
Message("Selected file in <File> constant: ","file$")
file$=CBK_OpenFile
Message("Selected file name: ","file$")
folder$=CBK_OpenDir
Message("Folder path of selected file: ","folder$")

**Show Open File dialog box at defined position (source application directory in this case) and with filter for TXT files.
Path$=<SrcDir>+'*.txt'
OpenFile("TXT Files (*.txt)|*.txt|All Files (*.*)|*.*||","Path$")
```

Additional Info

Dialog box is displayed on top and project's window can't be used until user closes dialog box.

SaveFile("Filter","DefualtExt")

Description

Displays Save File dialog box. First parameter sets extension filter, default extension is specified as a second parameter.

IMPORTANT! SaveFile command does not save anything! It's just a dialog box, which fills the MMB path macros with selected path!
Extension filter - sets what file formats will be displayed in a dialog box. MMB supports multiple masks, selectable through "Files of type" combobox inside Save File dialog box. Filter is set in a string as first command parameter. Elements and their parts are separated using **|** character.

Mask element is comprised of a label and extension:

```
TXT Files (*.txt) | *.txt
```

Blue part is extension element label, green part is extension itself. They're separated with | character. You'll write another extension element behind this one, separating it again using | character. So extension filter will now look like this:

```
TXT Files (*.txt) | *.txt| LOG Files (*.log)| *.log
```

When you're done adding extension elements, | character is needed at the end of parameter to signal MMB about it:

```
TXT Files (*.txt) | *.txt| LOG Files (*.log)| *.log |
```

Line above represents first command parameter. It is also common to specify "All files" as the last extension element, giving users ability to see all files (turn filtering off). Here's how such version of first command parameter looks like:

```
TXT Files (*.txt) | *.txt| LOG Files (*.log)| *.log | All Files (*.*) |
*.* |
```

It is also possible to specify multiple extensions in one element, giving users ability to choose among similar file extensions (.mpg & .mpeg, .htm & html, .jpg & . jpeg...) . Use ; as extension delimiter:

```
Text Files (*.txt,*.log) | *.txt ;*.log| All Files (*.*)| *.* |
```

Default extension element, the one that will be used for file filtering when Save File dialog box is displayed, is set using second command parameter.

It's one of element extensions specified in first parameter, for e.g.

```
*.txt
```

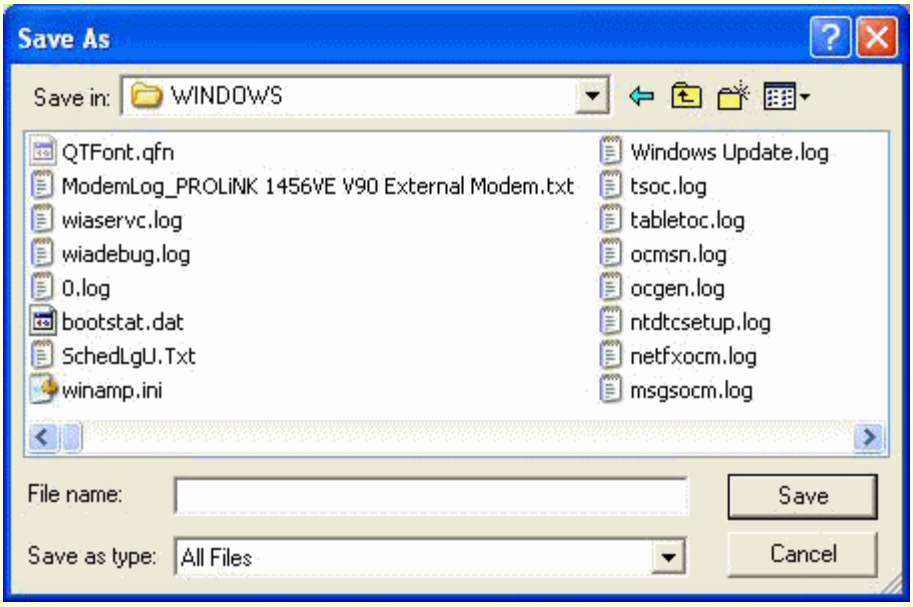
Using multiple extensions in one element affects default extension too, so you will also use **;** as an extension delimiter:

```
*.txt ;*.log
```

Put into SaveFile command, parameters look like this:

```
SaveFile(" Text Files (*.txt,*.log) | *.txt ;*.log| All Files (*.*)| *.
* | "," *.txt ;*.log ")
```

Here's example of Save File dialog box, showing all files:



Once user selects or writes file name and clicks Save button (or double-clicks selected file), MMB will fill **OpenFile$** string variable with full folder path and file name of selected file.

Three constants also contain dialog box result:

<File> - contains the same data as OpenFile$ variable
CBK_OpenFile - contains selected file name without folder path
CBK_OpenDir - contains folder path without name of selected file

Code Examples

```
**Show Save File dialog box with filters for TXT and LOG
**files with TXT as default filter
SaveFile(" TXT Files (*.txt)|*.txt|LOG Files (*.log)|*.log|", "*.txt")

**Show Save File dialog box with one filter for TXT and LOG
**files, labeled "Text Files"
SaveFile(" Text Files (*.txt,*.log)|*.txt;*.log|", "*.txt;*.log")

**Show Save File dialog box with filters for TXT and All files
**with TXT as default filter
SaveFile(" TXT Files (*.txt)|*.txt|All Files (*.*)|*.*|","*.txt")

**Show Save File dialog box with filters for TXT and All files
**with TXT as default filter. Result is displayed in message boxes
```

```
SaveFile(" TXT Files (*.txt)|*.txt|All Files (*.*)|*.*|","*.txt")
Message("Selected file in OpenFile$ variable: ","OpenFile$")
file$=<File>
Message("Selected file in <File> constant: ","file$")
file$=CBK_OpenFile
Message("Selected file name: ","file$")
folder$=CBK_OpenDir
Message("Folder path of selected file: ","folder$")

**Show Save File dialog box at defined position (source application directory in this case) and with filter for TXT files.
Path$=<SrcDir>+'*.txt'
SaveFile("TXT Files (*.txt)|*.txt|All Files (*.*)|*.*||","Path$")
```

Additional Info

Dialog box is displayed on top and project's window can't be used until user closes dialog box.

BrowseForFolder("Prompt","StartFolder")

Description

Displays Browse for Folder dialog box. First parameter sets dialog description, second parameter sets starting (root) folder.

Both parameters are optional.

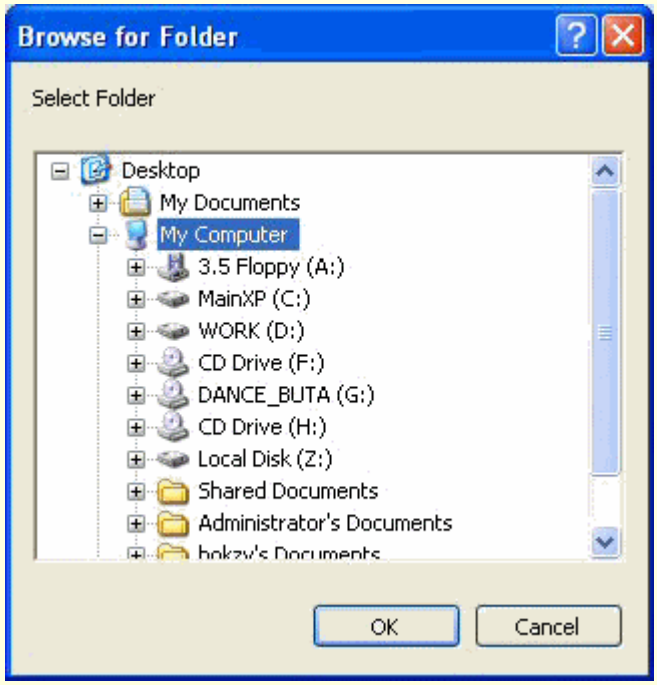
Dialog description - defines prompt that will be displayed in Browse for Folder dialog box. Usual prompt is:

```
Select folder:
```

Starting folder - sets root folder and disables browsing for upper folders. Default starting folder is:

```
My Computer
```

Here's an example of Browse for Folder dialog box:



Once user selects folder and clicks OK button, constant **CBK_OpenDir** will contain selected folder path.

Code Examples

```
**Show basic Browse for Folder dialog box:
BrowseForFolder("","" )

**Show Browse for Folder dialog box with custom prompt:
BrowseForFolder("Select Folder: ","" )

**Show Save File dialog box with filters for TXT and All files
**with TXT as default filter
SaveFile(" TXT Files (*.txt)|*.txt|All Files (*.*)|*.*|","*.txt")

**Show Browse for Folder dialog box with custom prompt
**and c:\ as starting folder
BrowseForFolder("Select Folder: ","c:\" )

**Show Browse for Folder dialog box with custom prompt
**and starting folder specified through string variables
prompt$='Select directory'
root$='c:\\'
BrowseForFolder("prompt$","root$" )
```

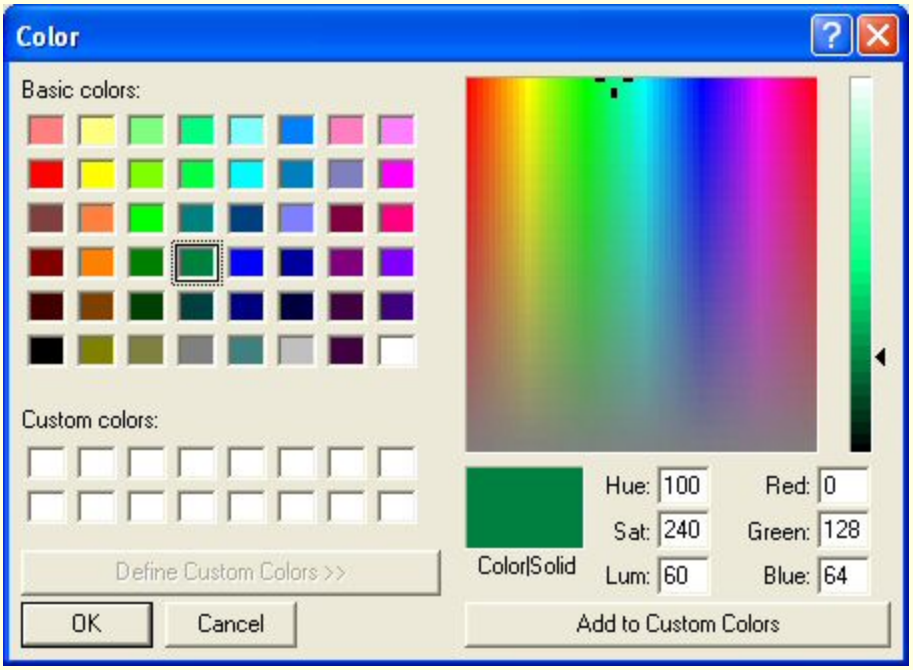
Additional Info	

Dialog box is displayed on top and project's window can't be used until user closes dialog box.

ColorPicker()

Description	

Displays Color Picker dialog box. Uses no input parameters.



Once user selects color and clicks OK button, constant **CBK_SelColor** will contain selected color.

Color picking uses RGB (Red, Green, Blue) system containing 3 components.

Each component can have one of 256 levels. Greater value of component increases color intensity. If you specify value 0, color component will not be used. Value 255 uses maximum color intensity.

Result of Color Picker is a comma-separated array of string values, each representing one RGB component (first: red, second: green, third: blue) in value

range 0-255.

R,G,B	Result
128,5,64	

R,G,B	Result
0,255,255	

To retrieve color value, simply assign contents of CBK_SelColor to a string variable:

```
color$=CBK_SelColor
```

Returned color value looks like this:

```
119,230,95
```

Code Examples

**Show Color Picker dialog box:

```
ColorPicker( )
```

**Show Color Picker dialog box, retrieve and show selected color string:

```
ColorPicker( )
color$=CBK_SelColor
Message("Selected color is: ","color$")
```

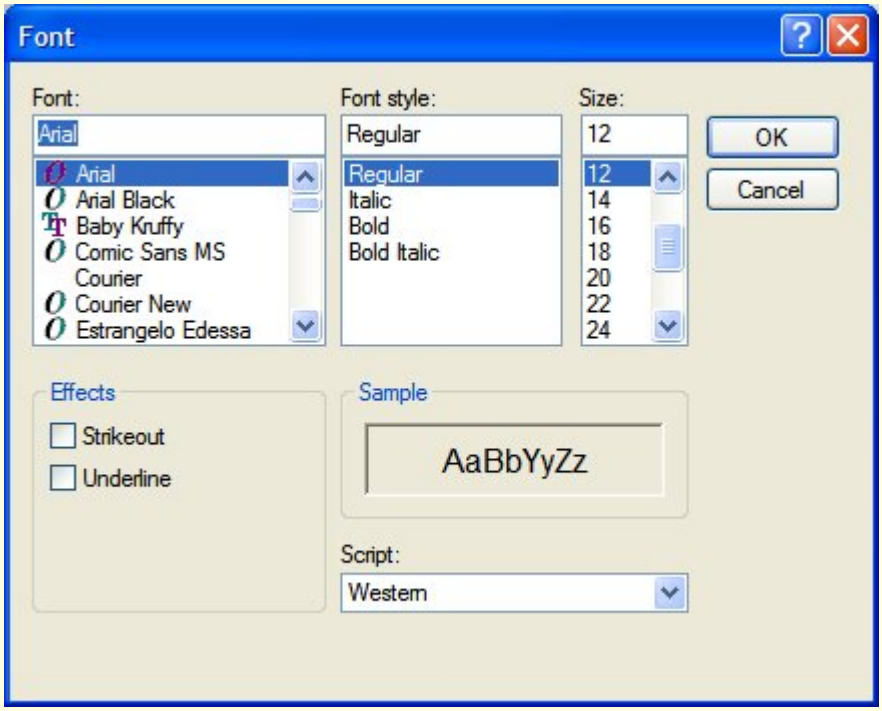
Additional Info

Dialog box is displayed on top and project's window can't be used until user closes dialog box.

FontPicker()

Description

Displays Font Picker dialog box. Uses no input parameters.



Once user selects color and clicks OK button, constant **CBK_SelFont** is filled with selected font parameters. The parameters are stored in this format (just a simple string array):
FONTNAME|FONTSTYLE|FONTSIZE|FONTSCRIPT|FONTEFFECT|

The individual parameters can be obtained from this array with **GetArrayItem** array function.

For an example of FontPicker dialog and the SetObjectParam function check the supplied 497_test_project.mbd sample project.

Code Examples

```
**Show Font Picker dialog box:
FontPicker()

**Show Font Picker dialog box, retrieve and show selected font name and its parameters in a Message box:
FontPicker()
```

```
FontParams$=CBK_Font
If (FontParams$<>'') Then
**font name
  item$[1]=GetArrayItem(FontParams$,|,1)
**font style
  item$[2]=GetArrayItem(FontParams$,|,2)
**font size
  item$[3]=GetArrayItem(FontParams$,|,3)
**font script
  item$[4]=GetArrayItem(FontParams$,|,4)
**font effect
  item$[5]=GetArrayItem(FontParams$,|,5)
  msg$= 'FONTNAME = ' + item$[1] + CHR(13) + CHR(10) + 'FONTSTYLE = ' +
item$[2] + CHR(13) + CHR(10) + 'FONTSIZE = ' + item$[3] + CHR(13) + CHR
(10) + 'FONTSCRIPT = ' + item$[4] + CHR(13) + CHR(10) + 'FONTEFFECT = '
+ item$[5] + CHR(13) + CHR(10)
  Message("Font parameters:","msg$")
End
```

Additional Info	

Dialog box is displayed on top and project's window can't be used until user closes dialog box.

12.10.7 System Commands

General-purpose commands that make your life with scripting easier are included in this section. System-oriented, these commands handle Windows Registry entries, file operations, Clipboard, installing of fonts, et

SysCommand("Command","Parameters")

Description	

A bit unconventional command, having more functions inside of it, handles window-related operations and file copying.

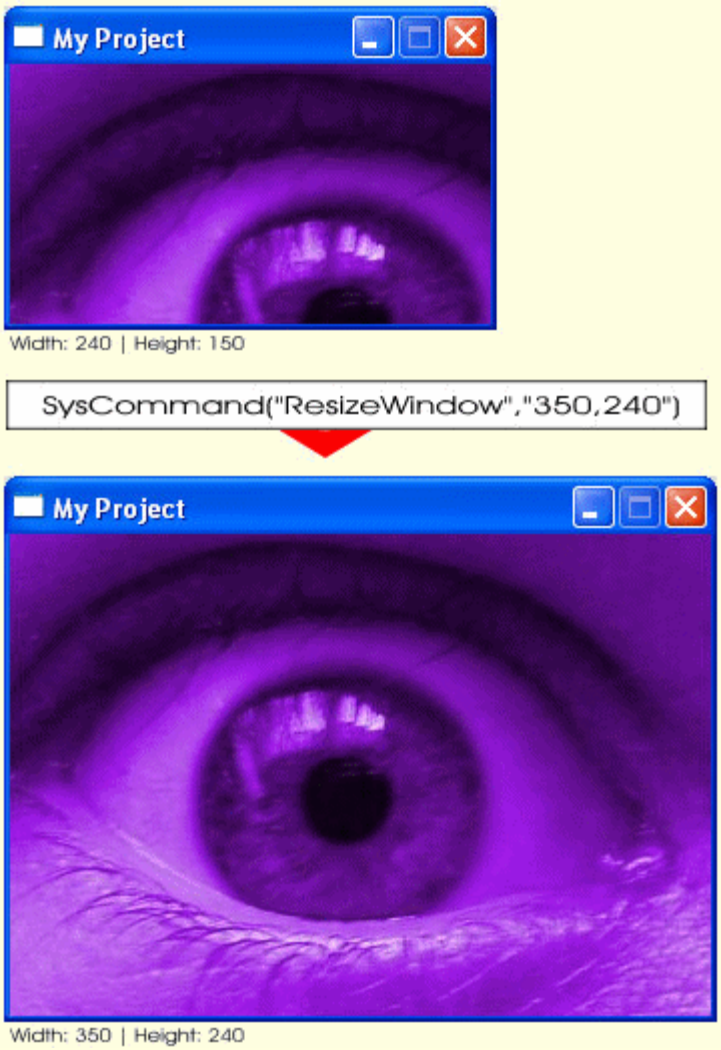
First parameter is command-specific function. Depending on purpose, such function may require one or two arguments (set as second parameter).

Functions are not case-sensitive.

ResizeWindow

Description	

Resizes program window to width and height values set as a second parameter. Both fixed and numerical variable parameters can be used.



First write width value:

```
640
```

...and after adding comma write height value:

```
480
```

Integrated with SysCommand:

```
SysCommand("ResizeWindow","640,480")
```

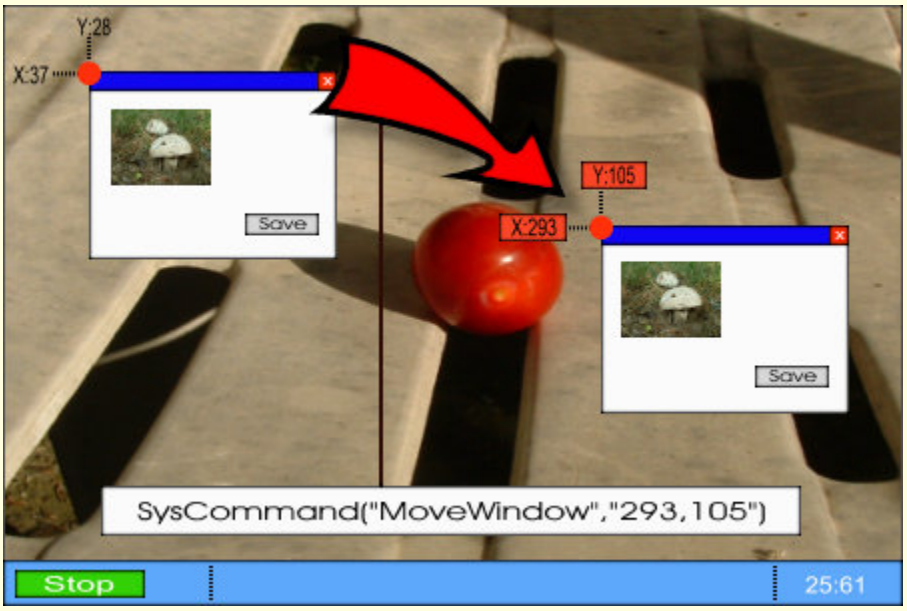
Example	

```
**Resize program window to 320 (width) x 200 (height)
SysCommand("ResizeWindow","320,200")
```

MoveWindow

Description

Moves program window to x (horizontal) and y (vertical) values set as a second parameter.



Both fixed and numerical variable parameters can be used.

First write x (horizontal) value:

```
200
```

...and after adding comma write y (vertical) value:

```
150
```

Integrated with SysCommand:

```
SysCommand("MoveWindow","200,150")
```

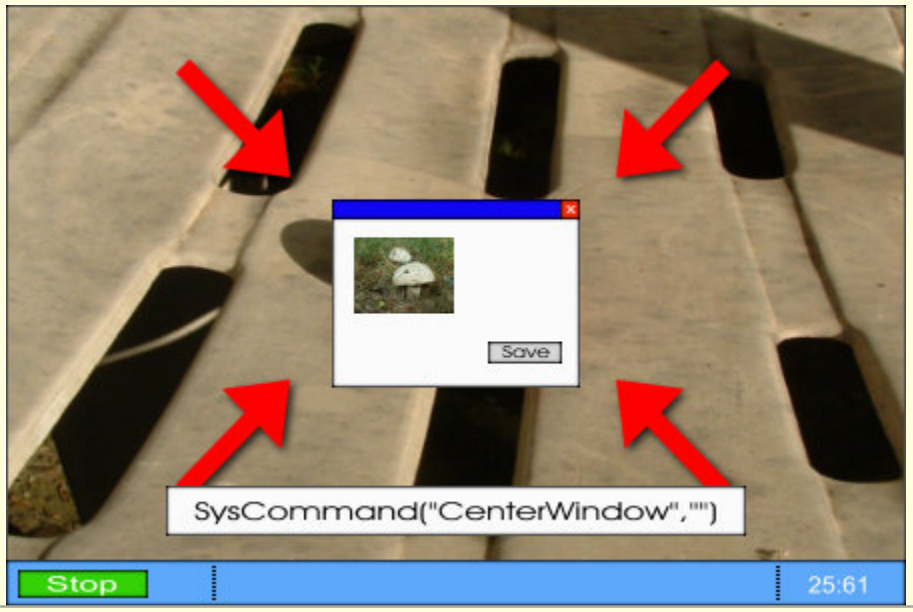
Example

```
**Move program window to 300 (x), 420 (y)
SysCommand("MoveWindow","300,420")
```

CenterWindow

Description

Centers program window on the screen. No parameters required.



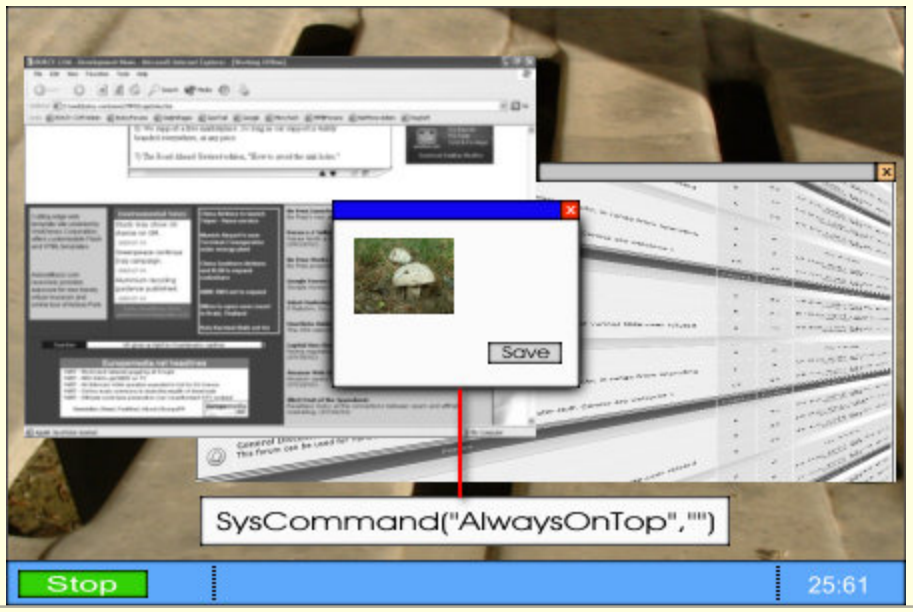
Example

```
**Center application window on screen
SysCommand("CenterWindow","")
```

AlwaysOnTop

Description

Puts program window on top of all opened windows and stays on top until some other window sets itself on top. No parameters required.



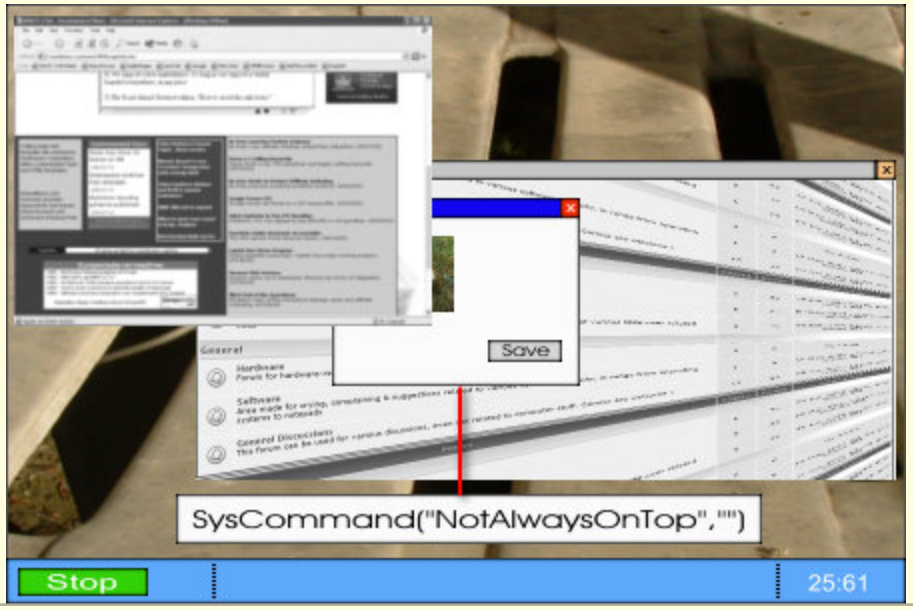
Example

```
**Set application window to 'always on top '
SysCommand("AlwaysOnTop","")
```

NotAlwaysOnTop

Description

Opposite to previous SysCommand function, this one restores program window status to normal (not on top).



Example	

```
**Restore application window to normal state
SysCommand("NotAlwaysOnTop","")
```

CopyFile

Description	

Copies source file to destination file. Both files are specified through second command parameter.

Files are set as either:

a) folder path + file name, or
b) **path macro** + file name

First write source file:

```
<SrcDir>\MyProgram.ini
```

...and after adding comma write destination file:

```
<Windows>\MyProgram.ini
```

Integrated with SysCommand:

```
SysCommand("CopyFile","<SrcDir>\MyProgram.ini,
<Windows>\MyProgram.ini")
```

If directory structure doesn't exist, it will be created.

Both source and destination file names can differ.

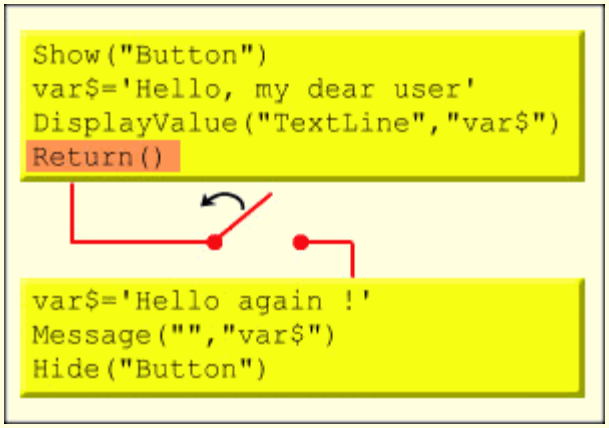
Example

```
**Copy file c:\Project\Readme.txt to a:\backup\BackupReadme.txt
SysCommand("CopyFile","c:\Project\Readme.txt,
a:\backup\BackupReadme.txt")
```

Return()

Description

Terminates execution of currently running script.



Does not affect other scripts run through script timers.

Code Example

```
**Terminate script execution:
Return()
```

Additional Info

This command is often being used in for..next loops and if-statements to terminate further execution when some event occurs.

Break()

Description

Jump from For..Next loop. While Return() skip the execution of entire code after its usage, Break() only jump from For..Next loop, but execution of script continues after Next keyword.

Does not affect other scripts run through script timers.

Code Example

This code is useless, but it will show you how the **Break()** exactly works;) After its execution you will get 5 messages from loop **i** but only two messages from **n** loop...

```
maxloop=5
For i=1 To maxloop
  Message("loop 1","i")
  For n=1 To 10
    If (n=3) Then
      Break()
    End
    Message("loop 2","n")
  Next n
Next i
```

...while The same code with **Return()** will show only one message from loop **i** and two messages from loop **n**. It's because Return() skips execution of entire code.

```
maxloop=5
For i=1 To maxloop
  Message("loop 1","i")
  For n=1 To 10
    If (n=3) Then
      Return()
    End
    Message("loop 2","n")
  Next n
Next i
```

Additional Info

This command is often being used in for..next loops and if-statements to terminate further execution when some event occurs. Look at included **break example.mbd**

FileExist("Path","Variable")

Description

Checks if specified file exists and returns result using numerical variable.

First parameter sets a file to check, as either:

a) folder path + file name, or
b) **path macro** + file name

Second parameter sets numerical variable that will receive the result of file existence check. This means that you'll retrieve info from this variable (not give as command input) and create cases using if-statements.

Checking existence of file <SrcDir>\text.txt using FileCheck as numerical variable for result:

```
FileExist ("<SrcDir>\text.txt","FileCheck")
```

...will output check result through FileCheck numerical variable and there are only two available states:

- 1 - file exists
- 0 - file doesn't exist

Using **if-statements** to make decision depending on received result is a very natural choice:

```
if(FileCheck=1) then
  Message("File exists !","")
else
  Message("File does not exist.","")
end
```

Code Example

```
**Check for existence of win.ini in <Windows> folder and display
**message box if it does - otherwise exit project:
FileExist("<Windows>\win.ini","check" )
if (check=1) then
  Message("WIN.INI found. Goody.","")
else
  Exit()
end
```

FileString("SubString","Variable")

Description

Checks if specified string exists in <File> constant and returns result through a numerical variable. Use this command after Open File dialog box command OpenFile() , usually to check file extension.

First parameter sets string to search for, usually file extension:

```
.avi
```

Second parameter sets numerical variable that will receive result of a string existence check. This means that you'll retrieve info from this variable (not give as a command input) and create cases using if-statements.

Checking for .avi file extension using Type as a numerical variable for result looks like this:

```
FileString(".avi","Type")
```

Result is given through Type numerical variable and there are only two available states:

- 1 - string exists
- 0 - string doesn't exist

Of course, you don't have to search for file extension. Here we look for existence of "My Documents" folder in <File> path and result is put into FolderCheck numerical variable:

```
FileString("My Documents","FolderCheck")
```

Using **if-statements** to make decision depending on received result will make your scripting easier:

```
if(FolderCheck=1) then
  Message("Thanks for choosing My Documents !","")
else
  Message("Please choose My Documents folder.","")
end
```

Code Example

```
**Check Open File dialog box selection for MP3 file type and display
**message box if type doesn't match. Otherwise play MP3 file:
FileString(".mp3","check" )
if (check=1) then
   PlaySound(" <File>")
else
  Message("Selected file is not MP3.","")
end
```

SaveVariable("Name","Variable")

Description

Saves contents of specified variable to Windows Registry.

Windows Registry is a database repository for information about a computer's configuration. The registry contains information that Windows continually references during operation - profiles for each user, programs installed on the computer, types of documents each program can handle, property settings for folders, hadware settins, etc.

You can use MMB's script commands SaveVariable and LoadVariable to handle Registry items under path assigned to your program:

`HKEY_CURRENT_USER\Software\MMBPlayer\ ProjectEXE\ RegName`

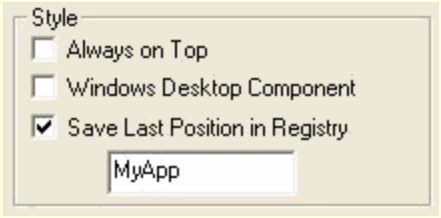
Let's split path above into sections.

HKEY_CURRENT_USER\Software\MMBPlayer\
Sets path to all MMB-made projects

ProjectEXE\
Represents file name of your compiled (EXE) project, without extension part. Program with file name MyAudioPlayer.exe would have this path assigned to it: MyAudioPlayer\

RegName

Represents name you specified in Project Settings to serve as a Registry path identifier of your application:



Even when "Save Last Position in Registry" option is turned off, specified identifier will be used for SaveVariable and LoadVariable commands. So you can set identifier label and turn off "Save Last Position..." - label is stored and that's where your program will save and load Registry items.

Summary of MMB project Registry path story:

Having compiled program labeled ImageViewer.exe and Registry identifier label set to "Viewer" will result in this Registry path:

```
HKEY_CURRENT_USER\Software\MMBPlayer\ ImageViewer\ Viewer
```

Every item you save or load using SaveVariable and LoadVariable commands is put into section under path above.

OK, now you know WHERE items are saved. More important is knowing HOW to save items in Registry.

SaveVariable command saves Registry items using path pattern discussed above. Use first parameter to set Registry item label, for example:

```
User Settings
```

Second command parameter sets source variable for Registry item contents. You can use both **string and numerical variables**. MMB will recognize type of variable and save item either as a String (for string variables) or DWORD (for numerical variables).

You don't have to worry about that. Here's an example of saving string variable content (pass$) to Registry item labeled "Password":

```
pass$='MySecretPassword'
```

```
SaveVariable("Password","pass$")
```

And example of saving numerical variable content (HiPoints) to Registry item labeled "UserPoints":

```
HiPoints=4910
SaveVariable("UserPoints","HiPoints")
```

At the minimal level, you'll use saving of user's input at the application exit. Just make sure to use individual, original labels for Registry items.

On program startup, use LoadVariable command to retrieve saved values - just like all grownup programs do !

Code Examples

```
**Save contents of string variable EditBox$ (assigned to EditBox object)
**to Registry item labeled "EditBoxInput"
SaveVariable("EditBoxInput"," EditBox$")

**Save contents of numerical variable Width (containing screen width)
**to Registry item labeled "CurrentWidth"
Width=ScreenWidth()
SaveVariable("CurrentWidth"," Width")

**Save contents of numerical variable InstFlag, that serves as a flag
**for application install status, to Registry item labeled "Installed"
InstFlag=1
SaveVariable("Installed"," InstFlag")
```

Additional Info

MMB's Registry item handling commands work with fixed paths. To have full access to all Registry items, use system-related MMB PlugIns.

LoadVariable("Name","Variable")

Description

Loads item from Windows Registry, that was previously saved using SaveVariable command, and sets retrieved contents to either string or numerical variable.

Read more about Windows Registry and item paths in SaveVariable command description box.

To load Registry item, use first parameter to set Registry item label you want retrieve value from, for example:

```
User Settings
```

Second command parameter sets destination variable for retrieved Registry item content. You can use both **string and numerical variables**. Make sure to set correct variable type.

Here's an example of loading Registry item labeled "Password" to string variable pass$:

```
Load Variable("Password","pass$")
```

And example of loading Registry item labeled "UserPoints" to numerical variable HiPoints:

```
LoadVariable("UserPoints","HiPoints")
```

Loading of Registry items is usually performed at the program startup, to load user input saved during last program session.

Of course, you can load items whenever you want :)

Code Examples

```
**Load Registry item labeled "EditBoxInput" to string variable
** EditBox$ and display it in EditBox object:
LoadVariable("EditBoxInput"," EditBox$")
LoadText("EditBox","EditBox$")

**Load Registry item labeled "CurrentWidth" to numerical variable
**Width and display it using message box:
LoadVariable("CurrentWidth"," Width ")
Message("Saved screen width is: ","Width")

**Load Registry item labeled "Installed", that serves as a flag
**for application install status, to a numerical variable InstFlag
**and make decision using if-statement:
LoadVariable("Installed","InstFlag ")
if (InstFlag=1) then
  NextPage()
else
  Message("Program is being run for the first time.","")
end
```

Additional Info

MMB's Registry item handling commands work with fixed paths. To have full access to all Registry items, use system-related MMB PlugIns.

InstallFont("Path")

Description

Users of MMB pay much attention to visual identity of their projects. In many cases, using standard font styles (pre-installed together with Windows) is not a satisfactory solution.

MMB helps in this case too - you only have to include font style in project distribution. Using InstallFont, MMB will check if user doesn't have specified font and if necessary - automatically install it.

On program exit, font will be uninstalled.

Only required parameter sets font file, as either:

a) folder path + file name, or
b) **path macro** + file name

Code Example

```
**Install font RoundStyle.ttf from <SrcDir> folder:
InstallFont("<SrcDir>\RoundStyle.ttf" )
```

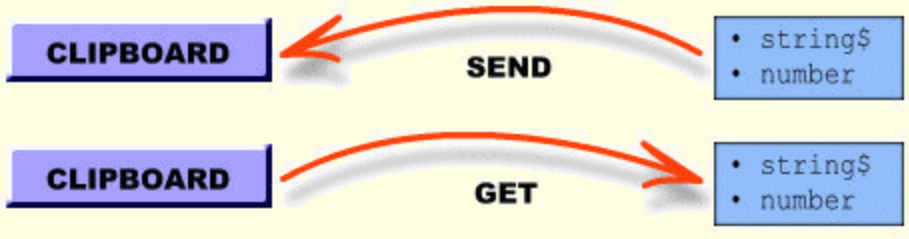
Clipboard("Send/Get","Variable")

Description

Performs communication with Windows Clipboard (container that holds everything you copy/paste).

Use first parameter to specify Clipboard function you want to use:

- `SEND` : puts variable content to Clipboard
- `GET` : retrieves content from Clipboard



Second parameter sets either string or numerical variable that will be used either as source or destination of content.

It is recommended to use string variables for content retrieving.

Code Examples

```
**Send content of string variable put$ to Clipboard:
Clipboard("SEND" ,"put$")

**Send content of numerical variable ClickCount to Clipboard:
Clipboard("SEND" ,"ClickCount")

**Retrieve content of Clipboard to string variable content$ and
**display it in a message box:
Clipboard("GET" ,"content$")
Message("Clipboard says: ","content$" )
```

Additional Info

You can only work with plain string and numerical data. Sending and retrieving of advanced content (bitmaps, objects) is not supported.

12.10.8 Audio Commands

Either you're making multimedia presentation or player, sound features are important part of entire story. MMB comes with variety of sound-related script commands, dealing with:

- Audio Volume
- Sound playback of following formats:
  MPG, MP1, MP2, OGG, WMA, ASF, WAV, MID, RMI, MOD, S3M, XM, IT
- Audio CD playback
- Audio Visualization

Volume Commands

VolumeUp("Volume")

Description
Increases Master Volume by 5% if no percentage is specified through command parameter. Otherwise sets volume to specified percentage value in range: 0-100

Code Example
Increase volume without custom percentage: **VolumeUp() **Set volume to 70%: **VolumeUp("70")** **Set volume to 20% using numerical variable: **volume=20** **VolumeUp("volume")**

Additional Info
Master volume is an output audio volume for all mixer lines.

VolumeDown()

Description
Decreases Master Volume by 5% . No parameters required.

Code Example

```
**Decrease Master Volume:
VolumeDown()
```

Additional Info
Master volume is an output audio volume for all mixer lines.

Sound Playback Commands

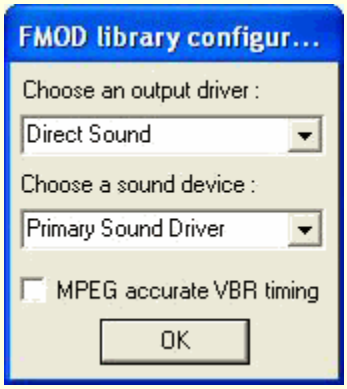
FMODConfig("Parameters")

Description

Displays FMOD library configuration dialog box, giving users ability to set output sound driver and device. MMB also allows setting configuration through this script command. If you call FMODConfig command without parameters...

```
FMODConfig("")
```

...configuration dialog box will appear:



As already mentioned alternative, configuration can be set through FMODConfig command itself using parameters, thus avoiding configuration dialog box. There are two parameters separated with comma.

With first one you set sound output driver using one of following numbers:

1 - DirectX sound
2 - Windows Media Wave Out
3 - No Sound

With second parameter you set status of MPEG accurate VBR (**V**ariable **B**it **R**ate) timing:

0 - MPEG accurate VBR timing off
1 - MPEG accurate VBR timing on

Put into command, parameters look like this:

```
FMODConfig("1,0")
```

In example above, green number represents sound output driver setting.
Red number represents status of MPEG accurate VBR timing.

Code Examples

```
**Show FMOD library configuration dialog box:
FMODConfig("")

**Set configuration to Windows Media Wave output driver with
**MPEG accurate VBR timing off:
FMODConfig("2,0")

**Set configuration to DirectX Sound output driver with
**MPEG accurate VBR timing on:
FMODConfig("2,1")
```

Additional Info

Configuration dialog box is displayed on top and project's window can't be used until user closes dialog box.

PlaySound("Path")

Description
Plays any supported type of sound file (Ogg, WMA, ASF, Wave, MOD, MIDI) specified as command parameter. Sound file parameter should contain either a) folder path + sound file name, or b) **path macro** + sound file name

Code Example
Play ogg sound file MyMusic.ogg from folder *c:\My Project* **PlaySound("c:\My Project\MyMusic.ogg") **Play WAVE sound file tada.wav from Window's media folder: **PlaySound("<Windows>\Media\tada.wav")** **Play MIDI sound file Autumn.mid from <SrcDir> folder: **PlaySound("<SrcDir>\Autumn.mid")**

Additional Info
Use **CBK contants** to retrieve info on opened sound file.

StopSound()

Description
Stops sound playback. This command not only stops playback of all audio files but it also unload the files from memory. No parameters required.

Code Example
Stop sound playback: **StopSound()

Additional Info

This command stops playback of all sound files, including background sound (if used).

AudioOpen("Path")

Description

Opens and plays OGG, WMA or ASF sound file specified as a command parameter.

Sound file parameter should contain either

a) folder path + sound file name, or
b) **path macro** + sound file name

Code Examples

```
**Play ogg sound file MyMusic.ogg from folder c:\My Project
AudioOpen("c:\My Project\MyMusic.ogg")

**Play WMA sound file FourSeasons.wma from <SrcDir> folder:
AudioOpen("<SrcDir>\FourSeasons.wma")
```

Additional Info

Use **CBK contants** to retrieve info on opened sound file.

If file has not been specified as a command parameter, MMB will display Open File dialog box. Multiple file selections are enabled (using CTRL and SHIFT keys) - first selected file will be played, others are stored in MMB's internal Song List and are retrievable through the <List> constant.

AudioPlay()

Description

Plays OGG, WMA or ASF sound file specified using AudioOpen command.

Code Example	

```
**Play opened sound
AudioPlay()
```

Additional Info	

Use **CBK contants** to retrieve info on played sound file.

AudioStop()

Description	

Stops playback of sound opened and played by either AudioOpen or AudioPlay command.

No parameters required.

Code Example	

```
**Stop sound playback:
AudioStop()
```

AudioPause()

Description	

Pauses playback of sound opened and played by either AudioOpen or AudioPlay command.

No parameters required.

Code Example	

```
**Pause sound playback:
AudioPause()
```

AudioRewind("Time","Parameters")

Description

Rewinds sound opened and played by either AudioOpen or AudioPlay command to a given position (in seconds).

First command parameter sets number of seconds for rewinding.

Second command parameter is optional, used for relative rewinding, and to use it set:

RELATIVE

...as a second command parameter.

Code Examples

```
**Rewind audio file to 125th second
AudioRewind("125","")

**Rewind audio file for 20 seconds using relative rewinding
AudioRewind("20","RELATIVE")
```

Additional Info

Learn about Matrix object to create a seek bar.

BackgroundPlay("Path","Parameters")

Description

Opens and plays any type of supported sound file, specified as command parameter, as background sound.

First command parameter requires either

a) folder path + sound file name, or
b) **path macro** + sound file name

Second command parameter is optional and with it you can loop (automatically repeat) playback of specified sound file. Parameter is:

LOOP

Code Examples

**Background play ogg sound file MyMusic.ogg from <SrcDir> folder
```
BackgroundPlay("<SrcDir>\MyMusic.ogg","")
```

**Background play WAV sound file Beat.wav from <CD> folder in loop
```
BackgroundPlay("<CD>\Beat.wav","LOOP")
```

Additional Info

Use **CBK contants** to retrieve info on played sound file.

BackgroundPause()

Description

Pauses playback of sound opened and played by BackgroundPlay command.

No parameters required.

Code Example

**Pause background sound playback:
```
BackgroundPause()
```

BackgroundStop()

Description
Stops playback of sound opened and played by BackgroundPlay command. No parameters required.

Code Example
Stop background sound playback: **BackgroundStop()

WavePlay("Path","Parameters")

Description
Opens and plays WAVE sound file specified as a command parameter. First command parameter requires either a) folder path + sound file name, or b) **path macro** + sound file name Second command parameter is optional and with it you can loop (automatically repeat) playback of specified sound file. Parameter is: LOOP

Code Examples
Play file MyMusic.wav from folder *c:\My Project* **WavePlay("c:\My Project\MyMusic.wav") **Play file FourSeasons.wav from <SrcDir> folder with looping: **WavePlay("<SrcDir>\FourSeasons.wav","LOOP")** **In case of sound file is embedded, don't use the full path..just name of embedded file defined in the "**Embedded Sounds**" dialog: **WavePlay("FourSeasons","")**

Additional Info
Use **CBK contants** to retrieve info on opened sound file. If file has not been specified as command parameter, MMB will display Open File dialog box. Multiple file selections are enabled (using CTRL and SHIFT keys) - first

selected file will be played, others are stored in MMB's internal Song List and are retrievable through the <List> constant.

WaveStop()

Description	

Stops playback of sound opened and played by WavePlay command.

No parameters required.

Code Example	

**Stop wave sound playback:
```
WaveStop()
```

MidiPlay("Path","Parameters")

Description	

Opens and plays MIDI sound file specified as command parameter.

First command parameter requires either

a) folder path + sound file name, or
b) **path macro** + sound file name

Second command parameter is optional and with it you can loop (automatically repeat) playback of specified sound file. Parameter is:

LOOP

Code Examples	

**Play file MyMusic.mid from folder *c:\My Project*
```
MidiPlay("c:\My Project\MyMusic.mid")
```

**Play file FourSeasons.mid from <SrcDir> folder with looping:
```
MidiPlay("<SrcDir>\FourSeasons.mid","LOOP")
```

Additional Info	

Use **CBK contants** to retrieve info on opened sound file.

If file has not been specified as command parameter, MMB will display Open File dialog box. Multiple file selections are enabled (using CTRL and SHIFT keys) - first selected file will be played, others are stored in MMB's internal Song List and are retrievable through the <List> constant.

MidiStop()

Description	

Stops playback of sound opened and played by MidiPlay command.

No parameters required.

Code Example	

```
**Stop midi sound playback:
MidiStop()
```

ModOpen("Path")

Description	

Opens MOD sound file specified as command parameter.

Parameter requires either

a) folder path + sound file name, or
b) **path macro** + sound file name

Code Examples	

```
**Open file MyMusic.mod from folder c:\My Project
ModOpen("c:\My Project\MyMusic.mod")

**Open file FourSeasons.mod from <SrcDir> folder
ModOpen(" <SrcDir>\FourSeasons.mod")
```

Additional Info	

Use **CBK contants** to retrieve info on opened sound file.

If file has not been specified as command parameter, MMB will display Open File dialog box. Multiple file selections are enabled (using CTRL and SHIFT keys) - first selected file will be played, others are stored in MMB's internal Song List and are retrievable through the <List> constant.

ModPlay()

Description	

Plays MOD file opened using ModOpen command.

No parameters required.

Code Example	

```
**Play MOD file:
ModPlay()
```

ModStop()

Description	

Stops playback of sound opened and played using ModOpen and ModPlay commands.

No parameters required.

Code Example	

```
**Stop MOD sound playback:
ModStop()
```

Audio CD Playback Commands

CDPlay()

Description
Starts playback of Audio CD from the start.

Code Example

```
**Play Audio CD from the start
CDPlay()
```

Additional Info
Use **CBK contants** to retrieve info about Audio CD.

CDStop()

Description
Stops playback of Audio CD.

Code Example

```
**Stop Audio CD playback
CDStop()
```

CDPause()

Description

Pause playback of current Audio CD track.

Code Example

```
**Pause Audio CD playback
CDPause()
```

CDTrack("Track")

Description

Plays Audio CD track specified using command parameter.

Parameter sets number of track, for example:

4

...will play Track 4.

Code Example

```
**Play Audio CD track no. 6
CDTrack("6")
```

Additional Info

Mixed-mode CD's use first track for data, so first audio track is Track 2.

CDForward()

Description
Plays next Audio CD track.

Code Example

```
**Play next CDDA track
CDForward()
```

CDBackward()

Description
Plays previous Audio CD track.

Code Example

```
**Play previous CDDA track
CDBackward()
```

CDPlayPause()

Description
Pauses or plays Audio CD. If playback has been paused/stopped, it will be resumed. Otherwise playback is paused. Useful when making one button for play/pause functions.

Code Example

```
**Play or pause Audio CD
CDPlayPause()
```

CDSkipForward()

Description
Sets position of currently played Audio CD track 10 seconds forward.

Code Example

```
**Forward current Audio CD track
CDSkipForward()
```

CDSkipBackward()

Description
Sets position of currently played Audio CD track 10 seconds backward.

Code Example

```
**Backward current Audio CD track
CDSkipBackward()
```

WhichCDTrack("TrackVar")

Description	

Retrieves number of currently played Audio CD track and puts result into numerical variable specified as command parameter.

Parameter sets numerical variable that will contain numer of played track.

For example:

```
TrackNo
```

...when put into command, it looks like this:

```
WhichCDTrack("TrackNo")
```

After performing code line above, numerical variable TrackNo will contain number of played audio track. You can now, for example, display it:

```
Message("Current Audio CD track:","TrackNo")
```

Code Example	

```
**Retrieve number of currently played Audio CD track and display result
**in text object labeled "AudioTrack":
WhichCDTrack("TrackNo")
DisplayValue("AudioTrack","TrackNo")
```

Additional Info	

Mixed-mode CD's use first track for data, so first audio track is Track 2.

Audio Visualization Commands

AudioVisualizationType("ObjectLabel","Type")

Description	

Sets type of Audio Visualization.

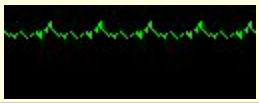
Uses two parameters - first parameter sets label of Audio Visualization object you want settings be applied to. For example,

```
AudioVis
```

Second parameter sets visualization type:

- ANALYZER

- OSCILLOSCOPE



Code Examples

**Set audio visualization object labeled MyVis to ANALYZER:
AudioVisualizationType("MyVis","ANALYZER")

**Set audio visualization object labeled AudioVis to
**OSCILLOSCOPE:
AudioVisualizationType("AudioVis","OSCILLOSCOPE")

Additional Info

Audio visualization is supported for OGG, WAV, XM, S3M, IT, MOD files.

AudioVisualizationColor("ObjectLabel","Parameters")

Description

Sets color of Audio Visualization elements.

First parameter sets label of Audio Visualization object you want settings be applied to. For example,

```
AudioVis
```

Second parameter sets visualization color.

Color setting uses RGB (Red, Green, Blue) system containing 3 components.

Each component can have one of 256 levels. Greater value of component increases color intensity. If you specify value 0, color component will not be used. Value 255 uses maximum color intensity.

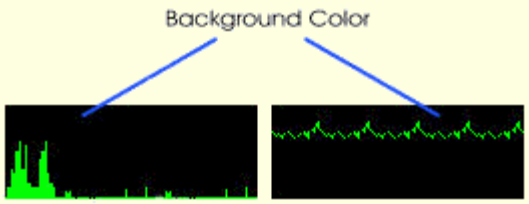
Color parameter for Audio Visualization is a comma-separated array of string values, each representing one RGB component (first: red, second: green, third:

blue) in value range 0-255.

R,G,B	Result
128,5,64	

R,G,B	Result
0,255,255	

Both Analyzer and Oscilloscope visualizations have background color



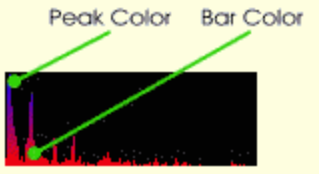
that can be changed using following parameter:

```
BACKGROUND=R,G,B
```

Put with numerical RGB values:

```
BACKGROUND= 150, 79, 205
```

Analyzer visualization uses two color settings:



First you set RGB values for bar color:

```
RGB(255,0,0)
```

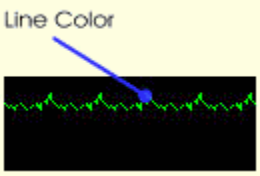
And after comma, add RGB values for peak color:

```
RGB(0,0,255)
```

Put together, second parameter for analyzer looks like this:

```
RGB(255,0,0),RGB(0,0,255)
```

Oscilloscope visualization uses only one (line) color setting:



Example of RGB values for oscilloscope line color:

```
RGB(255,0,0)
```

Code Examples

```
**Set background color of Audio Visualization object labeled AudioVis:
AudioVisualizationColor("AudioVis","BACKGROUND=150,79,205")

**Set Analyzer bar and peak colors of Audio Visualization object
**labeled AudioVis:
AudioVisualizationColor("AudioVis","255,0,0,0,0,255")

**Set Oscilloscope line color of Audio Visualization object labeled AudioVis:
AudioVisualizationColor("AudioVis","255,0,0")

** an advanced example with ColorPicker()
BGColor$=''
ColorPicker()
BGColor$=CBK_SelColor
If (BGColor$>'') Then
 AudioVisualizationColor("AudioVisualization","BACKGROUND=CBK_SelColor
")
End
```

It opens "Select Color" dialog and allow you to select a color, then replace the current BG color with new selected.

Additional Info

Audio visualization is supported for OGG, WAV, XM, S3M, IT, MOD files.

12.10.9 Image Commands

Main features of image-related commands are:

- replacing images in bitmap and hotspot objects
- viewing images in a separate window
- resizing, rotating, zooming of images

For examples of usage the below commands can be found in **test_image.mbd**, **ImageDemo.mbd** or **clock_3.mbd**

ViewImage("Path","Parameters")

Description

Displays external JPEG or BMP image in a separate window.

First parameter sets file to load and requires either

a) folder path + video file name, or
b) **path macro** + video file name

Second parameter is optional and can be used to center image window on the screen:

CENTER

Code Examples

```
**View JPEG file NewImage.jpg placed in folder c:\My Project
ViewImage("c:\My Project\NewImage.jpg","")

**View BMP file HotImage.bmp placed in <SrcDir> folder
**having window centered on screen
ViewImage("<SrcDir>\HotImage.bmp","CENTER")
```

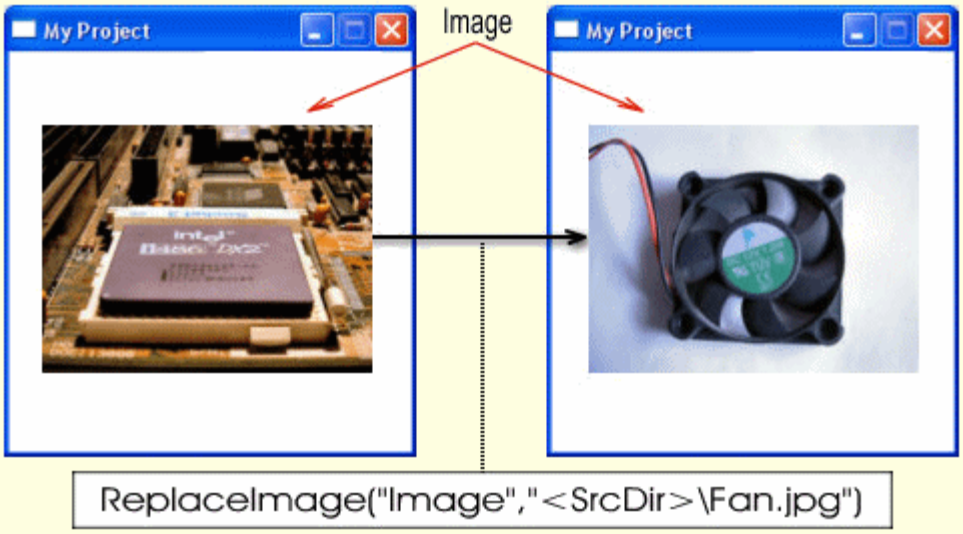
Additional Info

Position of image window is saved and will be used on the next window opening if second parameter (CENTER) is not set.

ReplaceImage("ObjectLabel","Path")

Description

Replaces current image of specified Bitmap, HotSpot or Polygonal HotSpot object with an external JPEG or BMP file.



Use first parameter to set Bitmap object label, for example:

```
MainImage
```

Second parameter sets file to load and requires either

a) folder path + video file name, or
b) **path macro** + video file name

Code Examples

```
**Replace image in bitmap object labeled MainImage with JPEG file
**NewImage.jpg placed in folder c:\My Project
ReplaceImage("MainImage","c:\My Project\NewImage.jpg")

**Set image in hotspot object labeled ColdSpot from BMP file
**HotImage.bmp placed in <SrcDir> folder
ReplaceImage("ColdSpot","<SrcDir>\HotImage.bmp")
```

Additional Info

Using this command without second parameter will clear image in the specified object:

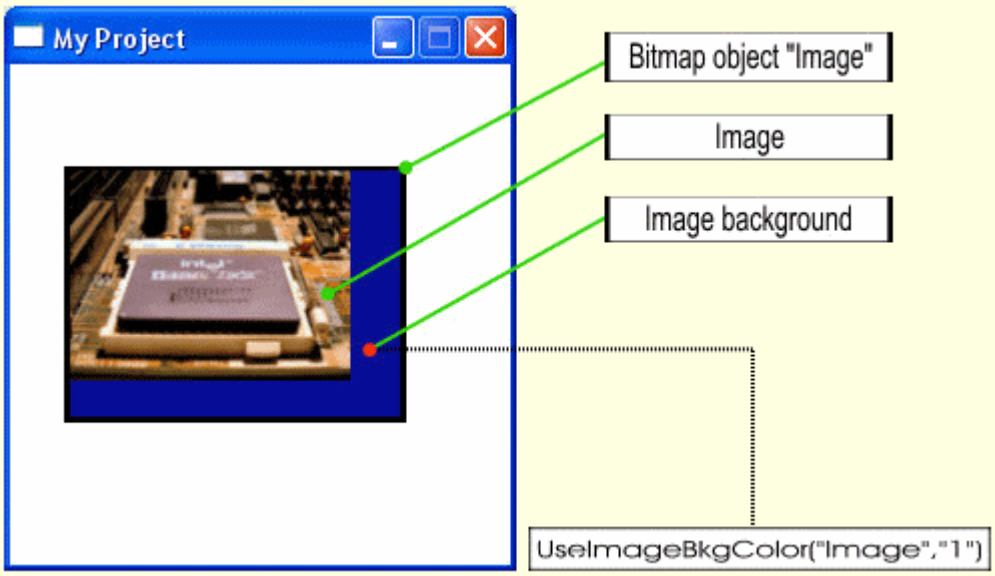
```
ReplaceImage("FacePicture","")
```

Previously (in 4.9 or older) ReplaceImage used on HotSpot resize the HotSpot object to Image dimensions. But ReplaceImage in 4.9.5 keep the HotSpot width/height unchaged and the Image is fit to the HotSpot window dimensions. If you want to change hotspot window to the image dimensions, use **RestoreImage** ("HotSpot") right after ReplaceImage command.

UseImageBkgColor("ObjectLabel","Parameters")

Description

Sets status of background color under Bitmap object. If used, Bitmap object is filled with background color before displaying image.



Use first parameter to set Bitmap object label, for example:

```
MyImage
```

Second command parameter enables or disables image back color:

- `1` : sets image background color on
- `0` : sets image background color off

Code Examples	

```
**Set image background color on for Bitmap object "MyImage":
UseImageBkgColor( "MyImage","1")

**Set image background color off for Bitmap object "MyImage":
UseImageBkgColor( "MyImage","0")
```

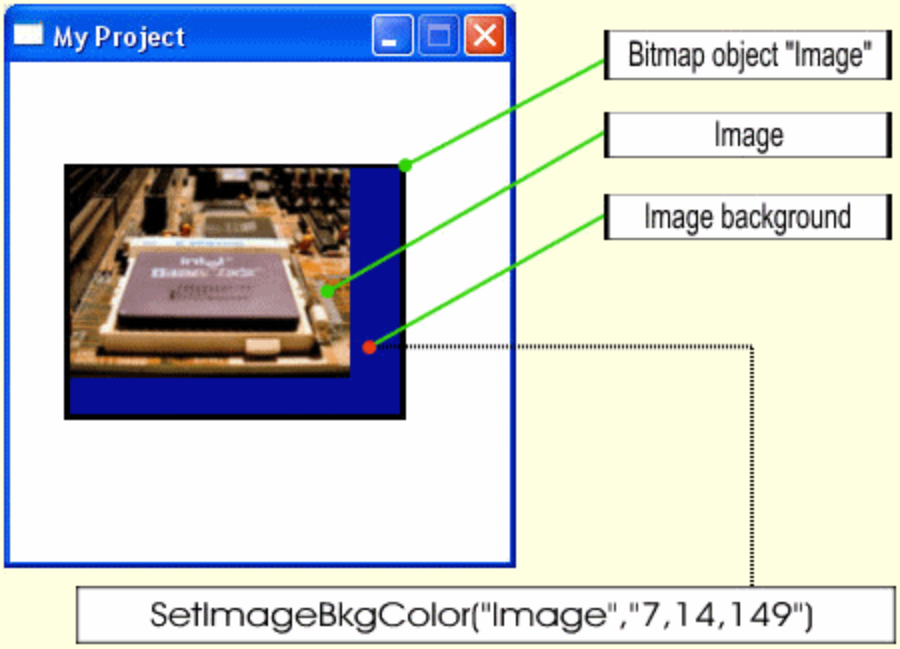
Additional Info	

Use SetImgBkgColor command to set image background color.

SetImageBkgColor("ObjectLabel","R,G,B")

Description	

Sets image background color.



Use first parameter to set Bitmap object label, for example:

```
BigImage
```

Color setting uses RGB (Red, Green, Blue) system containing 3 components.

Each component can have one of 256 levels. Greater value of component increases color intensity. If you specify value 0, color component will not be used. Value 255 uses maximum color intensity.

Second parameter for SetImageBkgColor is a comma-separated array of string values, each representing one RGB component (first: red, second: green, third: blue) in value range 0-255.

R,G,B	Result
128,5,64	

R,G,B	Result
0,255,255	

Example of second parameter:

```
150, 79, 205
```

Put together, command with parameters looks like this:

```
SetImageBkgColor("BigImage"," 150, 79, 205")
```

You can also use string variable for color setting:

```
color$=' 150, 79, 205'
SetImageBkgColor("BigImage","color$")
```

Code Examples

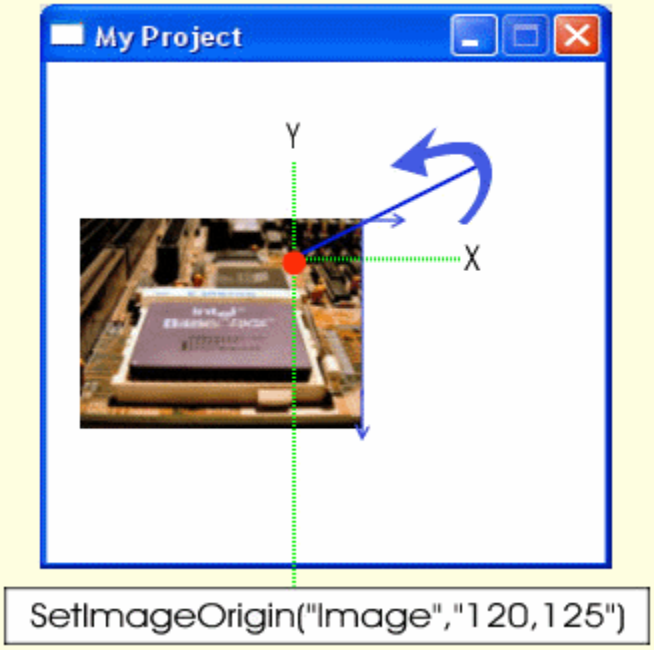
```
**Set blue background color for image labeled "RearImage":
SetImageBkgColor(" RearImage ","0,0,255")

**Set cyan background color for image labeled "RearImage" using
**string variable as a color source:
MyColor$='0,255,255'
SetImageBkgColor(" RearImage ","MyColor$")
```

SetImageOrigin("ObjectLabel","X,Y")

Description

Sets starting point (origin), relative to the top left image corner, that will be used for image resizing and rotation.

Both resizing and rotation will be performed around this point.



Use first parameter to set Bitmap object label, for example:

```
RotoImage
```

Second command parameter consists of two coordinates.

Horizontal (X) coordinate, for e.g.:

```
80
```

Vertical (Y) coordinate, for e.g.:

```
30
```

Coordinates in second parameter are separated with commas, so here's an example of complete command:

```
SetImageOrigin("RotoImage","80,30")
```

Code Example

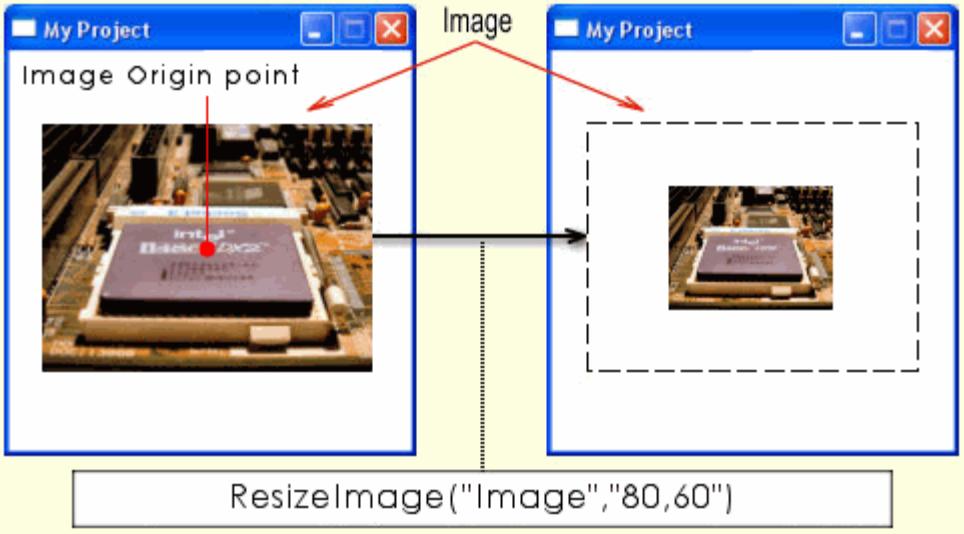
```
**Set starting point for object "RotoImage" to 154 (X), 84 (Y):
SetImageOrigin( "RotoImage","154,84")
```

ResizeImage("ObjectLabel","W,H")

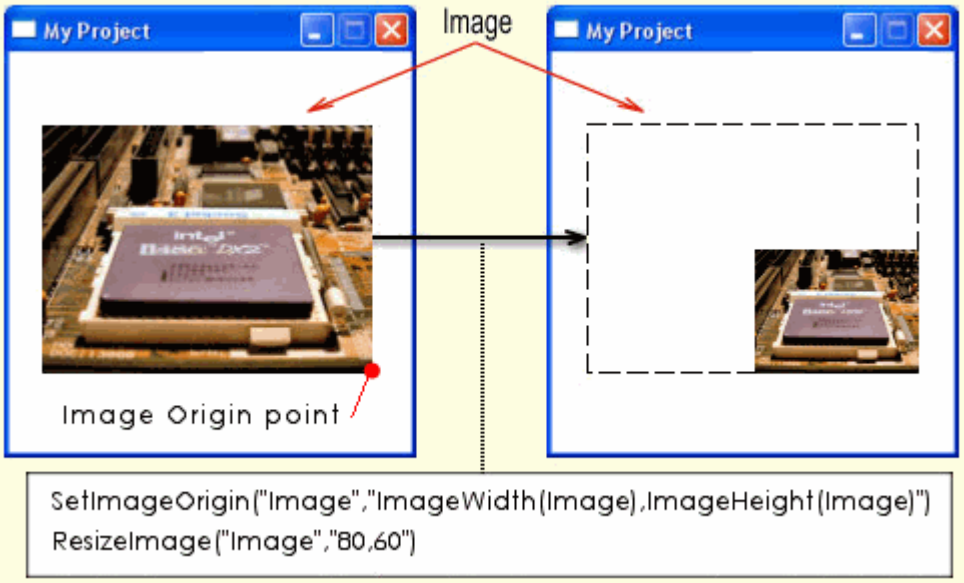
Description

Resize the Image Object to the new size specified using the origin as the transformation center.

This is a simple example of resizing Image with default origin point:



And the example here shows you how to resize image using the image Origin set to right-bottom corner of the original image.



Use first parameter to set Bitmap object label, for example:

```
AlterImage
```

Second command parameter consists of two coordinates.

Horizontal (W) coordinate, for e.g.:

```
80
```

Vertical (H) coordinate, for e.g.:

```
60
```

Coordinates in second parameter are separated with commas, so here's an example of complete command:

```
ResizeImage("Bitmap","120,90")
```

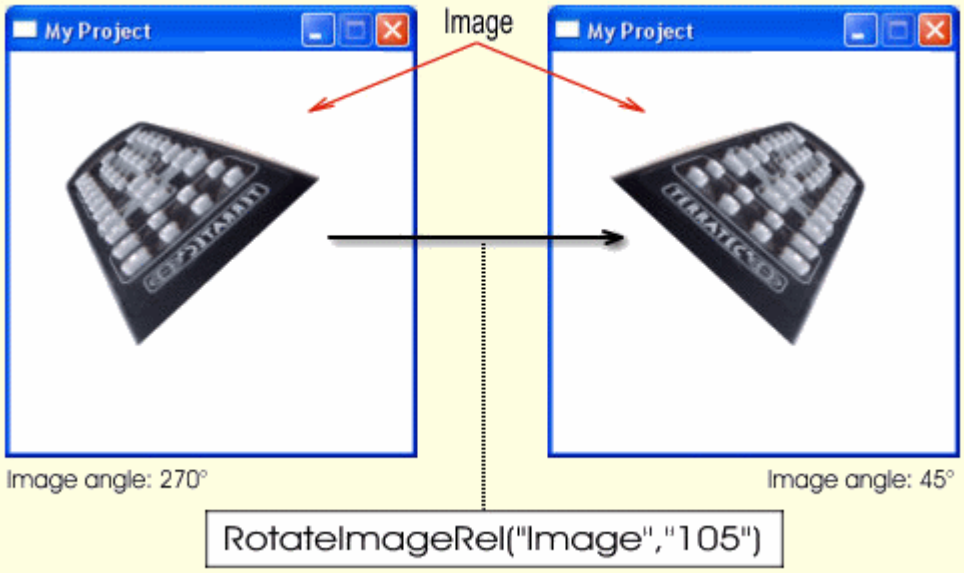
Code Example

```
**Resize Bitmap object image "AlterImage":
ResizeImage("AlterImage","80,30")

**Set image Origin to Right-Bottom corner and then resize image "AlterImage" to half size of the original image:
IW=ImageWidth(AlterImage)
IH=ImageHeight(AlterImage)
SetImageOrigin("AlterImage","IW,IH")
ResizeImage("AlterImage","IW/2,IH/2")
```

RotateImageRel("ObjectLabel","Angle")

Description

Rotates Bitmap object image relatively from current angle.



Use first parameter to set Bitmap object label, for example:

`RotoImage`

Second command parameter sets relative rotation angle in degrees.

Value in degrees can be either positive or negative. For e.g.:

`80` - rotates image 80 degrees clockwise

`-80` - rotates image 80 degrees counterclockwise

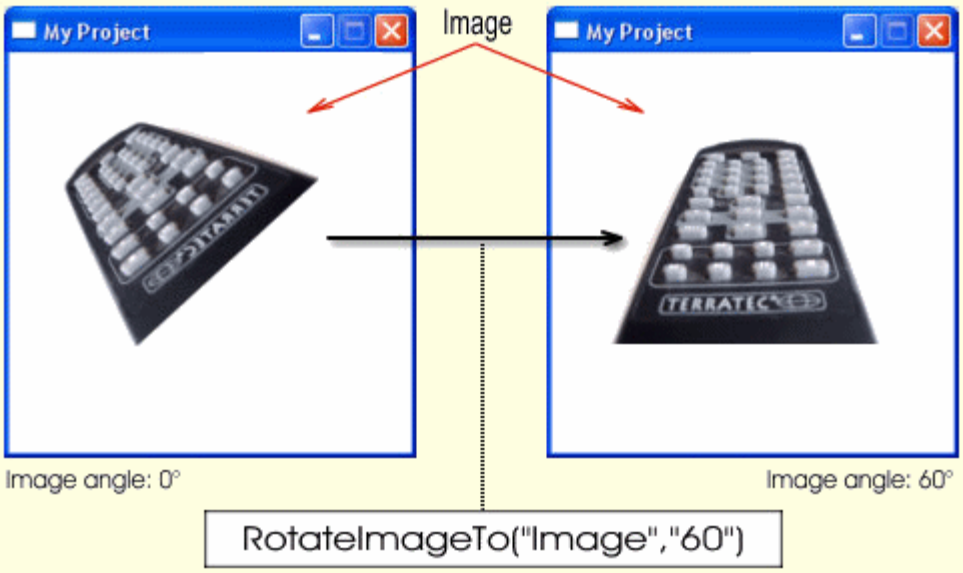
Code Examples

```
**Relatively Rotate image in Bitmap object "RotoImage"
**45 degrees clockwise
RotateImageRel( "RotoImage","45")

**Relatively Rotate image in Bitmap object "RotoImage"
**45 degrees counterclockwise
RotateImageRel( "RotoImage","-45")
```

RotateImageTo("ObjectLabel","Angle")

Description

Performs absolute rotation of Bitmap object image in degrees, starting from 0 (default object angle).



Use first parameter to set Bitmap object label, for example:

`RotoImage`

Second command parameter sets absolute rotation angle in degrees.

Value in degrees can be either positive or negative. For e.g.:

`80` - rotates image 80 degrees clockwise

`-80` - rotates image 80 degrees counterclockwise

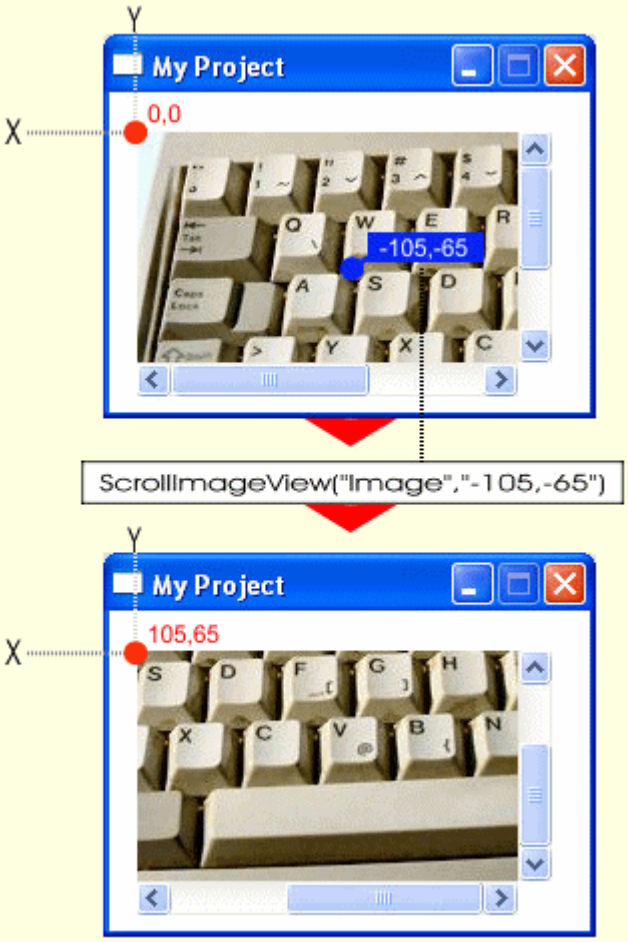
Code Examples

```
**Perform absolute rotation of image in Bitmap object "RotoImage"
**45 degrees clockwise
RotateImageTo( "RotoImage","45")

**Perform absolute rotation of image in Bitmap object "RotoImage"
**45 degrees counterclockwise
RotateImageTo( "RotoImage","-45")
```

ScrollImageView("ObjectLabel","X,Y")

Description	

Sets displayed part of Bitmap object image using coordinates from the second command parameter. This can be used when image has a different width or height than Bitmap object canvas.



Use first parameter to set Bitmap object label, for example:

```
BigImage
```

Second command parameter sets image section upper left coordinates.

When image is smaller than Bitmap object canvas, use positive coordinates to move displayed section to left (x) and up (y).

Positive horizontal (X) coordinate, for e.g.:

```
80
```

Positive vertical (Y) coordinate, for e.g.:

```
30
```

When image is larger than Bitmap object canvas, use negative coordinates to move displayed section to right (x) and down (y).

Negative horizontal (X) coordinate, for e.g.:

```
-80
```

Negative vertical (Y) coordinate, for e.g.:

```
-30
```

Coordinates in second parameter are separated with commas, so here's an example of complete command:

```
ScrollImageView ("BigImage","-80,-30")
```

Code Examples

```
**Display section of Bitmap object "BigImage" starting with coordinates
**-130 (X), -80 (Y)
ScrollImageView( "BigImage","-130,-80")

**Display section of Bitmap object "SmallImage" starting with coordinates
**30 (X), 80 (Y)
ScrollImageView( "SmallImage","30,80")
```

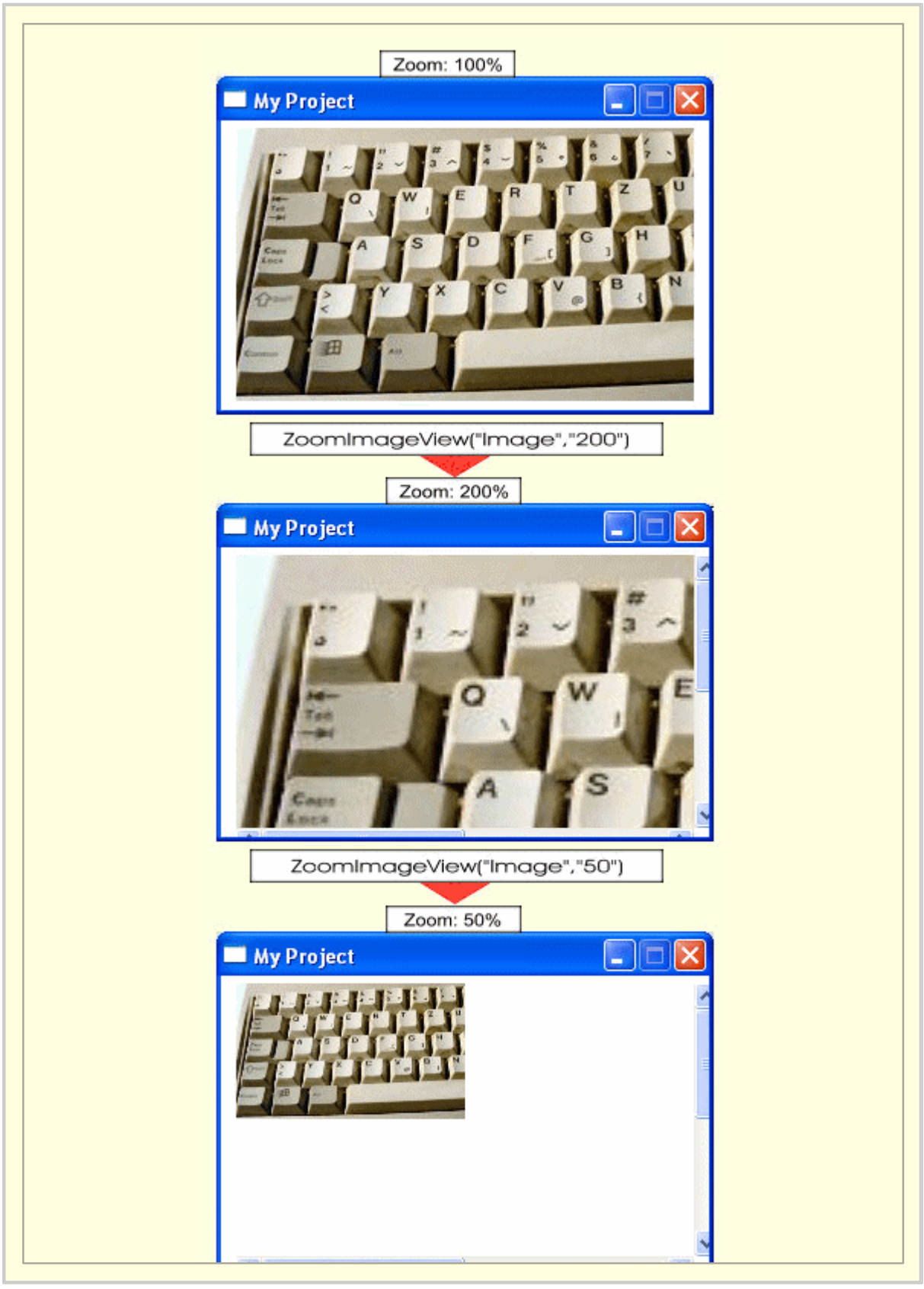
Additional Info

To obtain actual X/Y position of the left top corner of an image inside the Image object use **ImageScrollX**, **ImageScrollY** functions.

ZoomImageView("ObjectLabel","Zoom")

Description	

Zooms in or out image of specified Bitmap object.



Use first parameter to set Bitmap object label, for example:

```
MapImage
```

Second command parameter sets zoom percentage. Default image zoom value is 100. You can use both lower and higher values than default one.

Zooming out to 80% of image original size:

```
80
```

Zooming in to 150% of image original size:

```
150
```

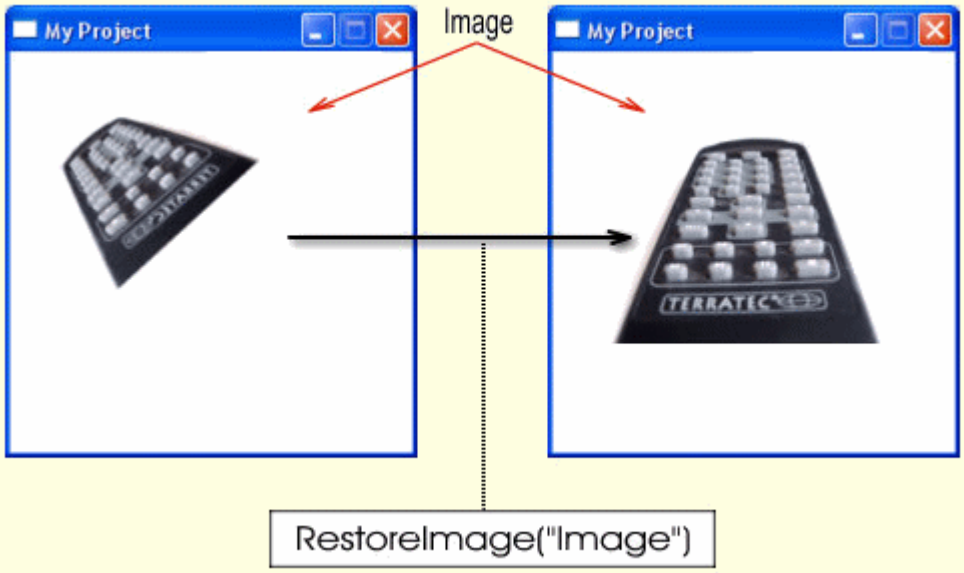
Code Examples

```
**Zoom out Bitmap object image "MapImage" to 50% of original:
ZoomImageView( "MapImage","50")

**Zoom in Bitmap object image "MapImage" to 200% of original:
ZoomImageView( "MapImage","200")
```

RestoreImage("ObjectLabel")

Description

Resets size and rotation angle of specified Bitmap object.



Use first parameter to set Bitmap object label, for example:

```
AlterImage
```

Warning! RestoreImage will not work if you remove unused graphics from the project with the "File>>> Reduce Size..." option. It simply remove all backup copies of the images created within the project creation process.

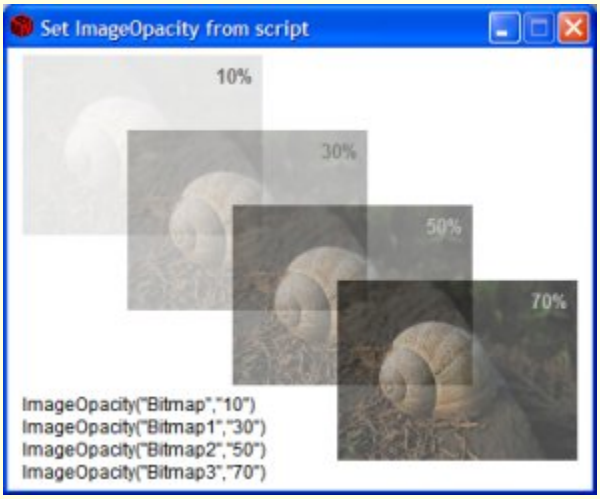
Code Example	

```
**Restore Bitmap object image "AlterImage":
RestoreImage("AlterImage")
```

ImageOpacity("OpacityLevel")

Description	

Sets the image (Bitmap object) opacity in range 0-100.



Set the OpacityLevel = 0 if you want fully transparent Bitmap object and OpacityLevel = 100 to make the object fully opaque.

Code Example

```
**Set Bitmap object image 50% transparent:
ImageOpacity("50")

**Set Bitmap object image fully transparent:
level=0
ImageOpacity("level")
```

12.10.1 Print Commands

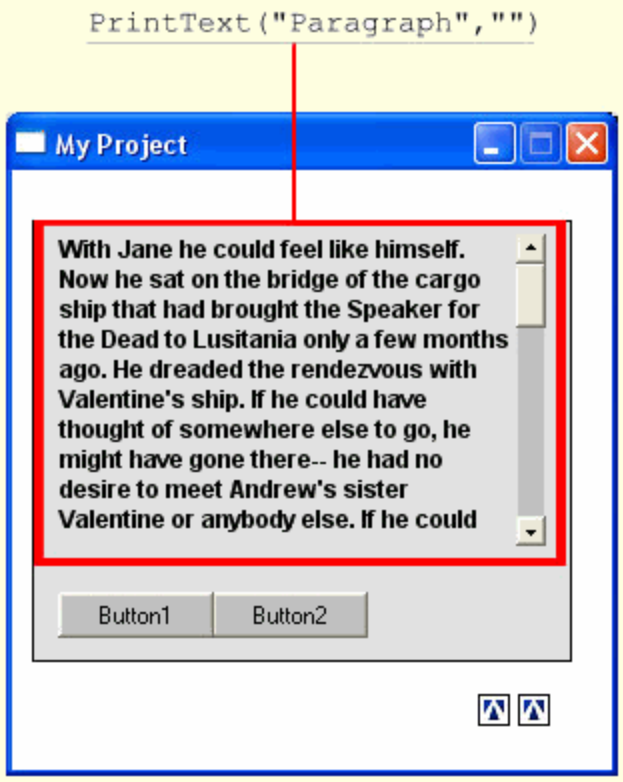
There are three commands in MMB that deal with printing:

- PrintText - prints text from some object
- PrintPage - prints entire MMB project page
- PrintRect - prints everything inside specified rectangle

PrintText("ObjectLabel","Parameters")

Description

Prints content of specified text object.



Use first parameter to set label of text object you want to print text from. For example:

Paragraph

Second parameter is optional. By default, text is printed using default font style and size. Using this parameter:

FONT_FROM_OBJECT

...text will be printed using original font style and size of text object.

Code Examples

```
** Print text from object labeled "StoryBox" using default text style:
PrintText("StoryBox","")

** Print text from object labeled "Paragraph" using object's text style:
PrintText("StoryBox","FONT_FROM_OBJECT")
```

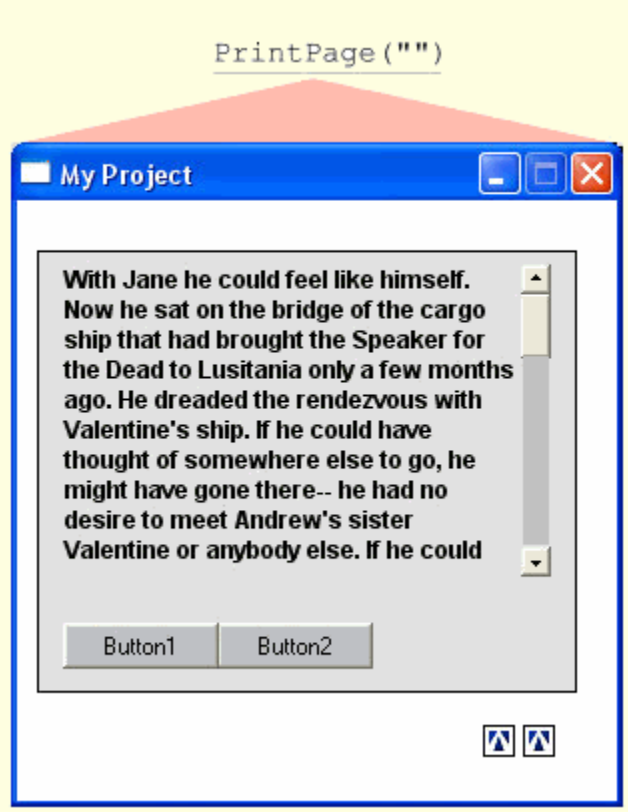
Additional Info

This command prints text from MMB's Text and Paragraph objects.

PrintPage("Page")

Description

Prints content of currently displayed project page.



There is one available parameter, and it's optional. If you leave it empty or set:

100

...page will be printed with full fit to page.

Setting lower values, like

80

...will print page at 80% of width of the paper.

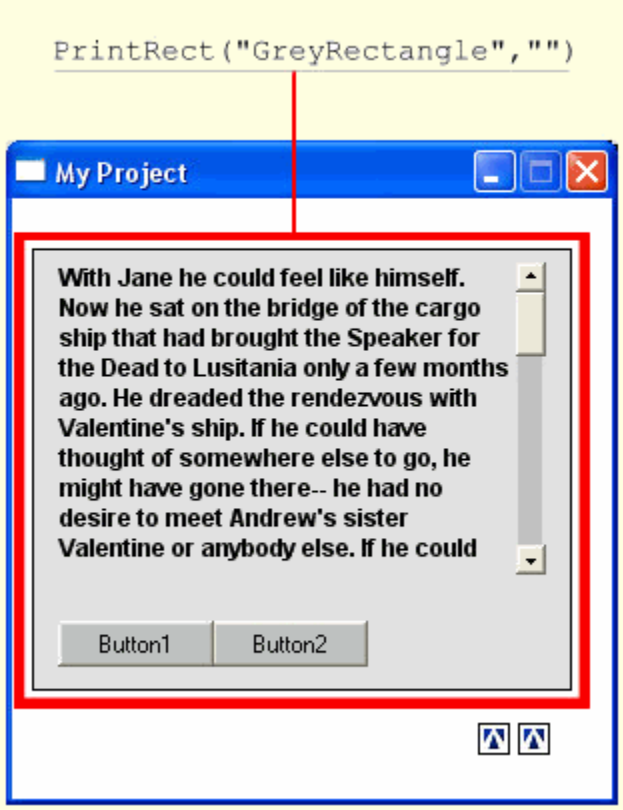
Code Examples

```
** Print current page with full fit:
PrintPage("")

** Print current page at 60% of paper width:
PrintPage("60")
```

PrintRect("ObjectLabel","Zoom")

Description

Prints content inside specified rectangle object.



First parameter sets label of rectangle object that is being used as a contour (border) for print content. For example:

GreyRectangle

Second parameter is optional. If you leave it empty or set:

100

...page will be printed with full fit to page.

Setting lower values, like

80

...will print page at 80% of width of the paper.

Code Examples

```
** Print everything inside of RectForPrint:
PrintRect("RectForPrint","")

** Print everything inside of PrintBox at 90% of paper width:
PrintRect("PrintBox","90")
```

Additional Info

Either shape of rectangle object or it's visibility is irrelevant for this command, so you can have hidden square work for PrintRect too.

12.10.1 Video Commands

MMB comes with video playback support, and main features are:

- Video playback of following formats:
  AVI, QT, MOV, MPG, MPEG, M1V, ASX, ASF, WMV
- Video positioning
- Setting of video speed and size

Supported Video scripting commands:

VideoLoad("ObjectLabel","Path")

Description
Opens video file (create still image) specified as a command parameter. Use first parameter to set video object label, for example: `VideoBox` Second parameter sets file to load and requires either a) folder path + video file name, or b) **path macro** + video file name

Code Examples
Play file Intro.avi from folder *c:\My Project* in video object **labeled "VideoBox" **VideoLoad("VideoBox","c:\My Project\Intro.avi") **Play file Intro.avi from <SrcDir> folder in video object **labeled "VideoBox" **VideoLoad("VideoBox","<SrcDir>\Intro.avi")**

Additional Info
Use **CBK contants** to retrieve info on opened video file. If file has not been specified as a command parameter, MMB will display Open File dialog box. If VideoLoad refuse to create a Still Image from the loaded video, try increase the Color Depth in Display settings. With VideoLoad you can not only load Video files, but also some audio formats! For deatiled list of supported audio formats check **this page**.

VideoClose("ObjectLabel")

Description

Close Video (unload it from memory) and Hide the video object (Hide still image of video). The same functionality as ***Close&Hide*** option in Video Properties (After Video Finish event).

Use first parameter to set video object label, for example:

```
VideoBox
```

Code Examples

```
**Close video and hide the still image.
VideoClose("Video")
```

VideoPlay("ObjectLabel")

Description

Plays video file in a video object specified as a command parameter.

If video object is labeled:

```
VideoWindow
```

...then it's label put into command will look like this:

```
VideoPlay("VideoWindow")
```

Code Example

```
**Play video file loaded in object labeled "VideoWindow":
VideoPlay("VideoWindow")
```

Additional Info

Use **CBK contants** to retrieve info on opened video file.

Prior to playing, video file must be loaded using VideoLoad command.

VideoPause("ObjectLabel")

Description

Pauses or plays video.

If playback has been paused/stopped, it will be resumed. Otherwise playback is paused.

Useful when making one button for play/pause functions.

Object that video should be paused for is specified as a command parameter. If video object is labeled:

```
VideoWindow
```

...then it's label put into command will look like this:

```
VideoPause("VideoWindow")
```

Code Example

```
**Pause/resume video in object labeled "VideoWindow":
VideoPause("VideoWindow")
```

VideoStop("ObjectLabel")

Description

Stops video in object specified as a command parameter.

If video object is labeled:

```
VideoBox
```

...then it's label put into command will look like this:

```
VideoStop("VideoBox")
```

Code Example

```
**Stop video in object labeled "VideoBox":
VideoStop("VideoBox")
```

VideoRewind("ObjectLabel","Parameters")

Description

Rewinds video, opened and played by VideoOpen and VideoPlay commands, to a given position.

Position format depends on settings of VideoParam command, so positioning can be performed either in milliseconds or frames.

Use first parameter to set video object label, for example:

```
VideoBox
```

Second command parameter sets video position in frames or milliseconds:

```
3214
```

To use relative rewinding set:

```
RELATIVE
```

...as a third command parameter. In this case, second parameter makes offset positioning, so for example moving video 120 frames forward is represented as positive number:

```
120
```

...and moving video 120 frames backwards is represented as negative number:

```
-120
```

Notice that parameters in this command are not separated with double quotes, but with commas, so here's example of complete command:

```
VideoRewind("VideoBox","-120,RELATIVE")
```

Video positioning is not available for all video formats.

Code Examples

```
**Set video position in object "VideoBox" to 25th second:
**(VideoParam Mode must be set to TIME)
```

```
VideoRewind("VideoBox","25000")

**Set video position in object "VideoBox" to a frame 7328:
**(VideoParam Mode must be set to FRAME)
VideoRewind("VideoBox","7328")

**Set relative video position in object "VideoBox" forward for 5 seconds:
**(VideoParam Mode must be set to TIME)
VideoRewind("VideoBox","7328,RELATIVE")

**Set relative video position in object "VideoBox" backward for
**180 frames (VideoParam Mode must be set to TIME)
VideoRewind("VideoBox","-180 ,RELATIVE")
```

Additional Info

Learn about Matrix object to create a video seek bar.

VideoSpeed("ObjectLabel","Parameters")

Description

Sets playback speed of video opened by VideoOpen command.

Use first parameter to set video object label, for example:

VideoBox

Second command parameter sets video speed either as a fixed parameter or through numerical variable, in range:

0 - 2000

Default playback rate is 1000. Setting rate below default will lower playback rate, raising it above default will set higher playback rate.

To use relative setting of playback rate set:

RELATIVE

...as a third command parameter. In this case, second parameter makes setting of playback rate relative to current rate, so for example raising playback rate for 200 is represented as a positive number:

```
200
```

...and lowering playback rate for 200 is represented as negative number:

```
-200
```

Notice that parameters in this command are not separated with double quotes, but with commas, so here's example of complete command:

```
VideoSpeed("VideoBox","-200,RELATIVE")
```

Changing of playback rate is not available for all video formats.

Code Examples

```
**Set video playback rate in object "VideoBox" to 1500:
VideoSpeed("VideoBox","1500")

**Raise relative video playback rate in object "VideoBox" for 200:
VideoSpeed("VideoBox","200,RELATIVE")

**Lower relative video playback rate in object "VideoBox" for 200:
VideoSpeed("VideoBox","-200,RELATIVE")
```

Additional Info

Learn about Matrix object to create a playback rate gauge.

VideoScale("ObjectLabel","Width,Height")

Description

Sets video object size.

Use first parameter to set video object label, for example:

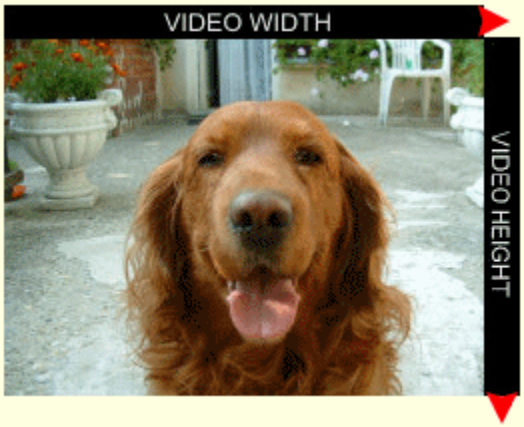
```
VideoBox
```

Second command parameter sets **width** of video object, for e.g.:

```
640
```

Third command parameter sets **height** of video object, for e.g.:

```
480
```



Notice that parameters in this command are not separated with double quotes, but with commas, so here's an example of complete command:

```
VideoScale("VideoBox","640,480")
```

Code Example

```
**Set video in object "VideoBox" to 320 (width) x 240 (height):
VideoScale("VideoBox","320,240")
```

VideoParam("ObjectLabel","Parameters")

Description

Sets additional parameters for video object.

Use first parameter to set video object label, for example:

`VideoBox`

Second command parameter sets following video object properties:

`FULLSCREEN` - uses ON/OFF switch to set FullScreen mode of video:

- `FULLSCREEN=ON` : sets video FullScreen on
- `FULLSCREEN=OFF` : sets video FullScreen off

`LOOP` - uses ON/OFF switch to set repeating of video:

- `LOOP=ON` : sets video looping on
- `LOOP=OFF` : sets video looping off

MUTE - uses ON/OFF switch to set sound mute for video object:

- MUTE=ON : sets audio muting on
- MUTE=OFF : sets audio muting off

Set the video either to time or frame format:

- TIME : sets time mode
- FRAME : sets frame mode

Code Examples

**Set video in object "VideoBox" to full screen:
VideoParam("VideoBox","FULLSCREEN=ON")

**Turn looping on for video in object "VideoBox":
VideoParam("VideoBox","LOOP=ON")

**Turn muting on for video in object "VideoBox":
VideoParam("VideoBox","MUTE=ON")

**Set frame mode for video in object "VideoBox":
VideoParam("VideoBox","FRAME")

Additional Info

CBK_VTime returns current time only if VideoParam is set to TIME and VFrame returns current frame only if VideoParam is set to FRAME. You can't use FRAME and TIME at the same time!

VideoParam is by default set to TIME.

If CBK_VTotalFrames returns 0 after loading the video, it means that video doesn't support frames so you won't be able to position video by frames using Video Skip FW/BW commands (in "Interaction with other objects") or using VideoRewind command (with pre-defined **VideoParam**("VideoBox","FRAME") settings).

12.10.12 MCI Commands

MCIObject("ObjectLabel","Command")

Description
Sends a command to MCI Object. The possible commands are: Play Pause Stop Close

Code Example

```
** Start playing MCI object
MCIObject("MCI_Object","Play")

** Pause MCI object playback
MCIObject("MCI_Object","Play")
```

MCICommand("MCIString")

Description
This will send a string to a device. The string will tell the device what to do. With this command you can access the devices for playing MPEG, video discs, record sound, play animation etc.. You can use <SrcDir>, <SrcDrive> or <CD> in the string. Also use <This> with the parent command to tell the device the MMB window will be the parent. For detailed information about the possible MCI strings, please, refer to the **Mcistr.hlp** documentation included in MMB root folder.

Code Example

Example: Here is a small sample to play MPG movie inside the mbd project in the position (100,50,100,100):

```
MCICommand("open <SrcDir>\sample.mpg alias MPEG style child parent <This>")
MCICommand("put MPEG window at 100 50 200 200")
MCICommand("window MPEG state hide")
```

```
    MCICommand("play MPEG")

Example:(Open and close door on CD (beer holder))
    MCICommand("set cdaudio door open")
    MCICommand("set cdaudio door closed"
```

Some of the MCICommand functions can return a value (for example number of tracks on CD)
The variable **MCIResult** will have the result (integer) number.

```
Example: (returns number of tracks on CD)
    MCICommand("status cdaudio number of tracks")
    DisplayValue("Text","MCIResult")


Example2: (returns length (in seconds) of track 1)
    MCICommand("status cdaudio length track 1")
    DisplayValue("Text_totalsec","MCIResult")


Example3: (returns status of CD audio)
MCICommand("status cdaudio mode")
If (MCIResult=-2) Then
   Show("Playing")
End

If (MCIResult=-1) Then
   Show("Stopped")
End
```

For a complete understanding and the syntax, consult Microsoft Documentation or **Mcistr.hlp** already installed in MMB folder.

WARNING! MCICommand accessing the device directly – be prepare for crashing if you are going to experiment!

NOTE: MCI device has a path-length limit of 128 characters! If your media files are buried inside the nested subfolders and the path is longer than 128 characters, your media won't play.

Also, if the path has some spaces inside, don't forget to enclose the path inside the double quotes like in this example...

```
file$='<SrcDir>\sample.mpg'
open$='open "' + file$ + '" alias MPEG style child parent <This>' MCICommand
("open$")
```

12.10.13 Flash Commands

Flash("ObjectLabel","Command/Path")

Description

Specify a command you want to execute in Flash player.

Possible commands:
PLAY - play the Flash movie.
STOP - stop the Flash movie.
LOOP - loop the Flash movie.
SHOWMENU - show the Flash movie rclick menu.
HIDEMENU - hide the Flash movie rclick menu.
MINMENU - show the minimalized Flash movie rclick menu.
FULLMENU - show the full Flash movie rclick menu.
BACK - jump to the previous frame of the Flash movie.
FORWARD - jump to the next frame of the Flash movie.
REWIND - rewind the Flash movie.

Anything else in EVENT parameter (including string variables) will be opened as a file or url flash movie.

Code Example

This small example will show the OpenFile dialog (with predefined mask) and after your selection will load and run the selected Flash movie:

```
OpenFile("Flash Files (*.swf)|*.swf|All Files|*.*||","*.swf")
If (OpenFile$ <> '') Then
  Flash("Flash","OpenFile$")
  Flash("Flash","PLAY")
End
```

For more examples check the **flash_examples.mbd** or **flash_menu_on_master_page.mbd** files or some more **here**.

Additional Info

MINMENU and FULLMENU is useless in a combination with HIDEMENU option.

FlashGetVar("ObjectLabel","FlashVar/MMBVar")

Description

Get variable from flash movie defined by name, and store it in mmb variable

Note: Variable must exist in Flash movie prior to calling it. Because the Flash actions are called asynchronously, then the result of the actions can be returned at an indeterminate time. Therefore, if you are not sure, if the called variable in the Flash movie already exist (or is filled), then use **Pause**() command in your MMB script to pause the MMB prior to calling the **FlashGetVar** or **FlashSetVar**. This should give the Flash movie a sufficient time to completing the running tasks and filling the variable that you want to get from Flash movie. Another way is calling the MMB commands directly from Flash. However, this requires a direct access to Flash movie source file (*.fla) and recompiling the Flash movie.

Code Example

This small example is taken from **flash_examples.mbd** file that is available in Samples\Flash_Examples folder (with all Flash sources). It will show you a real example how to Set or Get variable to/from a Flash movie:

```
FlashStr$= 'MmbText.mmbstr,'+ 'GetDateTime'
** Set the 'GetDateTime' string to MmbText.mmbstr variable located in Flash movie
FlashSetVar("Flash","FlashStr$")
** Give the Flash a little time to processing all the needed actions.
Pause("100")
**  Date$ and Time$ variable were filled directly from Flash by below two lines code:
**  fscommand ("mmb","Date$ =" + MMBCurDateFull);
**  fscommand ("mmb","Time$ =" + MMBCurTimeFull);
LoadText("DateStr","Date$")
LoadText("TimeStr","Time$")
**  but the Day$ variable is read by FlashGetVar command directly from MMB
FlashGetVar("Flash","MmbText.MMBCurDay,Day$")
LoadText("DayStr","Day$")
```

FlashSetVar("ObjectLabel","FlashVar/MMBVar")

Description

Set a new value to Flash movie variable.
Note: the same things as for **FlashGetVar** command ;-)

FlashSetFrame("ObjectLabel","Frame")

Description	

This will allow you to go to any frame right from your application. Simply set the Flash object name to the first parameter and to the second parameter set the frame to which you want jump.

Code Example	

```
** Jump to 50th frame of flash file loaded in Flash object
FlashSetFrame("Flash","50")
Flash("Flash","Play")
```

For more "real" examples check the **flash_examples.mbd** or **flashserie.mbd** files.

FlashGetFrame("ObjectLabel","Variable")

Description	

Almost the same as FlashSetFrame but it obtains the current frame of the Flash movie.

Code Example	

```
** Returns current frame of flash file loaded in Flash object
FlashGetFrame("Flash","CurFrame")
```

For more "real" examples check the **flash_examples.mbd** or **flashserie.mbd** files.

FlashGetProp("ObjectLabel","Property,Variable")

Description	

Get a defined property from the Flash movie and store it in a string variable.

Property can be one of following:
SCALE - returns the scale of Flash movie.
BGCOLOR - returns the RGB code of the background color in the hexadecimal value (e.g. ffff99 = red color)
QUALITY - returns the current quality level of Flash player.
PLAYING - returns if movie is playing or not (e.g. true/false string)
MOVIE - returns the name of Flash movie.
TOTALFRAMES - returns the number of all frames.
ORIGINALWIDTH - returns the original width of flash file.
ORIGINALHEIGHT - returns the original height of flash file.
PLAYERVERSION - returns the version of already installed Flash player.
FILEVERSION - returns the version of Flash player required/used by a given Flash file.

Code Example

This small example will show you the usage of **FlashGetProp**:

```
** if you wan to return the background color of played movie, use this code:
FlashGetProp("Flash","BGCOLOR,bgcolor$")

** or if you want to check if the Flash movie is playing, then use this:
FlashGetProp("Flash","PLAYING,play$")

** get the original width of the Flash file (the width in which was the file originally created).
FlashGetProp("Flash","ORIGINALWIDTH,owidth$")
```

For more "real" examples check the **flash_examples.mbd** or **flashserie.mbd** files.

Calling the MMB commands directly from Macromedia Flash Action Script:

To communication between the Flash movie and MMB is used the fscommand command. It has two parameters: *command* and *arguments*. If you want to send a command to MMB, then the command parameter must be always string "mmb" and the second parameter (argument) should contain the MMB scripting action.

Example of usage of fscommand command in Flash movie:

```
// this example will fill the Time$ variable from the Flash variable
fscommand("mmb","Time$ =" + MMBCurTimeFull);

// this example will call the LoadText MMB command and fill the DirStr text object in MMB
fscommand("mmb","LoadText(\"DirStr\",\""+ DirString +"\")");
```

```
// this example will go to next MMB page
fscommand("mmb","NextPage()");

// this example will go to a defined MMB page - in this case "Page 2"
fscommand("mmb","Page(\"Page 2\")");

// this example will run an external mbd file (and jump to "Page 1" of this mbd)
from the source folder where the application.exe is located
fscommand("mmb","RunMBD(\"<SrcDir>\\test_1.mbd\",\"Page 1\")");
```

To include a quotation mark in a string, precede it with a backslash character (\). This is called "escaping" a character. There are other characters that cannot be represented in ActionScript except by special escape sequences. The following table provides all the ActionScript escape characters:

Escape sequence	Character
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Line-feed character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Tab character (ASCII 9)
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash
\000 - \377	A byte specified in octal
\x00 - \xFF	A byte specified in hexadecimal
\u0000 - \uFFFF	A 16-bit Unicode character specified in hexadecimal

Consult the Macromedia Flash documentation for more information about the fscommand command usage and Action Script programming as such.

12.10.1.4 ListBox Commands

ListBoxAddItem("ObjectLabel","Parameters")

Description

Add item(s) to a ListBox.

Possible parameters:

- **simple text** (e.g. bla-bla-bla)
- **string variable** (e.g. text$)
- **string array** (e.g. text[n]$)
- **string separated by separator** (e.g. str$='item1#item2#item3#')
- **text file** (e.g. c:\Documents\testfile.txt)
- **<List>** to insert internal lists
- **playlist** MMB(*.m3l) and WinAmp playlists(*.pls, *.m3u) - (e.g. <SrcDir>\mix.m3u)
- **RESET** to clear listbox content

RANDOMIZE parameter to randomize the listbox content is **no longer used** in MMB 4.9.5. Use the below **ListBoxSort** function to randomize or sort the ListBox contents.

Code Example

This small example will show the OpenFile dialog (with predefined mask) and after selection will load the selected playlist to the ListBox:

```
OpenFile("Play Lists(*.m3l;*.m3u;*.pls)|*.m3l;*.m3u;*.pls|All Files|*.*||","*.m3l;*.m3u;*.pls")
ListBoxAddItem("SongList","RESET")
ListBoxAddItem("SongList","OpenFile$")
```

String variable separated by a separator:

```
str$= 'item 4#item 5#item 6#'
ListBoxAddItem("ListBox1","str$,#")
```

For more examples check the **ListBox_tut.mbd**.

Additional Info

If you want to add a string containing the filename with extension use STRING: word in front of the filename string...

```
filename$='STRING:'+ExtractName(Path$)+ExtractExt(Path$)
ListBoxAddItem("ListBox","filename$")
```

ListBoxDeleteItem ("ObjectLabel","ItemNum")

Description

Delete a given item from a ListBox. To delete all items use -1 as a parameter

Note: *The the first item in the ListBox is always 1.*

Code Example

The following example will delete the item #5 from the ListBox:

```
ListBoxDeleteItem("ListBox"," 5")
```

Here is an advanced example of **ListBoxDeleteItem** action. It will delete the selected item(s) from the ListBox after pressing Delete key:

- Create a new Script object and specify the keyboard shortcut to DELETE key.
- Insert the below code to the Script section:

```
ListBoxGetSelectedItems("ListBox","SelItemsArray$,NumSelItemsArray$,#,
NumOfSelItems")
For i=NumItems To 1
  numitem$ = GetArrayItem(NumSelItemsArray$,#,i)
  numitem = VAL(numitem$)
  ListBoxDeleteItem("ListBox","numitem")
Next i
```

ListBoxSortItems("ObjectLabel","Parameters")

Description

Function for randomizing and sorting the ListBox contents. Possible parameters:
NAME - sort items by name
TIME - sort items by time
VALUE - sort items by (integer) value
REVERSE - reverse current state
RANDOMIZE- randomize ListBox

Code Example

This simple example sorts the ListBox contents according the NAME of items

ListBoxSortItems("ListBox","NAME")

For example these items:

12 test
11 test
1 test
32 test
9 test

...sorted by NAME:

1 test
11 test
12 test
32 test
9 test

...but sorted by VALUE:

1 test
9 test
11 test
12 test
32 test

ListBoxSelectItem("ObjectLabel","ItemNum")

Description	

Select a given item in a ListBox. To select all items use -1 as a parameter.

Code Example	

The following example will select the item # 5 in the ListBox:

ListBoxSelectItem("ListBox"," 5")

ListBoxDeselectItem("ObjectLabel","ItemNum")

Description
Deselect a given item. To deselect all items use -1 as parameter

Code Example
The following example will deselect all items in the ListBox: **ListBoxDeselectItem**("ListBox"," -1")

ListBoxMoveItem("ObjectLabel","ItemNum")

Description
Move selected item to position specified in Parameters field.

Code Example

The following example will move selected item **UP**.
 	
ListBoxGetSelectedItems("ListBox","SelItems$,SelItemsNum$,#,AllNumItems")
For j=1 To AllNumItems
  SelItem$ = GetArrayItem(SelItems$,#,j)
  SelNum$ = GetArrayItem(SelItemsNum$,#,j)
  SelNum = VAL(SelNum$)
Next j
If (SelNum>1) Then
  ListBoxMoveItem("ListBox","SelNum-1")
End
 	
The next example will move selected item **DOWN**.
 	
ListBoxGetItems("ListBox","ItemsArray$,NumItemsArray$,#,NumOfAllItems")
ListBoxGetSelectedItems("ListBox","SelItems$,SelItemsNum$,#,AllNumItems")
For j=1 To AllNumItems
  SelItem$ = GetArrayItem(SelItems$,#,j)
  SelNum$ = GetArrayItem(SelItemsNum$,#,j)
  SelNum = VAL(SelNum$)
Next j
If (SelNum<NumOfAllItems) Then

```
  ListBoxMoveItem("ListBox","SelNum+1")
End
```

Additional Info	

Limitation: Multiple selected items cannot be moved!

For more information check **move_up_down.mbd** example file.

ListBoxGetItems("ObjectLabel","Parameters")

Description	

Retrieves all items from ListBox object. The first parameter must be string variable which contains items delimited by user defined delimiter. The second parameter must be also string variable which contains items indexes separated by delimiter too. The third parameter defines delimiter. The fourth parameter is a numeric variable, which contains number of all items in ListBox

Code Example	

The following example return items from the ListBox:

```
ListBoxGetItems("ListBox","ItemsArray$,NumItemsArray$,#,NumOfAllItems")
For i=1 To NumOfAllItems
  GetItem$ = GetArrayItem(ItemsArray$,#,i)
  Message("ListBox Item...","GetItem$")
Next i
```

Additional Info	

ListBoxGetItems can not only retrieve items from ListBox, but it can also feed the internal SongList with the ListBox items. Just use <List> constant as a second parameter of **ListBoxGetItems** function instead of usual **ListBoxGetItems** parameters.

Use this script for feeding the internal SongList <List> with the ListBox contents:

```
ListBoxGetItems("SongList","<List>")
```

ListBoxGetSelectedItems ("ObjectLabel","Parameters")

Description

Retrieves selected items from ListBox object. The first parameter must be string variable which contains items delimited by user defined delimiter. The second parameter must be also string variable, which contains items indexes separated by delimiter too. The third parameter defines delimiter. The fourth parameter is a numeric variable, which contains number of selected items.

Code Example

The following example returns selected items in ListBox:

```
ListBoxGetSelectedItems("ListBox","SelItemsArray$,NumSelItemsArray$,#,
NumOfSelItems")
For i= 1 To NumOfSelItems
  SelItem$ = GetArrayItem(SelItemsArray$,#,i)
  Message("Selected Item...","SelItem$")
Next i
```

Additional Info

For more information check **listbox_on_master_page.mbd** or **listbox_fileextensions.mbd** example files or some other examples from MMB \Samples folder.

ListBoxParam("ObjectLabel","Parameters")

Description

Changes some of ListBox parameters defined in design-time in a run-time.

DRAG&DROP=ON/OFF - turns drag&drop on/off
NUMBERS=ON/OFF - show/hide item numbers
TIMES=ON/OFF - show/hide item times (only supported Audio/Video files)
BACKGROUND=R,G,B - sets background color
TEXT=R,G,B - set text color
IDTAGS=ON/OFF - search for (audio) IDtags when is ON.

Code Example

The below examples shows you how to change some of ListBox parameters:

ListBoxParam("ListBox","DRAG&DROP = ON")

ListBoxParam("ListBox","NUMBERS = OFF")

ListBoxParam("ListBox","TIMES = ON")

ListBoxParam("ListBox","BACKGROUND = 128,128,128")

ListBoxParam("ListBox","TEXT = 128,0,0")

ListBoxParam("ListBox","IDTAGS = OFF")

12.10.1 Song List Commands

You have your "internal" sound list where you can add ogg/wma/asf files or import files from external list m3l/m3u/pls/txt. Even the description 'Song List' predetermine to use Song List as a storage of audio files, you can use it as a general storage of video, picture or txt files too. Added files are stored in <List> storage.

See **AudioTags.mbd** sample file in Samples directory.

SongListReset()

Description
Delete All items from song internal list <List>. It's useful in cases, when you need to replace the actual songlist with the new content.

Code Example

```
SongListReset()
```

SongListAdd("Path")

Description
Add audio (video, image, etc.) file to the internal list <List>

Code Example

```
**This will add a file to the internal list <List>
background-color:#ffffe0("<SrcDir>\mysound.ogg")

**This will add a selected file from OpenFile dialog to the internal list <List>
OpenFile("MP3 Audio Files (*.mpg;*.mp1;*.mp2;*.mp3)|*.mpg;*.mp1;*.mp2;*.
mp3|OggVorbis Files(*.ogg)|*.ogg|Win Media Audio Files (*.wma)|*.wma|
Advanced Streaming Format Files (*.asf)|*.asf|Playlist Files (*.m3l;*.m3u;*.pls)|
*.m3l;*.m3u;*.pls|All Files|*.*||","*.mp3")
If (OpenFile$ <> '') Then
SongListAdd("OpenFile$")
```

```
End

And finally this advanced example shows you how to feed SongList and ListBox
with the content of a selected directory (with subdirectories) and sort them by
Name

** Browse for folder
BrowseForFolder("Select a folder:","")
** Feed variable with the path to selected directory
Audiodir$=CBK_OpenDir
If (Audiodir$ <> '') Then
** Search files in selected directory and feed the SongList
  SearchForFiles("Audiodir$","")
** Reset ListBox
  ListBoxAddItem("ListBox","RESET")
** Feed the ListBox with SongList content
  ListBoxAddItem("ListBox","<List>")
** Sort ListBox by Name
  ListBoxSortItems("ListBox","NAME")
** Reset internal SongList..
  SongListReset()
**..and refill it with sorted ListBox
  ListBoxGetItems("ListBox","<List>")
End
```

SongListDel("Number")

Description

Delete a given item from internal list <List>.

Code Example

```
**This will delete first item from the internal list <List>
   SongListDel("1")

**This will delete items 1 to 10 from internal list <List>
For i=1 To 10
   SongListDel("i")
Next i
```

SongListPlay("Number")

Description	

Play the internal list of audio files <List>, you can specify from which item to start. If no item is specified, then it will start from the first item.

Code Example	

```
**This will play internal list from 1st item.
SongListPlay("")

**This will start playback of selected file in ListBox
ListBoxGetSelectedItems("SongList","Items$,ItemsNum$,#,NumItems")
numitem$ = GetArrayItem(ItemsNum$,#,1)
SongListPlay("VAL(numitem$)")
```

SongListNext()

Description	

Play the next song from the internal song list <List>.

Code Example	

```
**This will play the next song from the <List>
SongListNext()
```

SongListPrev()

Description	

Play the previous song from the internal song list <List>.

Code Example

```
**This will play the previous song from the <List>
SongListPrev()
```

SongListLoad("Path","FileFormat")

Description

Feed the internal <List> with external list of files.
Supported file formats : *.m3l, *.m3u, *.pls, *.txt (plain txt file).

In "File Format" field you can specify format (M3L, M3U, PLS, TXT) for reading so you can read TXT file as M3L format or M3U as TXT. This is useful in some special cases (e.g. editing), because all these formats are different and sometimes you may need to read for example M3U as TXT. Also, if SongList contains links to non-audio files (like images) reading as TXT will assure that the list of files will be read as plain txt file and not list of audio files (i.e. doesn't attempt to get times from files).

Also, if you want to use TXT file as an audio list, you will have to always specify the loading in M3L format, because it uses some audio list reading improvements.

If the FileFormat is not defined, the SongListLoad will load the file in the original file format (i.e. m3u as m3u, txt as txt,...)

Code Example

```
**This load the m3u playlist to the internal list <List>
SongListLoad("<SrcDir>\list.m3u","m3u")

**This load the selected playlist to the internal list <List> and feed the ListBox
OpenFile("Play Lists(*.m3l;*.m3u;*.pls;*.txt)|*.m3l;*.m3u;*.pls;*.txt|All Files|*.*||","*.m3l;*.m3u;*.pls;*.txt")
If (OpenFile$ <> '') Then
   SongListReset()
   ListBoxAddItem("SongList","RESET")
   SongListLoad("OpenFile$","")
   ListBoxAddItem("SongList","<List>")
End
```

SongListRND()

Description

Randomize the Internal List <List>.

Code Example

```
**This randomize internal list <List>
SongListRND()

**This load the selected playlist to the internal list <List>, randomize it and feed the ListBox
OpenFile("Play Lists(*.m3l;*.m3u;*.pls;*.txt)|*.m3l;*.m3u;*.pls;*.txt|All Files|*.*||","*.m3l;*.m3u;*.pls;*.txt")
If (OpenFile$ <> '') Then
   **Reset previous <List> and ListBox
   SongListReset()
   ListBoxAddItem("SongList","RESET")
   **Load new <List>
   SongListLoad("OpenFile$")
   **Randomize new list
   SongListRND()
   **Feed the ListBox
   ListBoxAddItem("SongList","<List>")
End
```

SongListEdit()

Description

Open special window that enables user edit the virtual song list, save to disk or load it from disk. The adding new item is very easy - just drag and drop files from explorer to this edit window. From MMB4.9 you can save/load or edit the internal song lists by individual scripting commands (**SongListLoad, SongListSave, SongListDel**, etc.).

Code Example

```
**This open <List> edit dialog
SongListEdit()
```

SongListSave("SongList/ListBoxLabel","Path")

Description	

Save internal List into file. If file or ListBoxObject is not defined, "Save As" dialog will appear. Optionally, in Object field you can define ListBox object to store its content instead of internal List.

Code Example	

```
**This open SaveAs dialog, for selecting path and filename of SongList file and
then save the internal SongList <List>.
SongListSave("","")

Use the below code if you want to save <List> or ListBox content to a file with a
different extension than m3l. However, the output file will be still plain m3l file ;)

**This open SaveAs dialog, for selecting path and filename of SongList file and
then save the internal SongList <List>.
SaveFile("Play Lists(*.m3l;*.m3u;*.pls;*.txt)|*.m3l;*.m3u;*.pls;*.txt|All Files|*.
*||","*.m3l;*.m3u;*.pls;*.txt")
If (OpenFile$ <> '') Then
   SongListSave("<List>","OpenFile$")
End
```

SongListTime()

Description	

Calculate total time of songs (or video) in <List> and fill the **CBK_TotalList** or **CBK_TotalListSec**. These CBK constants are not filled right after loading the files into <List> because processing of very long SongList can take very long time (especially on slower computers). Therefore we decided to do not process the TotalList calculation automatically after loading the files to SongList, but rather provide this SongListTime function.

Code Example

```
**Simply use this function according your needs (for example after loading
SongList) and don't forget to add Text object with label CBK_TotalList or
CBK_TotalListSec label.
OpenFile("Play Lists(*.m3l;*.m3u;*.pls;*.txt)|*.m3l;*.m3u;*.pls;*.txt|All Files|*.
*||","*.m3l;*.m3u;*.pls;*.txt")
If (OpenFile$ <> '') Then
    SongListReset()
    ListBoxAddItem("SongList","RESET")
    SongListLoad("OpenFile$")
    ListBoxAddItem("SongList","<List>")
    SongListTime()
End
```

12.10.1 TTS Commands

MMB offers Text-to-Speech features through it's script language. Main features of TTS commands are:

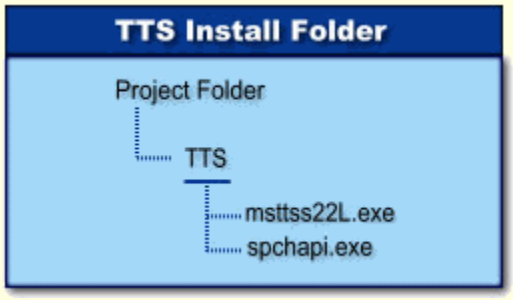
- Installation of TTS engine
- Speaking of text from variables and objects
- Voice pitch and speed settings

InstallTTS()

Description

Installs TTS engine if it has not been installed previously on the running system. Command checks TTS status and performs installation if needed. No parameters required.

When distributing project, make sure to include TTS engine installation files (msttss22L.exe, spchapi.exe) in folder labeled TTS, inside your project folder.



Code Example

```
**Install TTS engine :
InstallTTS()
```

Additional Info

If your project uses TTS features, it is recommended to call InstallTTS at the project startup, just to make sure TTS engine is ready for use.

TTS installation files are placed in TTS subfolder of MMB's folder. In case you don't have 'em, install files can be downloaded from:

http://www.mediachance.com

InitTTS()

Description

Initializes TTS engine. No parameters required.

During the initialization, black info box is displayed in the center of the screen. While TTS startup might take a few seconds on low-performance computers, it is recommended to:

a) call this command at the project startup

b) do not use TTS engine at the very beginning of project startup, but with commands like Pause() and ScriptTimer() give a few seconds to TTS engine for initialization.

Code Examples

```
** Initialize TTS engine:
InitTTS()
```

Say("Text$")

Description

Reads text set either as a command parameter or string variable .

In first case, text is fixed as a command parameter:

```
Say("Hello from MMB")
```

In second case, you will first set text to a string variable:

```
text$='Hello from MMB'
```

And then set string variable to Say command:

```
Say("text$")
```

TTS engine must be installed and initialized.

Code Examples

```
** Read text from fixed command parameter:
Say("Welcome to my project")

** Read text from string variable as a command parameter:
text$='Welcome to my project'
Say("text$")
```

SpeakText("ObjectLabel")

Description

Reads text from any Text, Button or Paragraph object.

Object label is set as a command parameter.

If object with text is labeled:

```
ShoutBox
```

...then it's label put into command will look like this:

```
SpeakText("ShoutBox")
```

TTS engine must be installed and initialized.

Code Example

```
** Read text from object labeled "StoryParagraph":
SpeakText("StoryParagraph")
```

StopTTS()

Description

Stops reading text (started using Say or SpeakText command).

Code Example

```
** Stop reading text:
StopTTS()
```

PauseTTS()

Description

Pauses reading of text (started using Say or SpeakText command).

Use ResumeTTS command to resume reading.

Code Example

```
** Pause reading text:
PauseTTS()
```

ResumeTTS()

Description

Resumes reading of text (after it was paused using PauseTTS command).

Code Example

```
** Resume reading text:
ResumeTTS()
```

PitchTTS("Frequency")

Description

Sets pitch of voice for TTS text reading.

Command parameter sets voice frequency (in Hz) either as a fixed parameter or through numerical variable, in range:

```
50 - 200
```

Default frequency is 100.

In the first case, frequency is fixed as command parameter:

```
PitchTTS("80")
```

In the second case, you will first set frequency to a numerical variable:

```
pitch=80
```

And then set numerical variable to PitchTTS command:

```
PitchTTS("pitch")
```

Code Examples

```
** Set pitch to 130 Hz from fixed command parameter:
PitchTTS("130")

** Set pitch  to 130 Hz  from numerical variable as a command parameter:
pitch=130
PitchTTS("pitch")
```

SpeedTTS("Speed")

Description

Sets speed of TTS text reading.

Command parameter sets number of words per minute either as a fixed parameter or through numerical variable, in range:

```
30 - 450
```

Default value is 150 words per minute.

In the first case, speed is fixed as a command parameter:

```
SpeedTTS("300")
```

In the second case, you will first set speed to a numerical variable:

```
speed=300
```

And then set numerical variable to SpeedTTS command:

```
SpeedTTS("speed")
```

Code Examples

```
** Set speed to 300 words/min from fixed command parameter:
SpeedTTS("300")

** Set speed to 300 words/min from numerical variable as a
** command parameter:
speed=300
SpeedTTS("speed")
```

12.10.1 Browser Commands

See **html_browser.mbd** and some other examples available in Samples folder.

Browser("ObjectLabel","Commands/URL Path")

Description
Controls the HTML Browser Object Possible commands: Back - return back to previous page Forward - jump forward Stop - stop loading HTML page Refresh - refresh current HTML page OpenFile - Show OpenFile dialog with htm/html files as a default extension Print - print HTML page Anything else (including string variables) will be opened as file (HTM/HTML, MHT, PDF...) or url. WARNING! HTML object can load the PDF files, but only if the appropriate PDF reader (AcrobatReader) is already installed on the user's computer. For a real example of a Browser object check the example here... **html_browser.mbd** NOTE: If you want to obtain the actual URL path from the Browser object, use the **CBK_URLPath** constant.

Code Example

```
** This will load the mediachance web
Link$='www.mediachance.com'
Browser("Browser","Link$")

** This will print contents of the Browser object
Browser("Browser","Print")
```

12.10.1 Image Matrix Commands

Summary

MatrixSet("MatrixName[Column,Row], "Image Index")
sets an image (represented with image index) into a cell that addressed with **Column** & **Row**.

MatrixGet("MatrixName[Column,Row], "VariableName")
reads image index from a cell that addressed with **Column** & **Row**. The image index is then stored in **VariableName** as an integer value.

MXCOL ===> Returns current column
MXROW ===> Returns current row

Now, we can set any of images in a addressed cell.

MatrixSet("MatrixName[Column,Row],"Image Index")

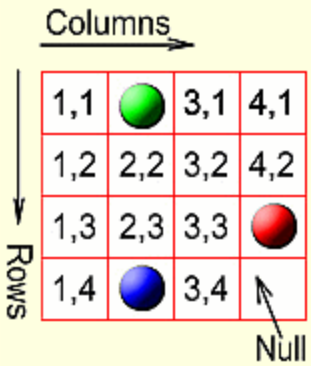
Description	

MatrixSet("MatrixName[Column,Row], "Image Index")
sets an image (represented with image index) into a cell that addressed with **Column** & **Row**.

Code Examples	

```
MatrixSet("MyMatrix[2,1]","2")
MatrixSet("MyMatrix[4,3]","1")
MatrixSet("MyMatrix[2,4]","3")
MatrixSet("MyMatrix[4,4]","0")
```

Above mentioned scripts will set the images into the specified cells :



How can we fill the whole matrix with a specified image?

```
RedBall=1
**Fill antire matrix with redball
```

```
MatrixSet("MyMatrix[0,0]", "RedBall")


Or how can we fill the specified row with a specified image?
    GreenBall=2
    **Fill second row with greenball
    MatrixSet("MyMatrix[0,2]", "RedBall")

And we also fill the specified column with a specified image?
    BlueBall=3
    **Fill fourth row with second row with blueballl
    MatrixSet("MyMatrix[4,0]", "RedBall")

How can we clear whole matrix or a specific part of matrix?
    NoImg=0
    MatrixSet("MyMatrix[0,0]", "NoImg") **Fill whole cells with null image (no image)
    MatrixSet("MyMatrix[0,2]", "NoImg") **Fill second row with null image (no image)
    MatrixSet("MyMatrix[4,0]", "NoImg") **Fill fourth row with null image (no image)
    MatrixSet("MyMatrix[3,2]", "NoImg") **clear cell(3,2)
```

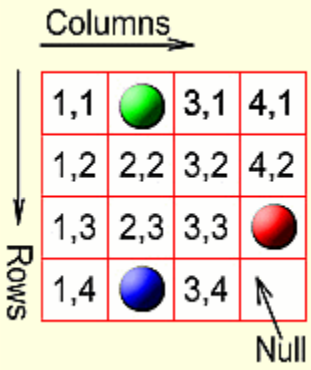
What about reading from matrix?

MatrixGet("MatrixName[Column,Row],"VariableName")

Description	

MatrixGet("MatrixName[Column,Row], "VariableName")
reads image index from a cell that addressed with **Column** & **Row**. The image index is then stored in **VariableName** as an integer value.

Code Examples	



MatrixGet("MyMatrix[2,1]", "CurImg") ** CurImg = 2
MatrixGet("MyMatrix[4,3]", "NoImg") ** CurImg = 1
MatrixGet("MyMatrix[2,4]", "NoImg") ** CurImg = 3
MatrixGet("MyMatrix[4,4]", "NoImg") ** CurImg = 0

Note: if you use MatrixGet and the col or row is out of range it returns -1.

MatrixGet("MyMatrix[7,5]", "NoImg") ** CurImg = -1 because our Matrix dimensions are 4x4

How can we know which cells is active (in other words selected by user)? There are two variables that always represent current Column & current Row.

MXCOL/MXROW

Description

MXCOL ===> Returns current selected column
MXROW ===> Returns current selected row

MXCOL & MXROW returns 0 if no cell is selected.

Code Examples

```
**Get the image index of the current cell
MatrixGet("MyMatrix[MXCOL,MXROW]","MyVariable")

**Set the green ball to the current cell
GreenBall = 2
MatrixSet("MyMatrix[MXCOL,MXROW]","GreenBall")

**Check the user whether selected a cell or not
If (MXCOL=0 & MXROW=0) Then
   Message("You should select a ball","")
End
```

12.10.1 Animated GIF Commands

AGifPlay ("ObjectLabel")

Description	

Play gif (for example if autoplay feature is off).

Code Example

```
** Start playing animated gif
AGifPlay("AniGif")
```

AGifStop ("ObjectLabel")

Description	

Stop gif. The next play will continue from this frame.

Code Example

```
** Stop playing animated gif
AGifStop("AniGif")
```

AGifReset ("ObjectLabel")

Description	
Reset gif. Set the first frame.	

Code Example	

```
** Reset animated gif
AGifReset("AniGif")
```

Although using GIF is an easy way to add the image animations to MMB project, we would suggest you to use some more scripting to animate bitmap/hotspot objects. See for example **giflikeanim_frommasterpage.mbd** example project.

Top Level Intro

This page is printed before a new
top-level chapter starts

Part

13 MMB Command Reference

13.1 List of Commands

Publication

FirstPage()
LastPage()
NextPage()
PrevPage()
Page("PageLabel")
Exit()
Minimize()
Maximize()
Restore()

Object

CreateTextButton("InLabel","OutLabel$,x,y,w,h,text")
CreateText("InLabel","OutLabel$,x,y,text")
CreateParagraph("InLabel","OutLabel$,x,y,w,h,text")
CreateCircle("InLabel","OutLabel$,x,y,w,h,r,g,b")
CreateRectangle("InLabel","OutLabel$,x,y,w,h,r,g,b")
CreateLine("InLabel","OutLabel$,x,y,w,h,r,g,b")
CreateLineAB("InLabel","OutLabel$,x1,y1,x2,y2,r,g,b")
CreateHotSpot("InLabel","OutLabel$,x,y,w,h")
CreateScript("InLabel","OutLabel$")
DeleteObject("ObjectLabel")
Hide("ObjectLabel")
Show("ObjectLabel")
Invert("ObjectLabel")
MoveObject("ObjectLabel","x,y,w,h")
MoveTo("ObjectLabel","x,y,steps,type")
ReorderObject("ObjectLabel","Parameters")

Global Object Parameters

SetObjectParam("ObjectLabel,"Parameters")
SetProjectParam("ProjectParamName","Parameters")

System

Timers

PageTimer("ms","PageLabel")
ExitTimer("ms")
ScriptTimer("ObjectLabel","ms")
RunScript("ObjectLabel")
RunScriptCode("ObjectLabel")
Pause("ms")
Return()
Break()
Refresh()

Print

PrintText("ObjectLabel","Parameters")

PrintPage("Page")
PrintRect("ObjectLabel","Zoom")

PlugIns
PluginSet("PlugInLabel","Variable")
PluginGet("PlugInLabel","Variable")
PluginRun("PlugInLabel","Command")

Others
Message("String","Variable")
MessageEx("Title","Text, Flag[, Timeout]")
SysCommand("Command","Parameters")
DisplayValue("ObjectLabel","Value")
SendCommand("BindingObjectLabel","Menu ID a,b[,c]")
LoadText("ObjectLabel","Path/StringVariable")
Run("Path","Parameters")
RunMBD("Path","PageLabel")
OpenFile("Filter","Default extension")
SaveFile("Filter","Default extension")
BrowseForFolder("Prompt","StartingFolder")
ColorPicker()
FontPicker()
FileExist("Path","Variable")
FileString("SubString","Variable")
SearchForFiles("Path","Filter")
SaveVariable("Name","Variable")
LoadVariable("Name","Variable")
InstallFont("Path")
Browser("BrowserObjectLabel","Command/URLPath")
Clipboard("SEND/GET","Variable")

Sound
System
VolumeUp("Volume 0-100")
VolumeDown()
FMODConfig("Driver[,MPEG mode]")
BackgroundPlay("Path","Parameters")
BackgroundPause()
BackgroundStop()
PlaySound("Path")
StopSound()

WAV
WavePlay("Path","Parameters")
WaveStop()

MID
MidiPlay("Path","Parameters")
MidiStop()

MOD
ModOpen("Path")
ModPlay()

ModStop()

OGG

AudioOpen("Path")
AudioPlay()
AudioStop()
AudioPause()
AudioRewind("Seconds","Parameters")

SongList

SongListReset()
SongListAdd("Path")
SongListDel("Number")
SongListPlay("Number")
SongListNext()
SongListPrev()
SongListLoad("Path")
SongListRND()
SongListEdit()
SongListSave("SongList/ListBox","Path")
SongListTime()

CD

CDPlay()
CDStop()
CDPause()
CDTrack("Track")
CDForward()
CDBackward()
CDPlayPause()
CDSkipForward()
CDSkipBackward()
WhichCDTrack("TrackVar")

Adv. Objects

Image

ViewImage("Path","Parameters")
ReplaceImage("ObjectLabel","Path")
UseImageBkgColor("ObjectLabel","Parameters")
SetImageBkgColor("ObjectLabel","R,G,B")
SetImageOrigin("ObjectLabel","X,Y")
ResizeImage("ObjectLabel","W,H")
RotateImageRel("ObjectLabel","Angle")
RotateImageTo("ObjectLabel","Angle")
ScrollImageView("ObjectLabel","X,Y")
ZoomImageView("ObjectLabel","Zoom")
RestoreImage("ObjectLabel")
ImageOpacity("OpacityLevel")

ListBox

ListBoxAddItem("ObjectLabel","Parameters")
ListBoxDeleteItem("ObjectLabel","ItemNum")
ListBoxSortItems("ObjectLabel","Parameters")

ListBoxSelectItem("ObjectLabel","ItemNum")
ListBoxDeselectItem("ObjectLabel","ItemNum")
ListBoxMoveItem("ObjectLabel","ItemsNum")
ListBoxGetItems("ObjectLabel","Parameters")
ListBoxGetSelectedItems("ObjectLabel","Parameters")
ListBoxParam("ObjectLabel","Parameters")

Flash

Flash
FlashSetVar
FlashGetVar
FlashSetFrame
FlashGetFrame
FlashGetProp

Video

VideoLoad("ObjectLabel","Path")
VideoPlay("ObjectLabel")
VideoPause("ObjectLabel")
VideoStop("ObjectLabel")
VideoRewind("ObjectLabel","Parameters")
VideoClose("ObjectLabel")
VideoSpeed("ObjectLabel","Parameters")
VideoScale("ObjectLabel","Width,Height")
VideoParam("ObjectLabel","Parameters")

MCI

MCIObject("ObjectLabel","Command")
MCICommand("MCIString")

GIF

AGifPlay("ObjectLabel")
AGifStop("ObjectLabel")
AGifReset("ObjectLabel")

AV Object

AudioVisualizationType("ObjectLabel","Type")
AudioVisualizationColor("ObjectLabel","Parameters")

TTS

InstallTTS()
InitTTS()
Say("Text$")
SpeakText("ObjectLabel")
StopTTS()
PauseTTS()
ResumeTTS()
PitchTTS("Frequency")
SpeedTTS("Speed")

Matrix

MatrixSet("MatrixName[Column,Row],"Image Index")
MatrixGet("MatrixName[Column,Row],"Variable Name")

13.2 List of Predefined Functions

Publication

- **PubX**()
- **PubY**()
- **PubWidth**()
- **PubHeight**()
- **ClientWidth**()
- **ClientHeight**()
- **IsMinimized**()
- **IsMaximized**()

Object

- **ObjectX**(object$)
- **ObjectY**(object$)
- **ObjectWidth**(object$)
- **ObjectHeight**(object$)
- **CurrentObject**()
- **IsVisible**(object$)
- **ImageScrollX**(object$)
- **ImageScrollY**(object$)
- **ImageWidth**(object$)
- **ImageHeight**(object$)
- **ScrollBarSize**(object$)
- **GetImageOpacity**(object$)
- **GetVideoParam**(object$)
- **MXCOL**
- **MXROW**

System

- **ScreenWidth**()
- **ScreenHeight**()
- **WorkAreaWidth**()
- **WorkAreaHeight**()
- **MouseX**()
- **MouseY**()
- **MouseLButton**()
- **MouseRButton**()
- **MouseMButton**()
- **WinVer**()
- **UsingWinNT**()
- **ScreenColors**()
- **ProcType**()
- **PrecFreq**()
- **GetMemory**()

Conditions/Loops

- **If..Else..End**
- **For..Next**
- **Infinity**
- **TRUE**

FALSE

Strings

VAL(string$)
CHAR(number)
CHR(number)
ORD(char$)
LEN(string$)
LOW(string$)
UPP(string$)
POS(char$,string$)
NOL(path$)
StrCopy(string$,position,count)
StrDel(string$,position,count)
StrIns(string$,insert$,position)
StrGet(string$,position)
StrSet(string$,position,char$)
StrOfChar(char$,count)
StrChange(string$,replacethis$,hereby$)
StrToFile(filename$,string$,append,linefeed)
StrToLine(filename$,string$,toline,overwrite)
StrFromFile(filename$,fromline,numberoflines)
ExtractExt(path$)
ExtractDir(path$)
ExtractName(path$)
ExtractDrive(path$)

Numbers

ABS(number)
INT(number)
RND(maxvalue)

Arrays

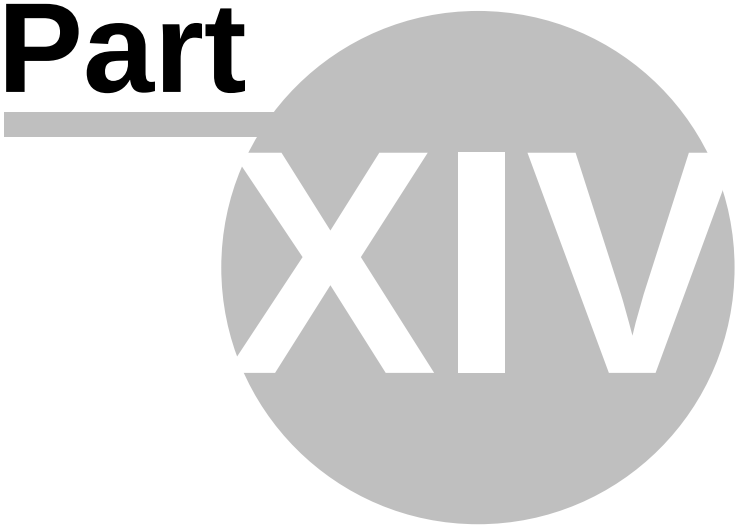
GetArrayItem(arrayofstrings$,delimiter,index)
GetArrayNum(arrayofstrings$,delimiter)

Paths

<CD>
<Embedded>
<File>
<List>
<SrcDir>
<SrcDrive>
<System>
<Temp>
<Windows>
<This>

Top Level Intro

This page is printed before a new top-level chapter starts



14 MMB Plugins

14.1 MMB PlugIns

how to get

MMB PlugIns

work for you

Pocket-sized operator's manual

with pictures and schemes included

by bokzy, 2003.

Ok, so you tried MMB, bought it the very next day, spent weeks learning MMB scripting language and just as you thought you've got all twines in your hands - Murphy in the form of your first neighbour comes, storming you with words:

"John, can you borrow me some nails, hammer and a copy of tweak plugin with some samples for it ?"

Nails and hammer are somewhere in garage, but what in heck is tweak plugin, what he needs that for ?

You choose to improvise. Got some rusty nails, hammer's got a broken handle but it'll do it - and tweak ? He probably needs AC/DC plug, so you pick one from your collection and everything's there.

Except the smile on your neighbour's face.

It must be 'cause nails are little rusty.

Is it ?

Hm, not exactly.

Yes he took hammer and the nails, but why did he throw away that AC/DC you gave him ? It's broken ?

Very next day you're going for a visit. Entering neighbour's house, starting some ordinary chit-chat with a beer in your hand... having one eye on computer screen... yep, you see he's working on something in MMB. Yep, you see he's even using scripting editor (so that's why he spent last 10 nights awake).

Yep... Script commands flying around - Show, Hide, even MoveObject is there... VolumeUp, VolumeDown, Plugin.... errr... what was that ?

Plugin...Set ?

What for ? You were so successful in ignoring that command !

So far.

Some other film starts to roll in your head. He didn't want AC/DC plug ! It was a trap, to see how far you've gone in using MMB !

Finishing your second beer you hurry home, slam the doors, start computer... got some serious talks with MMB again !

Time to spend few extra nights not in bed with your deareast, but with MMB and that plugin thing !

PlugIns everywhere

Let's skip that part "it all began 200 years ago..."

...'cause it didn't ;-)

Most of today's computer users got a chance to experience PlugIns, sometimes without even knowing about it.

For a long period of time, extra functionality was handled through various sub-routines built into the main program. So you changed entire program to experience difference. Someone, somewhere thought of a new approach - why not make external units that could be added / changed like bricks. In fact, why not allow people all over the world to make these bricks ?

So they did it - once agreement was made, standards built, documentation & tutorials became available worldwide - didn't have to wait too long to get some attention !

Does it mean we had to wait for standardization to get PlugIns ?

Yep ! Standardization on more levels.

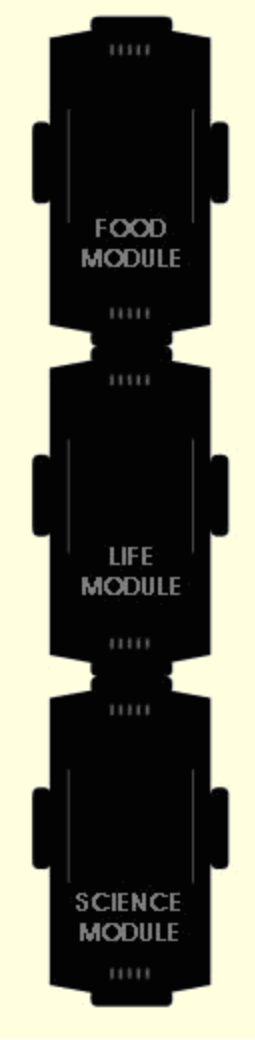
Today you can find PlugIns everywhere - in paint tools, video editing applications, multimedia players, e-mail clients...

But how do they work ? What's the game ?

Space station - collection of plug-ins

You surely did see at least one space station. They fly not so high above our heads, here and there fall and by shape are far from things we use to see in SF movies.

Now imagine MMB being a central structure of the space station:

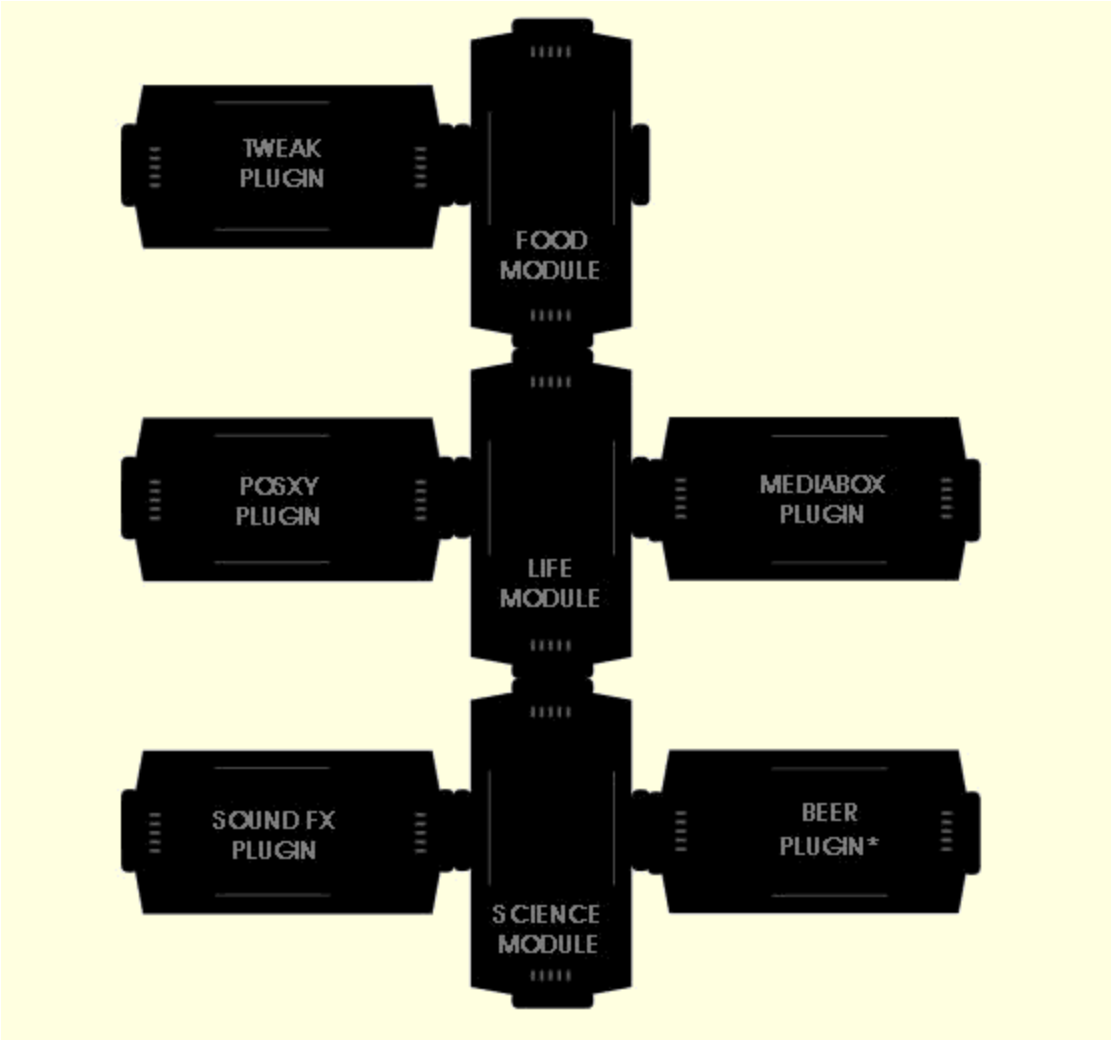


Ok, yeah, this doesnt' look much like a space station without solar cells... More like a composition of fire cannons attached one to another :-)

Anyway. Here we have a MMB Space Station, representing MMB as application itself.

You can live in this station. But after some time you'll feel a need to get some extra furniture. And that furniture will require some space. After all, you want a few more acres of space too, so you can kick off your shoes and enjoy in the fresh air.

Thanks to station gateways - you can ! It's all about going to a resource shop and taking whatever you need. Some are free, some require a small fee ! Cash & carry, easy installation, don't worry !



* Beer PlugIn is under development

Now you have The Station ! More space, more power and it looks a way cooler than your neighbour's !

Rubber gateways

Let's face it - for the very first plugin/module installations to your MMB Space Station you'll call a local plumber.

Don't want to ? Then welcome to:

Small Do-It-Yourself MMB Plumber Operator's Manual

While MMB takes care of all it's rubber gateways, your part of the installation job ain't very difficult if you track pictures of this manual. Yeah. It's such an enormous difference between difficulty of plugin installation and usage !

Ingredients:
- 1 MMB
- 1 PlugIn
- Unlimited amount of coffee

You started MMB, no problem, but how do you recognize MMB PlugIn ?

It's a file like any other, but with weird extension - DLL . Here are some examples of MMB PlugIn file names:

tweak.dll
misc.dll
posxy.dll
EditBoxPlugIn.dll

Names usually represent basic function of PlugIns. It's hard to find some cryptic one like

3ksvow843beer.dll

Got your cup of coffee ?

Good !

Where will you find MMB PlugIns ?

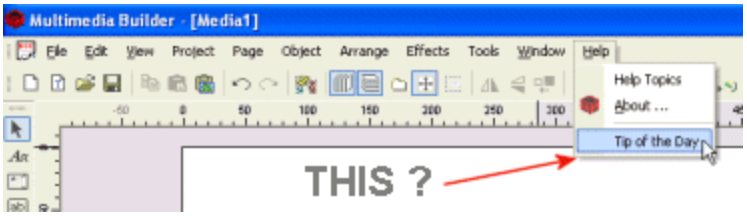
There are various places to start with, but to keep this answer short, recommendations of Operator's Manual are:
- visit www.mmbforums.com where you'll find MMB Community Links index, having good collection of links to sites with PlugIn resources
- visit some of MMB-related sites (called Resource Sites 'cause they hold various MMB community resources):
- MicroTech's MMB Resource Site:
http://www.advanced-microtechnologies.com
- Squash Productions plugins:
http://www.squashplugins.cjb.net/
- Yomo's site (home of Plugy plugin!):
http://www.mmdev.net/yomo/home.htm
- RastaWorld Productions:
http://www.rastaworld.com/
- Bokzy's site
http://www.bokzy.com/

Links might change in time, so if you get 404 try google search.

Back to our MMB plumber business !

You made a nice collection of those DLL's and now you're back to this manual. What now ?

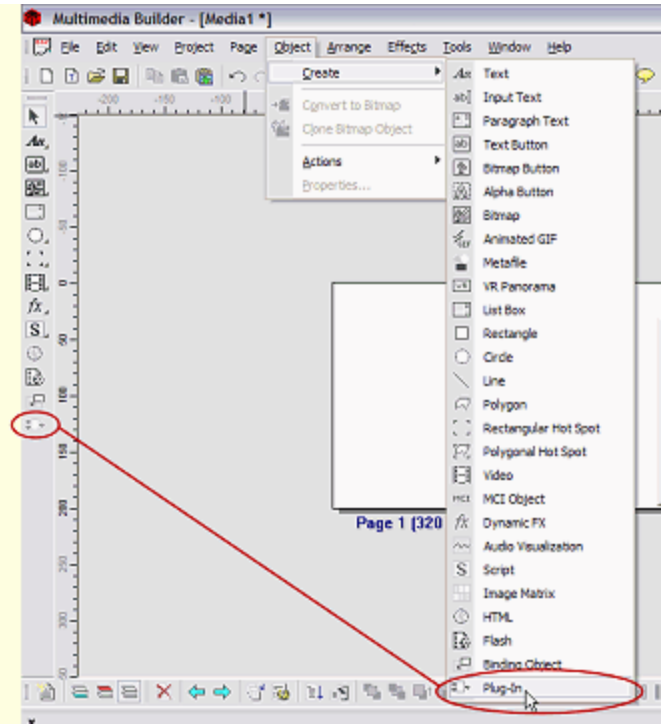
Take a look at MMB window. Let's see what can be found here...



Naaaahhh....

Keep searching.

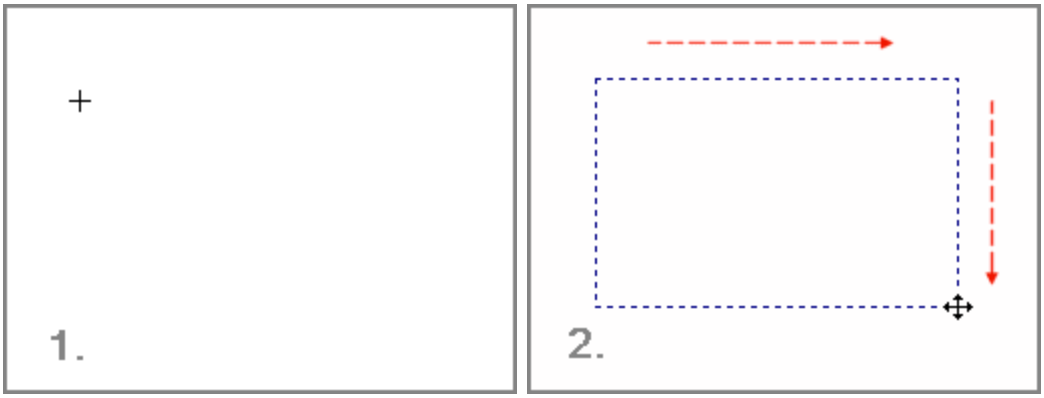
Trying again, and.... BINGO !



Well, more something like "PlugIn" than "BinGo", at the end of object menu list or Object ToolBar..

Click on "PlugIn" item and your mouse pointer will turn into a drawing cross, signalizing you'll have to draw a rectangle that will later be a PlugIn window (if it's one of those visible PlugIns).

You're able to adjust position and size of PlugIn's rectangle later, so don't pay much attention to it now.

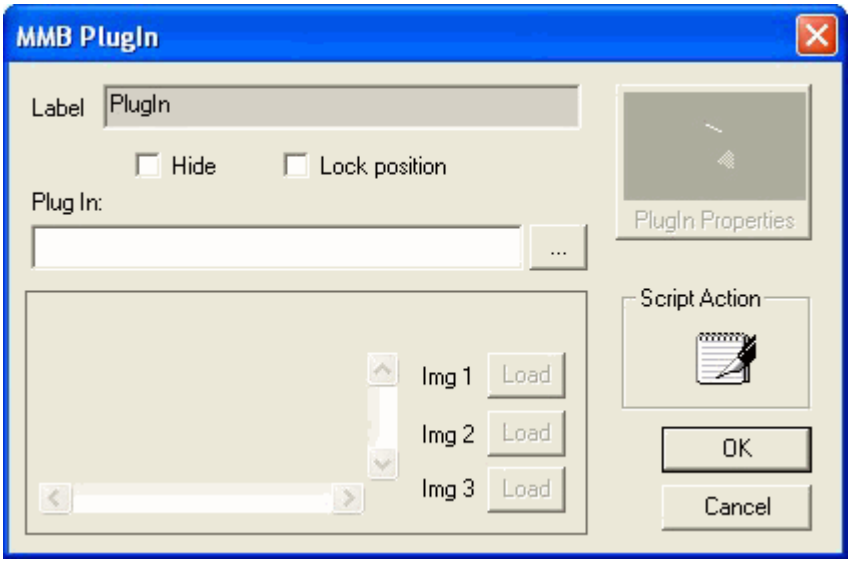


Hey ! This is flat and doesn't look like 3D module for my Space Station !

Well, yeah... you know, it's like one of those Santa Clause things... They tell you the truth later....

At least you have a PlugIn rectangle ! Double click on it.

New dialogbox pops up, it's pretty simple:



Label, Hide & Lock Position are features available for all MMB objects. You should only remember PlugIn Label.

Below this standard features, there's an editbox, titled "Plug In:" .

On the right side of editbox there's "..." button, signalizing you can browse for PlugIn file on your data drives. Push it !

Well-known "Open File" dialogbox is displayed, allowing you to select DLL file - MMB PlugIn.

If you made a collection of MMB PlugIns by downloading 'em from resource / plugin developer sites - you'll go to that location and select a file.

Don't have anything in collection ? Go to MMB's installation folder and then to \Plugins sub-folder. There you'll find small PlugIn provided by Mediachance together with MMB, called TenBlobs.dll

Once you select file and click "Open" button - MMB will ask what it should do with it:

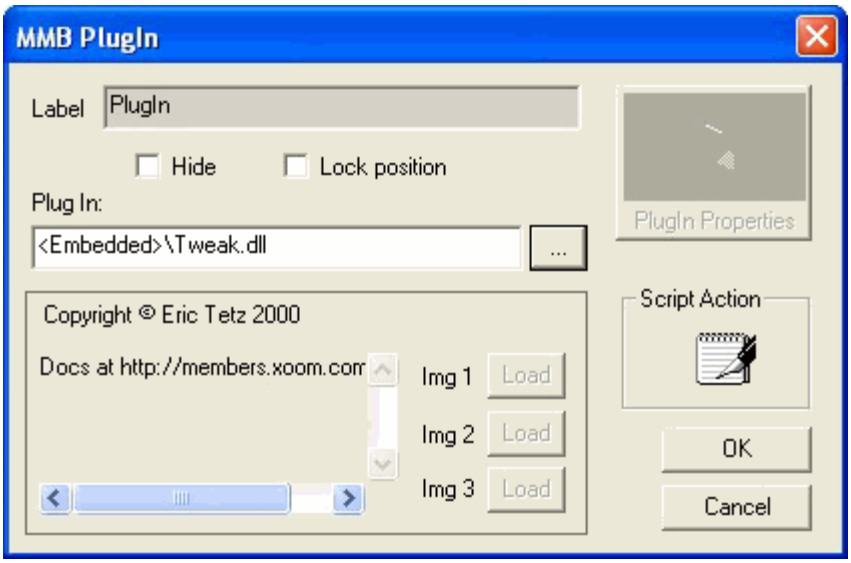
"Do you want to move PlugIn to embedded files ?"

Click -> Yes

In this case, PlugIn will be embedded (attached) to your project. Star Trek fans would use expression "assimilated" ;-)

Once PlugIn is embedded, you won't have to worry about it's location after finishing your application. MMB will pack everything together. Just for notice - it has it's drawback, but more about it later !

After you assimilated MMB PlugIn, dialogbox will look like this:



Path to DLL file contains prefix <Embedded>\

This is called "path macro tag", and in this case signalizes that Tweak.dll is embedded into your project. More about path macros available in MMB's help file.

Below path editbox there's a text memo with comment - place where PlugIn developers put some important info: copyright, author name, support page URL & e-mail, etc.

Click on OK button and that's it ! If you carefully listed Plumber's Manual - installation of MMB PlugIn in your project is done !

Save your project, close everything and come back tomorrow.

Good morning. You have 1 PlugIn object

After adding MMB PlugIn to project, maybe you felt like something's missing. Where are all those dialogboxes and controls well-known from other software PlugIns ?

Hm.

Approach to changing MMB PlugIn properties is somewhat different.

If you try using TenBlobs PlugIn, provided with MMB, you'll notice ability to change some parameters using Properties dialogbox.

All other MMB PlugIns use commands to do it.

Why is that so ?

While using MMB script language and trying some demos from MMB community, you probably figured out it can be used for much wider range of applications than autorun menus and install screens.

Starting from that point, MMB PlugIns offer ability to be controlled from your application. Anytime, anywhere - using MMB's script language.

That's a great advantage !

You can load, save & set different parameters, perform property changes in loops and during runtime mode...everything without the need for end-user interaction.

Imagine how annoying would be, to bother users with property dialogboxes popping up every minute, asking for intervention... ;-)

That's right ! Let's admit it - usage of PlugIns and script language is programming. So we have some serious business here.

Behind the curtain

Let's spend a few words for readers who want to know more on MMB PlugIn mechanics subject.

That's right, technical mumbo-jumbo coming, read carefully.

MMB PlugIns are DLLs - Dynamic Link Libraries. Meaning, they are organized collections of commands / functions.
Remember rubber gateways from MMB Space Station ? PlugIn developers use 'em to communicate with MMB.

:-)

So we're talking about some kind of communication channels here. And some are used just for internal purposes, but for sake of this manual let's point out those important to PlugIn users:

- setting & retrieving of integer variables
- setting & retrieving of string variables
- running of specified command

You must get familiar with string & integer variables to use MMB script language. Still didn't ? Close this manual, open Help in MMB and come back when you're done with variable research.

Here's another order of PlugIn<->MMB communication:

- setting parameters through variables (either integer or string)
- running of PlugIn command (parameter not always necessary)
- getting values from PlugIn through variables (not always necessary)

This order points out general & very important PlugIn usage sequence. More about it later.

Types of MMB PlugIns

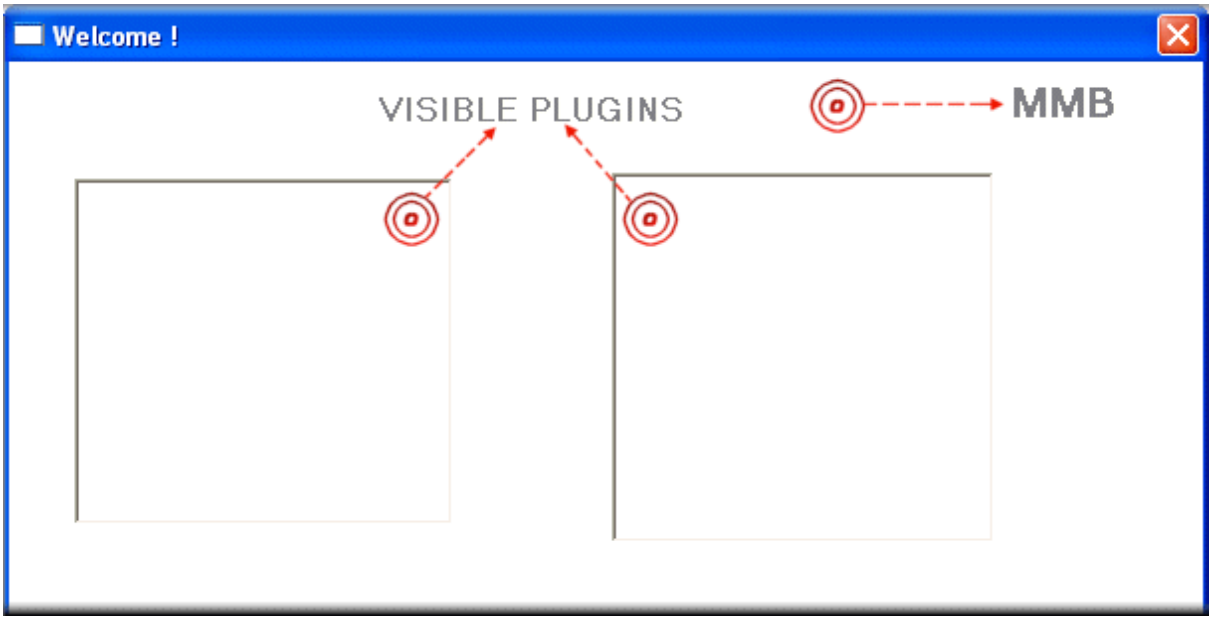
There are two (2) types of PlugIns:

MMB PlugIn Type 1: 98% of MMB PlugIns. They differ from standard applications by the fact they're DLLs communicating with parent (MMB) application through commands ("channels") and their visual elements are drawed inside the defined PlugIn window of MMB-made application.

Everything else is practically independent from parent (MMB) program - all command processing happens inside the PlugIn. Output of executed command either performs change on PlugIn objects or goes to string / integer variable. And that's entire scope PlugIn covers ! No much interactivity with MMB, eh.

This is very important to understand -Type 1 PlugIns work almost independently in their own windows set by MMB, so they cannot affect other MMB objects - but this lack of interactivity also disables covering of Type 1 PlugIn window with other MMB objects.

It's something you'll notice while using visible PlugIns - those with visual objects like EditBox PlugIn, SlideShow PlugIn, etc.



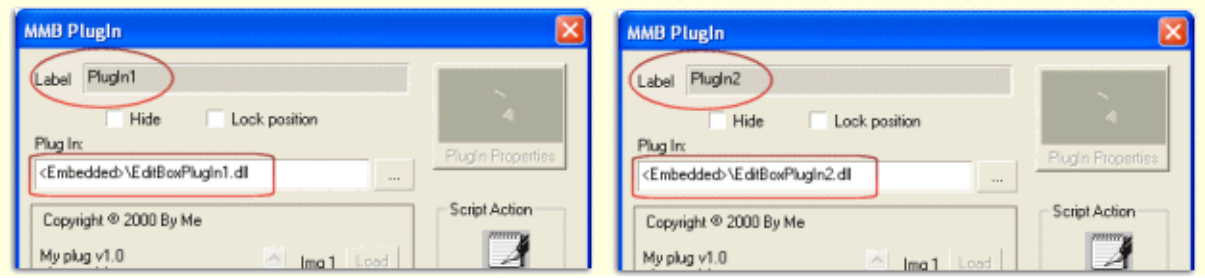
Another thing to keep in mind when using visible Type 1 PlugIns: they can't be run in multiple instances if one DLL is being used.

Resources can't be shared, so if you, for example, need two editboxes performed by EditBox PlugIn - two EditBoxPlugIn.dll files are required. You should rename them and have:

EditBoxPlugIn1.dll

EditBoxPlugIn2.dll

To get two PlugIns work simultaneously, you'll perform adding of PlugIn objects twice, as explained in Plumber's Operator Manual.

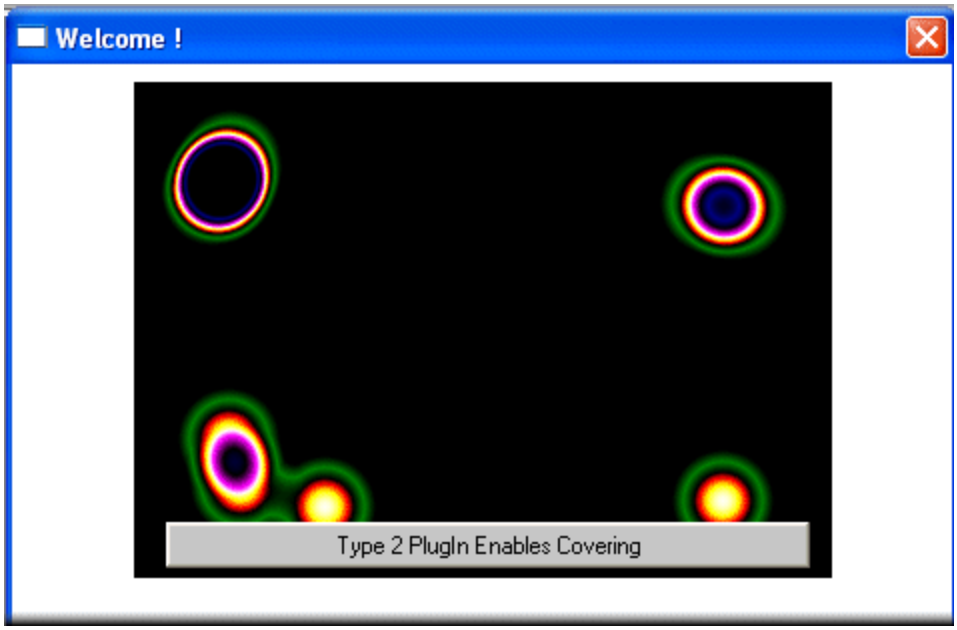


You have two PlugIn objects now. First is labeled "PlugIn1", and second one "PlugIn2". This labels are important while using PlugIn commands, they serve as reference to 'em.

Like when sending mail - you'll need destination address. In this case, you'll need PlugIn's destination - label.

And how do we use PlugIn commands ? That's most important part of this tutorial, coming up soon ! Keep reading ;-)

MMB PlugIn Type 2: this type enables multiple instances of visible PlugIns and covering with MMB objects. But it proved to be unstable, so it's rarely used. MMB comes with Type 2 PlugIn example, called TenBlobs.dll



In front of the curtain

Phew ! That previous chapter was a way too serious.

What have we learned so far ?
- PlugIns are everywhere
- They are additions to main products
- Their purpose is increasing of main product capabilities
- MMB PlugIns are not easy to install
- Neighbour's project is still more powerful

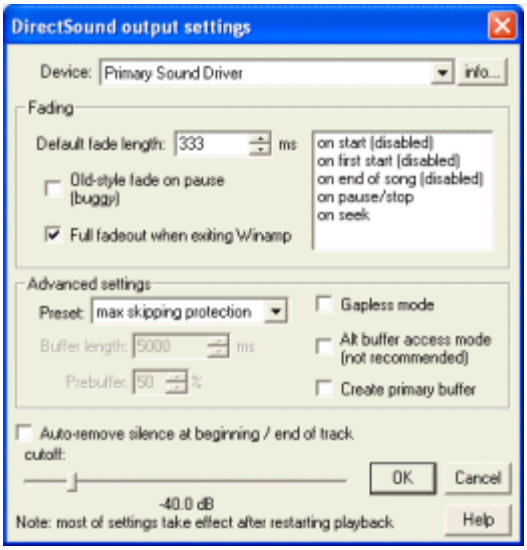
No more time to spend on theory ! We must move to practice !

ASAP !

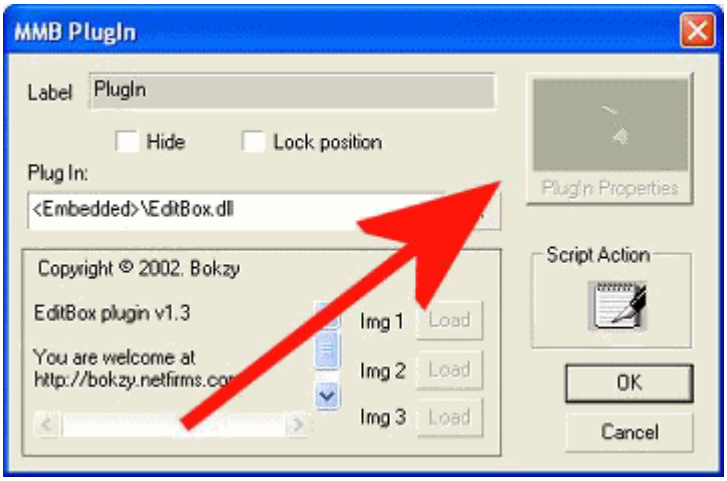
After crawling through installation of MMB PlugIns, one would probably ask:

OK, what's the story now ? They're not like space station modules (and you can live with that), but also not like audio plugins in WinAmp, graphics plugins in Photoshop and they're far away from Premiere video plugins !

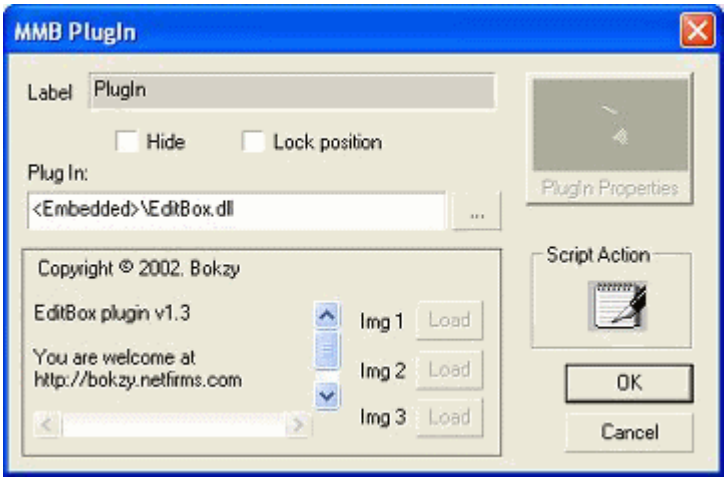
Opening of audio plugin properties in WinAmp will display something like this:



...but if you try something like that in MMB PlugIn Properties window...



...in 98% of PlugIns you will get:



NOTHING !

What now ?

- Hm. Seems like we'll have to change MMB PlugIn properties somewhere else.

Where ?

- In MMB Script Window, using MMB Script Language !

Visiting another planet

If your first impression on seeing this window was "What in heck is that ?!?" - you will unfortunately have to leave MMB PlugIn Operator's Manual at this point, take MMB Help or some other MMB General / Scripting Tutorial and start your research on

MMB Script Language

It's a foundation of using MMB PlugIns - without knowing how variables work or where to use them - you would pull half of your hair out by reading this tutorial any further.

And it's not like you must become an expert in this script language to start using PlugIns !

The most important scripting parts are:

- working with integer and string variables
- structure of MMB script language commands
- using functions (if-then statements, for-next loops) & path macro tags (<SrcDir>, <Embedded>, etc.)

The Practice

No longer can this Manual save readers from ugly business !

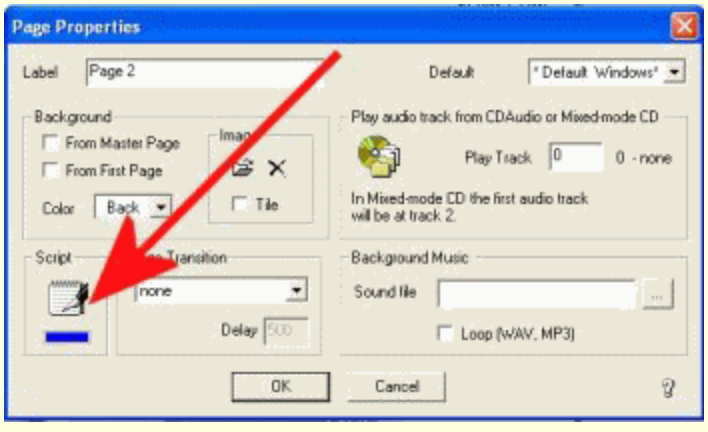
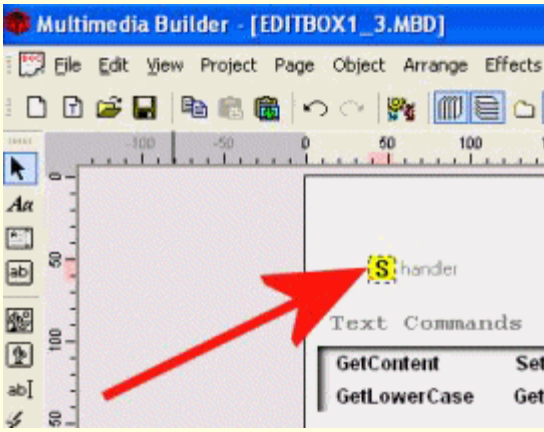
We're now going right to the center of Ugly PlugIn Beast?

Fasten your seat belts.

LAW & CONSTITUTION OF MMB PLUGINS

There are 3 (three) PlugIn-related commands you use in MMB script language to communicate with PlugIns !

You enter these 3 commands in MMB script window, either at:

Script Icon of MMB Object	or..	MMB Script Object
		

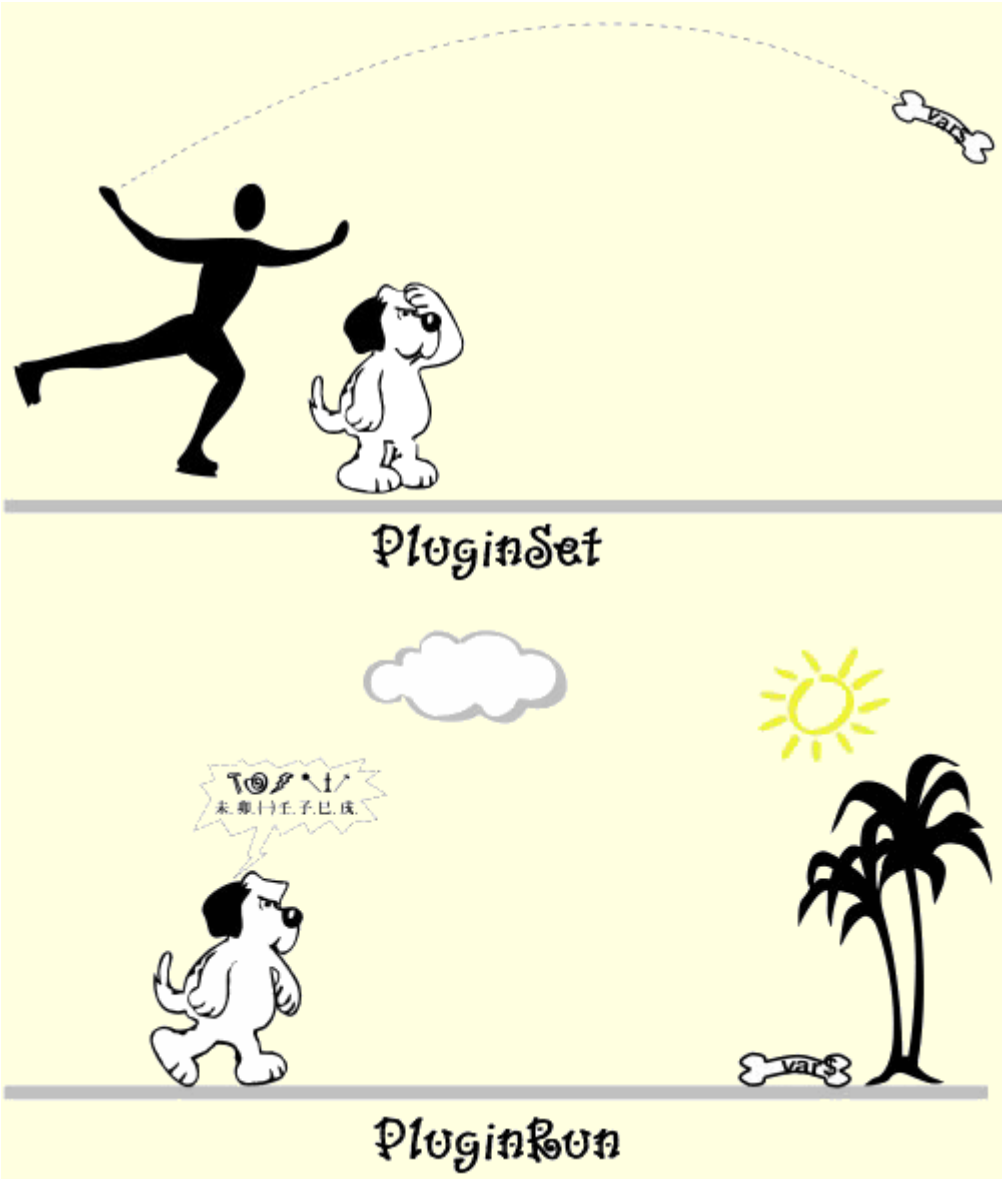
So we're talking about

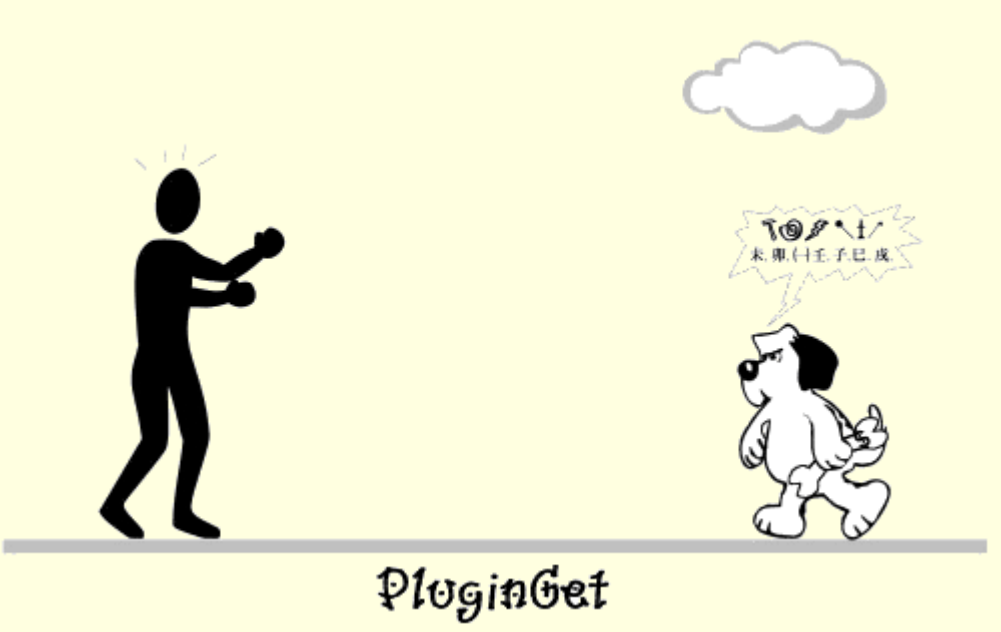
FELLOWSHIP OF 3 COMMANDS

PluginSet	MMB script command being used for **setting of parameters**. Once set, parameters are "memorized" in PlugIn. And PlugIn will use them for running of command (specified using PluginRun script command). What are these parameters ? MMB string and integer variables, containing some value. If you use PluginSet twice, previously "memorized" parameters will be substituted with new ones. So it's not some kind of container you'll fill repeatedly, and expect it to hold everything ;-)
PluginRun	MMB script command that is used for **running of PlugIn command**. To know what commands are available, refer to Documentation / Help file / Tutorial / Readme file that (in most cases) comes with PlugIn. If you're not able to find any reference, try contacting PlugIn author or visit MMB Forums, searching for help on this subject. Most of commands either require input parameters through PluginSet or give output you're able to retrieve using PluginGet. Executed commands: 1. **do not always require input** parameters set using PluginSet command 2. **do not always give output** retrievable using PluginGet command 3. sometimes change only PlugIn's status
PluginGet	MMB script command that is used for **retrieving of parameters**. PlugIn command executed using PluginRun can return some result (either into string or integer variable). With PluginGet you're able to get this content, so you'll naturally use it after PluginRun.

Fido - PlugIn Dog

Pictures are worth more than thousands of words, so let's now see how good old Fido deals with 3 MMB PlugIn commands !





Although our Fido doesn't look happy about his task, one thing is for sure - PlugIns won't complain and give you strange looks, no matter how many times you repeat 3 command steps, no matter how many bones you throw ;-)

Scripting versions

You must have thought "Yeah, right, no way these commands can work just like that, without anything attached to 'em".

And you're right - like most of MMB script commands, these also use brackets and quotes to enable setting of parameters.

PluginSet	
In Theory	**In Practice**
PluginSet	**PluginSet("PlugIn","var$")**

Explanation:
1. **PluginSet** - command that will set some parameters to be used by PlugIn later, through PluginRun command
2. **"PlugIn"** - label (inside quotes) of PlugIn object you refer to. Just as you would write address on letter envelope before sending, this part will tell MMB where to send parameters (in this case to PlugIn object labeled "PlugIn" ; but you can change that label by double-clicking on PlugIn object in MMB.)
3. **"var$"** - name of variable (inside quotes) that contains parameter(s) you want PlugIn to use later through PluginRun command. This variable can be either of integer or string type - and to know what type you should use - refer to

Documentation of PlugIn, where this important info will be mentioned in description or command example. Every PlugIn command can either
3.1.use *integer input* variable (for e.g. var),
3.2.*string input* variable (for e.g. var$) or
3.3.*do not use entire PluginSet command* at all (if it's only output-related, meaning, doesn't require any input).

PluginRun ain't different in structure. Only noticable change is third part of command, where you do not specify variables, but run actual PlugIn commands:

PluginRun	
In Theory	**In Practice**
PluginRun	**PluginRun("PlugIn","UpperCase")**

Explanation:
1. **PluginRun** - command that will run (now this is fuzzy, check this out:) - PlugIn command !

Command that runs command ? Yep.

2. **"PlugIn"** - label (inside quotes) of PlugIn object you refer to. Just as you would write address on letter envelope before sending, this part will tell MMB where to send parameters (in this case to PlugIn object labeled "PlugIn" ; but you can change that label by double-clicking on PlugIn object in MMB.)
3. **"UpperCase"** - actual PlugIn command that will be run upon calling PluginRun. How will you know what commands to use ? Refer to Documentation of PlugIn or some of it's demos - you'll also find descriptions and examples of PlugIn command usage, with all necessary input and/or output variables for PluginSet & PluginGet commands.

To clear this up: here we have two kinds of commands:
2. script command PluginRun (MMB's script command)\
3. PlugIn command that is run using PluginRun (and you'll have to read PlugIn documentation to see how these commands are named)

As you would run any program - once MMB executes PluginRun, PlugIn will do it's job. And if obliged will return output integer and/or string variable that you're able to retrieve using PluginGet script command.

Similar to PluginGet, this one also uses variables as command's third part, but this time to receive a value - from PlugIn !

PluginGet	
In Theory	**In Practice**

PluginGet	**PluginGet("PlugIn","var$")**

Explanation:

1. **PluginGet** - command that will retrieve a value from PlugIn after it's command has been run using PluginRun
2. **"PlugIn"** - label (inside quotes) of PlugIn object you refer to. Just as you would write address on letter envelope before sending, this part will tell MMB where to send parameters (in this case to PlugIn object labeled "PlugIn" ; but you can change that label by double-clicking on PlugIn object in MMB.)
3. **"var$"** - name of variable (inside quotes) that will be filled with contents received from PlugIn after PluginRun command. This variable can be either of integer or string type - and to know what type you should use - refer to Documentation of PlugIn, where this important info will be mentioned in description or command example. Every PlugIn command can either
   3.1.use *integer output* variable (for e.g. var),
   3.2.*string output* variable (for e.g. var$) or
   3.3.*do not use entire PluginGet command* at all (if it's only input-related or doesn't use any variables / parameters at all).

Put it, shake it, get it - vÔila ! Virtual example 1 !

Now we'll extract important lines from tables above, put 'em one after another and we'll get somethin' like...

```
PluginSet("PlugIn","var$")
PluginRun("PlugIn","UpperCase")
PluginGet("PlugIn","var$")
```

Here we are ! First block of PlugIn script lines has been completed !

Didn't hurt much, but what this particular code does ?

It's an example. Of some virtual PlugIn (object is labeled as default one would be - "PlugIn"), in service of script line block above, that

- takes input string variable (var$) with PluginSet,
- executes PlugIn command named "UpperCase" with PluginRun,
- retrieves a value from PlugIn after PluginRun and fills string variable (var$) using PluginRun

This PlugIn command serves as a good example - uses input variable to set some value to PlugIn for processing, runs PlugIn command and retrieves processed content to some variable.

Let's extend this script lines with a few more, MMB-related:

```
var$ ='Welcome home, Joe'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","UpperCase")
PluginGet("PlugIn","text$")
```

Message("MMB says:","text$")

Now we already have entire program ! It fills var$ string variable with "Welcome home, Joe" text, assigns that content to PlugIn (with PluginSet), runs PlugIn command UpperCase (with PluginRun), retrieves result into text$ output string variable (with PluginGet) and at the end - displays processed text in MMB's MessageBox !

It's about time to mention what actually UpperCase command from this PlugIn does - it converts input text to upper-case version and brings it back to MMB. Alive and upper-cased :-) So text "Welcome home, Joe" is converted to screaming "WELCOME HOME, JOE" :-)

Virtual Example 2 - Input without Output

We'll take our virtual, non-existing PlugIn once more and see what script code lines will be used if input through variables is required - but there's no output (PlugIn doesn't tell anything after PluginRun).

When do we need this ? Usually when input variable is just a parameter that will be 'consumed' by PlugIn - either to change some color, increase / decrease font size, start some important command in PlugIn (like Connect, EncryptFile, SetMute) or even to send message over the Internet or set parameter for some other PlugIn command (for e.g. setting of source file can be done with command like SetSourceFile, setting of destination file with similar SetDestinationFile, and after using both of 'em you'll use third PlugIn command - CopyAllFiles).

Needs input, but we can't expect output (more precisely, output through integer/string variable). Script commands that will be used to perform this are:
- PluginSet
- PluginRun

Having virtual PlugIn, let's now imagine browsing through it's documentation ! Our task is - finding description and example of some command that will show a Message Box. While this PlugIn can display Message Box, let's use it !

Searching through PlugIn's docs takes us to description of something that could serve this purpose:

Virtual PlugIn Documentation

General Commands

Command name: **DrinkCoffee**

Description: Makes your computer walk to the kitchen, drinks a coffee and returns to you fresh and ready for another portion of heavy work in MMB.

Example:
```
var$ ='Black,Milk,Sugar'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","DrinkCoffee")
```

Hmm.... well that could do it, but Message Box still won't be displayed :-)

Continue exploring strange jungle of PlugIn Documentation....

Virtual PlugIn Documentation

General Commands

Command name: **ShowMessageBox**

Description: Does not walk or drink coffee. This command shows Message Box with OK button. Use input string variable to specify message you want to display in Message Box.

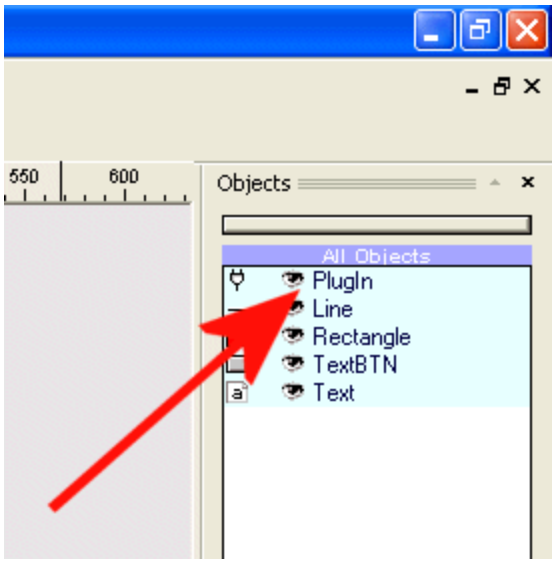
Example:

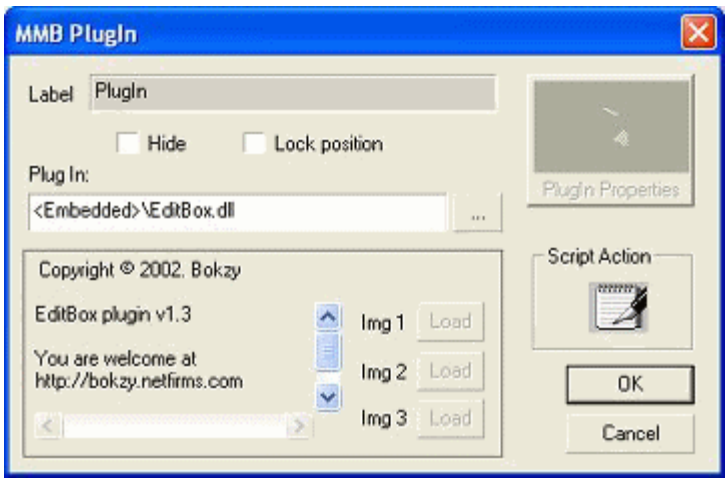
```
var$ ='I am not coffee addict, I swear !'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","ShowMessageBox")
```

Yeah, looks like it !

As you can see from this frames - documentation has it's title (Virtual PlugIn Documentation), subtitle that specifies command range (in this case - General Commands) and listings of commands with their descriptions and examples.

Most of MMB PlugIn docs presume that PlugIn's object in MMB will be called "PlugIn". So you should check it's label and adjust either PlugIn object label or script code lines.





After reading description of ShowMessageBox command, you decide to use provided example. By simple copying from PlugIn documentation and pasting into MMB's Script Window, you'll get:

```
var$ ='I am not coffee addict, I swear !'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","ShowMessageBox")
```

Let's get familiar with these lines...

First line assigns text to string variable named var$

```
var$ ='I am not coffee addict, I swear !'
```

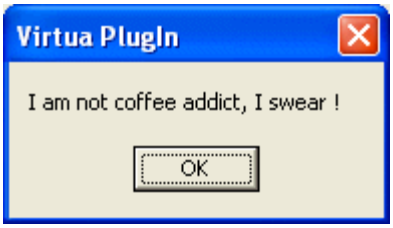
Second line uses PluginSet to assign var$ to PlugIn

```
PluginSet("PlugIn","var$")
```

And finally - we'll call PluginRun to tell PlugIn: "Hey,PlugIn , it's all set and now it's time for you to run ShowMessageBox command."

```
PluginRun("PlugIn","ShowMessageBox")
```

And the result ? We'll see it on the screen !



Notice: you won't always be able to copy/paste all command examples from documentation and expect 'em to work without adjustments. They can depend on some other commands or scripts, you can have different labels or requests from these commands - examples usually show basic functioning of PlugIn commands and it's on you to get the best from 'em (for e.g. improve the script above in the way your users can enter message in an EditBox and by clicking on the button display it using PlugIn's ShowMessageBox).

Virtual Example 3 - Output without Input

While exploring hundreds of commands through documentation of various PlugIns, you'll bump into this type of PlugIn commands - they don't require any input in the form of string or integer variable - but they are run using PluginRun and you can pick result of their work, using PluginGet script command.

What are commands of this type for ?

Mostly for retrieving:
- information from PlugIn
- information from the system

In most cases you'll use it to pick information, like system date & time, read INI files, available RAM memory, read Windows Registry, find out about mouse position or pressed keys, get media file info, etc, etc....

Let's pick a date !

Back to PlugIn docs !

Virtual PlugIn Documentation

Info Commands

Command name: **GetSystemDate**

Description: This command retrieves current system date and puts result into output string variable.

Example:

```
PluginRun("PlugIn","GetSystemDate")
PluginGet("PlugIn","date$")
```

We'll naturally copy/paste this example into MMB's Script Window.

```
PluginRun("PlugIn","GetSystemDate")
PluginGet("PlugIn","date$")
```

Notice a difference - there's no input variable anymore - we'll immediately use PluginRun to tell PlugIn "Hi again, PlugIn , sorry for bothering you, but could you please pick up current date from this system with that GetSystemDate you got there ?".

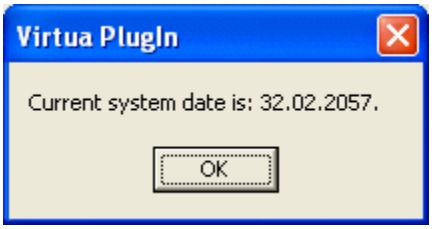
With PluginGet comes another instruction, saying to PlugIn:

"Doh, almost forgot - put the date into date$ so I can use it in my beautiful MMB project !".

You'll make adjustments in this example, of course, so retrieved date will be, if nothing else, displayed in MMB's MessageBox:

```
PluginRun("PlugIn","GetSystemDate")
PluginGet("PlugIn","date$")
Message("Current system date is:","date$")
```

Let's see the result:



Similar example to this one can use command like GetSystemYear to retrieve current year into integer variable:

```
PluginRun("PlugIn","GetSystemYear")
PluginGet("PlugIn","year")
Message("Current year is:","year")
```

Here, the result of PluginRun , that executed PlugIn's GetSystemYear command, went into integer output variable year

Once retrieved into integer variable, you can perform various math operations available in MMB's script language.

Virtual Example 4 - No Input - No Output ?

Yep, you'll find these too ! Strange kind of commands.... they ask for nothing....they give nothing. What are they for then ?

Well - for things that simply need no assistance through parameters and don't feel need to give you any result !

And there are plenty of them - Play, Stop, Show, Clear...

Another visit to PlugIn documentation, this time we'll look for a command that will show PlugIn window (let's presume this PlugIn is of visual type, having visualizations for which would be a shame not to have visibility in MMB application).

Virtual PlugIn Documentation

General Commands

Command name: **Show**

Description: This command shows PlugIn window and it's contents.

Example:

```
PluginRun(" PlugIn","Show")
```

Again we get this code example into MMB Script Window:

PluginRun("PlugIn","Show")

Now this one is easy ! No messing around with variables, you just call PluginRun and it executes PlugIn command Show !

With Show command in PlugIn also usually comes Hide command that will do the opposite - it will hide PlugIn window.

Also no variables needed, you just run that one sweet line:

PluginRun("PlugIn","Hide")

Combining Examples 1 & 2

We're moving to advanced example, that combines PlugIn's command types 1 & 2 - meaning, we'll

- use UpperCase command with all around it to convert text into upper-case version, and use the result to
- set input string variable for ShowMessageBox command

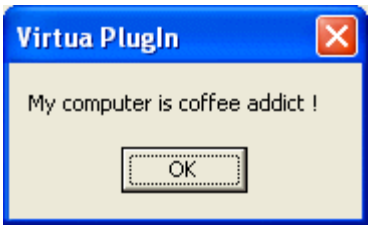
```
var$ ='My computer is coffee addict !'
PluginSet("PlugIn","var$")
```

```
PluginRun("PlugIn","UpperCase")
PluginGet("PlugIn","text$")
PluginSet("PlugIn","text$")
PluginRun("PlugIn","ShowMessageBox")
```

Check that out ! Entire sequence of commands that starts with

var$ ='My computer is coffee addict !'

...and ends with...



There's no limitation in number of sequences you can run in this way - and it's very neat to run 'em in loops.

PlugIns & Variables

Let's spend some time with examples that will prepare you for various aromas of input & output variables used with PlugIns.

Example with basic string input variable

In action we'll see already used virtual PlugIn command for displaying message box, called ShowMessageBox.

With input string variable var$ you set text that will be displayed in PlugIn's message box:

var$ ='Live long and prosper !'

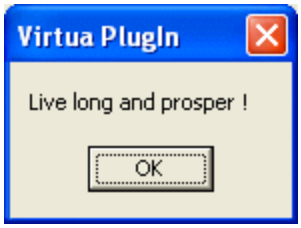
Using PluginSet command, content of var$ will be assigned to PlugIn:

PluginSet("PlugIn","var$")

And by calling PluginRun, message box will be displayed:

PluginRun("PlugIn","ShowMessageBox")

Result is visible on screen:



Examples with advanced string input variable

With normal strings, you'll set text either in word or sentence form. What exactly would be advanced string, then ?

It's a kind of input string that is required by PlugIn, to set it's property / setting.

In fact, there are two versions of PlugIn property/setting input strings...
- upper-cased property strings
- comma-delimited setting strings

Here's what makes difference between these:

Upper-cased property strings

As you guess, it has something to do with upper-case written words. What will you use it for ?

To set PlugIn's properties like window border and mouse click events. They require (usually single) word inputs as arguments that will set PlugIn's property after calling specific command. And upper-case words are convention used within MMB PlugIns.

This example will set single-line border on virtual PlugIn window, using word SINGLE, as input string parameter, and SetBorder PlugIn command to assign input parameter.

```
var$ ='SINGLE'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetBorder")
```

Authors of PlugIns mention available string input parameters in PlugIn's documentation, so you don't have to take a guess ;-)

Comma-delimited setting strings

Comma-delimited string = describes advanced string that uses comma , as border between two words.
Setting string = kind of string that will change PlugIn setting

Two most usual (and important) cases where you'll use comma-delimited setting strings are:
- Setting of background / foreground color values
- Setting of event handlers

About event handlers and colors read more below Manual's example section. This is quite important subject for comfortable life with MMB PlugIns, so it's recommended to jump over and read about these subjects right now.

Example of comma-delimited string in service of background color setting using virtual PlugIn's command SetBackColor coming up:

```
var$ ='10,201,20'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetBackColor")
```

After running this example, background color of virtual PlugIn window would look like this:

While we're here, let's see example of comma-delimited string in service of event handler setting using virtual PlugIn's command SetHandler:

```
var$ ='CTRL,SHIFT,ALT,U'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetHandler")
```

The result of setting event handler to this key combination would be demonstrated after event occurs - CTRL, SHIFT, ALT and U keys would be triggered and passed to your application.

Examples with integer input variable

Another commonly used input variable type is integer - there are numerous PlugIn commands what will use exactly this type to set volume, font size, media position, number of seconds, etc, etc...

To keep it simple, this example will set PlugIn's font size.

With input integer variable var you set TrueType font size:

```
var =18
```

Using PluginSet command, value of var variable will be assigned to PlugIn:

```
PluginSet("PlugIn","var")
```

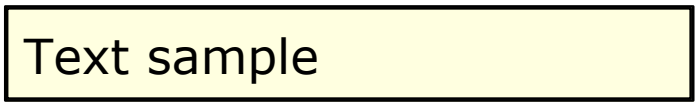
And by calling PluginRun, font size will be applied using virtual PlugIn's command SetFontSize:

```
PluginRun("PlugIn","SetFontSize")
```

Here's how code looks like if we put it together:

```
var =18
PluginSet("PlugIn","var")
PluginRun("PlugIn","SetFontSize")
```

And the result is PlugIn's font size set to 18:



Another standard integer input is setting of PlugIn's sound volume. Available values in this case are usually in range 0-100 and PlugIns will ignore any lower or greater values. For this example, SetVolume command of virtual PlugIn is being used to assign value from var integer variable and adjust PlugIn's sound volume.

```
var =18
PluginSet("PlugIn","var")
PluginRun("PlugIn","SetVolume")
```

(sound speakers and media players are not provided with this manual, so try to imagine the result of this script code)

Example with basic string output variable

Almost all PlugIn commands that return strings will give you result using basic string output type. The result will be ordinary word, couple of words, sentence, etc.

You already experienced string outputs in UpperCase and GetSystemDate command examples. Right after using PluginRun you'll call PluginGet to retrieve string - result of PlugIn's work.

Using virtual PlugIn's command GetComputerName:

```
PluginRun("PlugIn","GetComputerName")
```

...script command PluginGet will retrieve basic string output into var$ string variable:

```
PluginGet("PlugIn","var$")
```

Once retrieved, you can do whatever you want with it. This example will show computer's name using MMB's message box:

```
Message("Computer name is:","var$")
```

Code put together:

```
PluginRun("PlugIn","GetComputerName")
PluginGet("PlugIn","var$")
Message("Computer name is:","var$")
```

Here's result displayed in MMB's message box:

Examples with advanced string output variable

This kind of string output is being used in two cases:
- PlugIn's event messages
- character-delimited strings

Let's get to:

PlugIn's event messages

Some paragraphs above this manual redirected you to sections that explain event handling and color properties. You did read 'em before coming back, didn't you ?

Good ! Then you know about PlugIns returning string or integer event messages, informing you what exact event has occured.

Once event happens, script object set to catch key combination, sent by PlugIn, will use PluginGet to retrieve output variable. This paragraph is dedicated to string output variables, so in this case string output will be retrieved:

PluginGet("PlugIn","var$")

What will var$ be filled with ? PlugIn returns event message, so if it just finished drinking his beer, returned string might be

I just finished my third beer. Buurp.

Once this event message is sent to var$ output string variable, you'll either display it using some text object or use if-statements to decide what to do in the case of a third beer:

```
If (var$ ='I just finished my third beer. Buurp.') Then
Exit()
Return()
End
```

It's important to follow exact letter cases and spaces - MMB is case-sensitive, so your if-statement string must exactly match PlugIn's event message. Example above won't work if you use:

if (var$ ='i just finished my third beer.buurp.') then

...because upper-case letters and one space are missing.

Character-delimited strings

Don't worry. This type is rarely used.

PlugIn will return output string variable that consists of multiple strings, delimited with character (set by PlugIn's author).

With this string type (something like array) PlugIn can return multiple items without using PluginRun over and over again, but to separate individual strings you'll need some extra coding - so there are just a few cases when PlugIn will return this string output type.

Once PluginRun is executed, you will use PluginGet to retrieve output string variable:

PluginGet("PlugIn","var$")

What will var$ be filled with in this case ? Character-delimited string that will look like:

dog|cat|horse|snake|rabbit|mouse

This example uses | character as delimiter (separator). Of course, it can be some other character:

dog*cat*horse*snake*rabbit*mouse

...and to know what character is being used in PlugIn (in some cases you can even set your own delimiter character), check out PlugIn's documentation.

Examples with integer output variable

PlugIn commands will frequently give you result using integer (number) output type. From count & calculation results to position status and sizes in bytes - all numerical outputs go into integer output variables.

So far you experienced integer output in GetSystemYear command example. Right after using PluginRun you'll call PluginGet to retrieve integer - result of PlugIn's work.

Ordinary integer output example

Using virtual PlugIn's command GetRamCapacity:

PluginRun("PlugIn","GetRamCapacity")

...script command PluginGet will retrieve integer output (capacity of RAM in bytes) into var integer variable:

PluginGet("PlugIn","var")

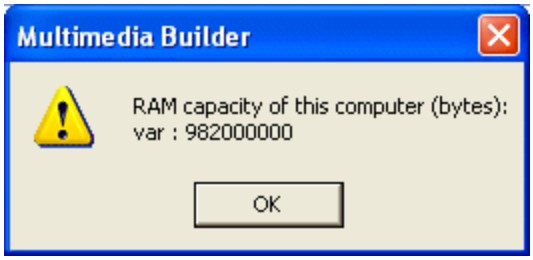
Once retrieved, you can do whatever you want with it. This example will show computer's RAM memory capacity in message box:

Message("RAM capacity of this computer (bytes):","var")

Code put together:

```
PluginRun("PlugIn","GetRamCapacity")
PluginGet("PlugIn","var")
Message("RAM capacity of this computer (bytes):","var")
```

Here's the result displayed in MMB's message box:



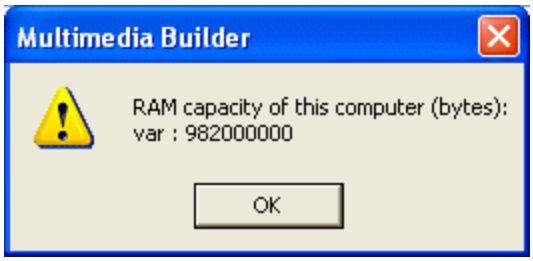
This was raw example of integer output, automatically displayed (used) in MMB.

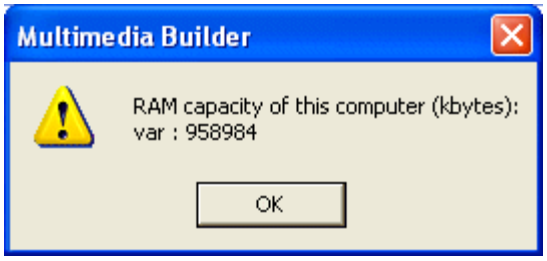
You'll take advantage of numerical variable type, to perform math operations on the result retrieved from PlugIn.

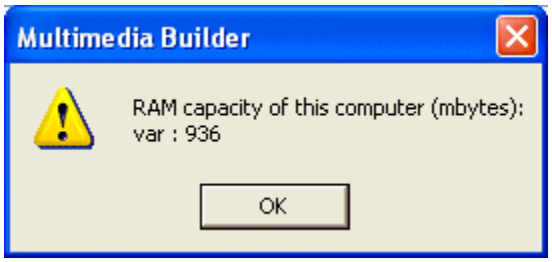
Example above, if enhanced with MMB's script math functions, can look like this:

```
PluginRun("PlugIn","GetRamCapacity")
PluginGet("PlugIn","var")
Message("RAM capacity of this computer (bytes):","var")
var =INT (var/1024)
Message("RAM capacity of this computer (kbytes):","var")
var =INT (var/1024)
Message("RAM capacity of this computer (mbytes):","var")
```

Results displayed in MMB:







PlugIn's event integer output example

Just like in advanced string output case, PlugIns might send you integer output as event.

When will you see that happening ? If something is in progress (like media file playback, local or network file transfer) and PlugIn can send you current progress value - it will be sent as event to output integer variable. You'll probably use it for progress bars / gauges in MMB, resized upon PlugIn's progress event.

Still didn't read about event handling ? Go to sections below Manual's examples and read about it !

Here comes example for virtual PlugIn's event integer output, in this case for file transfer progress.

Once event happens, script object set to catch key combination, sent by PlugIn, will use PluginGet to retrieve output variable.

This paragraph is dedicated to integer output variables, so in this case integer output will be retrieved:

PluginGet("PlugIn","var")

What will var be filled with ? Virtual PlugIn returns file transfer progress event, so if percentage of transfer is 50%, value will be:

50

You didn't expect it would be something else, heh. Once this event message is sent to var output integer variable, you'll either display it using some text object, resize some MMB's object (for e.g. Rectangle as gauge) or use if-statements to decide what to (if percentage is 100, for e.g.)

This code will pick integer output (progress percentage) and resize MMB's object labeled

Rectangle:

```
PluginGet("PlugIn","var")
MoveObject("Rectangle","10,10,var,20")
```

To do something after file transfer through PlugIn is over, you'll use code similar to this example:

```
PluginGet("PlugIn","var")
If (var =100) Then
NextPage()
Return()
End
```

Some MMB PlugIn may contain movable control inside. So if PlugIn returns integer output as event upon control position change, code might look like this:

```
PluginGet("PlugIn","var")
Message("You changed equalizer to:","var")
```

Event handling

Events = happenings, developments that occur in MMB PlugIn and your application is being informed about 'em through:

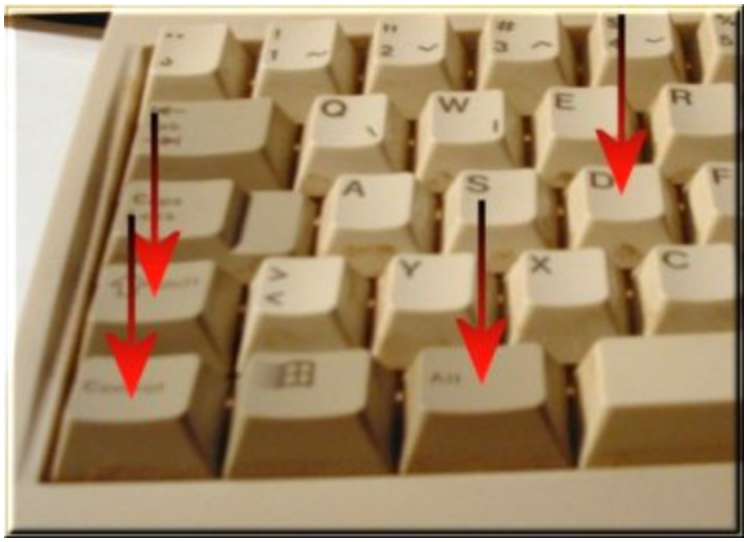
Event handlers = MMB Script Objects that serve as a kind of the "net", catching events sent by PlugIns.

How this all works ?

As you see from examples in this manual, by using PlugIn commands you can receive output. But, in some cases, PlugIn can't return output right after PluginRun.

There are tasks with:
- extensive data processing (coding/decoding, encryption/decryption, copying, loading, etc.),
- prolonged waiting (network connections, system events, user interactions, etc.) and
- periodic events (playback position changes, system time changes, etc.)

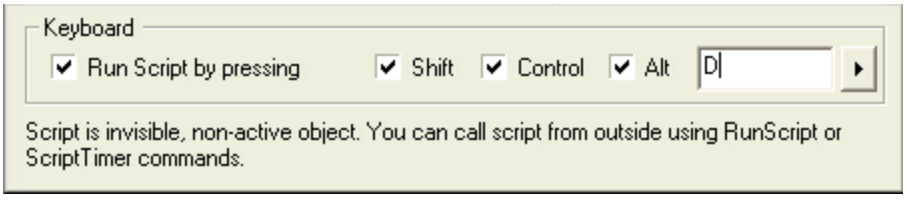
So how will MMB PlugIn react when some delayed event (result) in PlugIn occurs ? It will simulate key presses (individual or combinations of more keys) and send it to your MMB application !



To make things more adjustable, MMB PlugIns allow setting of key combinations that will be used. More about it later.

This scheme shows you sequence (procedure, path) of event being sent from PlugIn (1), through emulating key combination (2) and MMB Application (3) to Script object set to receive pressed key combination (in this case: CTRL+SHIFT+ALT+D):

That's it ! Key combination sent from PlugIn triggers MMB's Script object, set to catch exactly that keystrokes:



Now you know what are Event Handlers - script objects set to receive specific key combination (as image above shows). They are being triggered when event occurs, so all script lines inside will be executed. Of course, you can do whatever you want in these script lines, but it's usual to pick some output value (PlugIns will always give some output after event occurs) using:

PluginGet("PlugIn","var") - for integer outputs

PluginGet("PlugIn","var$") - for string outputs

These are usually first lines in event handler script objects.

Why ? Output values might change if you interact with PlugIn using some other commands before picking up event (string or integer) output, so it's preferrable to do it as soon as event occurs.

Scheme above uses key combination CTRL+SHIFT+ALT+D .

PlugIns allow you to set your own combinations, using PlugIn command with advanced string input. Remember that section explaining various PlugIn commands with strange variable inputs and outputs ? That's right, we'll use advanced string input type with PlugIn command to set key combination for event handling !

Of course, you'll have to refer to PlugIn's documentation and see exact name of command being used for this task. So for purposes of this example, we'll (again) use virtual PlugIn with even more virtual command SetHandler:

First of all, use string input variable var$ to specify keys that should be used for event triggering. Being advanced string, it'll use comma ,to separate key names. First three keys are system keys (CTRL - Control key, SHIFT - Shift Key, ALT - Alt key), fourth can be any alphanumeric key.
Caps should be used for all key names, especially for fourth (alphanumeric) key - after all - MMB's Script objects set & catch only upper-case characters.

So. First script line will assign key names:

var$ ='CTRL,SHIFT,ALT,U'

Maybe you don't wanna use all system keys ? Just leave commas in place where system key won't be used:

var$ ='CTRL,,ALT,U'

String above leaves out SHIFT key.

var$ =',SHIFT,ALT,U'

And this string above leaves out CTRL key.

var$ =',,,U'

With this string, PlugIn ain't using system keys at all ! Just U character. But using only alphanumeric keys should be avoided if your application accepts any inputs from it's users - otherwise, U might be very popular character in your app ;-)

Notice 1: alphanumeric (fourth) key is obligatory.

Notice 2: specified key combinations should not be occupied by various task schedulers, automation tools and similar - these tools can register key combinations globally in operating system, so author of this manual recommends using strange combinations

like:

var$='CTRL,SHIFT,ALT,]' or var$ ='CTRL,SHIFT,ALT, 0 '

Enough about combinations ! Let's nicely assign value of var$ variable to PlugIn using PluginSet :

PluginSet("PlugIn","var$")

Now when PlugIn input variable is set, the only thing left to do is call PluginRun and execute command which PlugIn uses for setting of event handler key combination (virtual PlugIn uses SetHandler command).

PluginRun("PlugIn","SetHandler")

Lines are scattered around, let's put 'em together !

```
var$='CTRL,SHIFT,ALT, ]'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetHandler")
```

With these three lines, you set PlugIn handler key combination.

You're not done, you know ? Don't forget, something else must be set to this key combination !

- Fido ?

Nope.

- Coffee ?

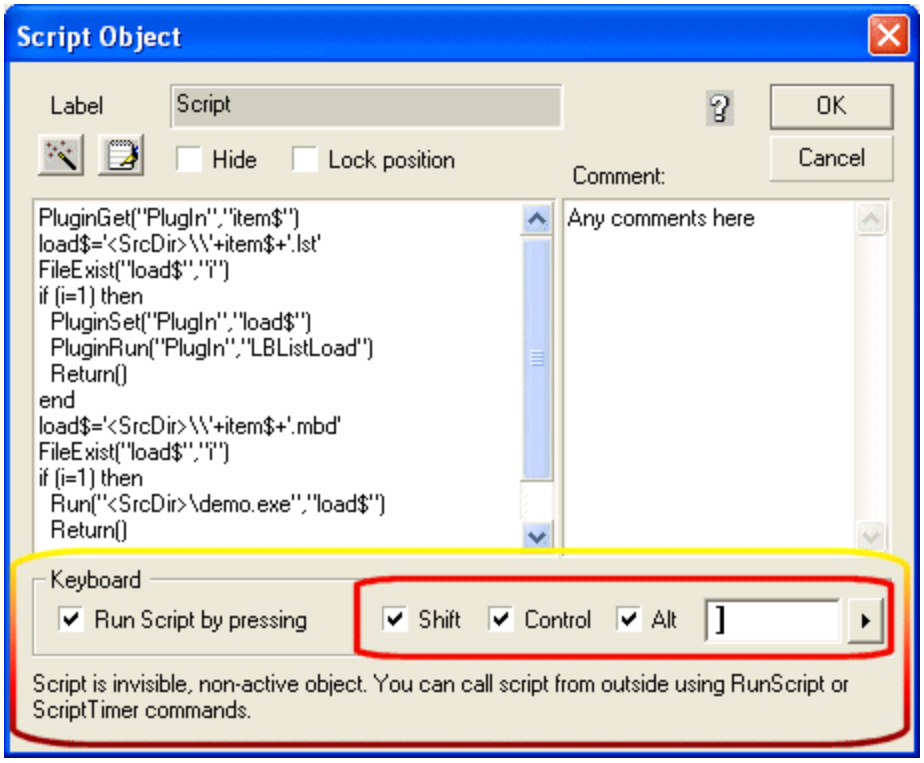
Nope.

- Neighbour ?

That would be interesting, but in this case - nope.

- Maybe MMB's Script object, a.k.a. Event Handler ?

YEP !



On image above it's visible that Script object in it's first line retrieves PlugIn string output:

PluginGet("PlugIn","item$")

Depending on PlugIn and events, triggered event will fill either output integer or output string variable. You should read PlugIn's documentation and check what variable is being returned.

Generally, PlugIns use output string variables to return important events like:

CONNECTED
POSITION CHANGED
FILE COPIED
Files Dropped

(because of MMB scripting language, being case-sensitive, make sure to write events using correct case letters when you use if-statements in script lines)

PlugIns use output integer variables mostly to return media file position, either by percentage or original format:

12
99
7321

Once again, in practice, under event handler (Script object),

you'll use:

PluginGet("PlugIn","event$")

...to retrieve string variable output of triggered event, and

PluginGet("PlugIn","event")

...to retrieve integer variable output of triggered event.

Once variables are filled with PlugIn event output, you can display events using some text object:

LoadText("Text","event$")

LoadText("SoundPosition","event")

...or use if-statements to decide what your application should do when some event occurs. This example uses virtual event generated by virtual PlugIn that informs your application about connection to Internet:

```
PluginGet("PlugIn","event$")
If (event$ ='Connected to Internet') then
Message("PlugIn detected Internet connection","")
Return()
End
```

Once again ! This script is put into Script object, a.k.a. event handler, set to receive key combination from PlugIn.

Important notice on Event Handling subject:

Receiving of events by event handlers in MMB ain't always available - while PlugIns send key strokes, if your MMB application is not focused - these key strokes might be sent to

-> some other application ! <-

Also, using of dialogboxes in application could become difficult if events are sent repeatedly, because PlugIns try to focus window of main application.
To avoid dialogbox defocusing, temporarily turn off sending of events in PlugIn (if available as PlugIn feature) while dialogbox is being used, and turn back on when you're done using 'em.

Frequent, repetitionous event sending might also cause interference with end-user's pressing of system keys (if being used).

Future releases of MMB shall feature advanced approach to this subject, so hopefully

events will be easier for both PlugIn developers and PlugIn users.

Color properties

MMB Visual PlugIns have another very neat feature - colors of their objects can be changed dynamically, on run-time. This is very important, giving application end-users ability to adjust visual properties according to their preferences.

There are usually two color properties you can set :
- background (back) color - sets background color of PlugIn's window or object
- foreground (font) color - sets foreground (or font) color of PlugIn's window or object

Changing of color properties is using PlugIn command with advanced string input. Once again, like in event handling case, we'll use advanced string input type with PlugIn command, this time to set color properties.

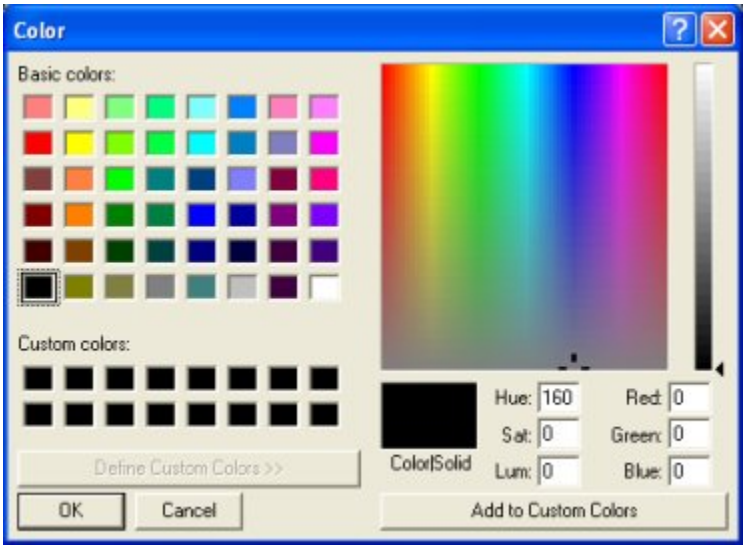
To specify color as input parameter, MMB PlugIns use RGB color system, just like monitors, TVs - and software, of course.

RGB stands for:

Red
Green
Blue

...or...

These three components represent basic colors. When you combine them in different ratios, all other colors can be displayed.



Setting of ratios, naturally, requires range of available values.

MMB PlugIns use the same range already known from standard windows color boxes:

0 - 255

Lower value of color component means lower participation of that color in RGB result:
- 0 - lowest value, removes component participation
- 255 - highest value, component will participate 100%

Advanced string for setting of PlugIn color property uses comma , as separator between three basic color components:

red, green,blue

And with values looks like this:

255, 150,50

...in practice:

var$=' 255,150, 50'

Combination of this color component values will result in this color:

That means - either PlugIn's foreground or background will become orange after running color property command.

As usual, you'll refer to PlugIn's documentation for exact command names. For purposes of this manual, let's use virtual PlugIn's background color command SetBackColor .

First of all, var$ is used as input string variable for setting of advanced string - color values of basic three components:

var$ ='170,40,170'

With PluginSet, color values in var$ are assigned to PlugIn:

PluginSet("PlugIn","var$")

Once values are set, call PluginRun to execute virtual PlugIn's command SetBackColor for setting of background color:

PluginRun("PlugIn","SetBackColor")

The result is pinky-like PlugIn background color:

Similar script lines would be used for setting of font color. Let's use virtual PlugIn's font color command SetFontColor:

```
var$ ='102,102,153'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetFontColor")
```

The result is blue-gray PlugIn font color:

Ability to change colors on run-time enables simulating color changing animation, through loops:

```
For i=1 To 255
var$ =CHAR(i)+',102,153'
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetFontColor")
Pause("20")
Next i
```

For skinnable applications, you'll probably use PlugIn color features to adjust appearance, according to loaded skin.

Color values should be saved, for example using MMB's Registry script commands:

```
SaveVariable ("BackColor","var$")
...and loaded / applied on application startup:
LoadVariable ("BackColor","var$")
PluginSet("PlugIn","var$")
PluginRun("PlugIn","SetFontColor")
```

MMB PlugIns: Summary

By reading manual, you found out this about MMB PlugIns:

- they're not like any other software plug-ins

- ain't looking like space shuttle or AC/DC plug, but 2D window in MMB application - sometimes visible, sometimes not
- almost all are Type 1 PlugIns so you'll need more copies of DLL files to run more than one instance
- all settings & processing is coordinated through MMB's script language, using string and integer variables
- PlugIn Constitution recognizes three laws (commands): PluginSet, PluginRun & PluginGet
- there are numerous variations on variable input & output subject - some PlugIn commands take inputs, some give outputs, strange ones neither give or take...
- always refer to original PlugIn's documentation for command labels, input & output parameters
- strange thing - event handling - informs your application when something (delayed) happens in PlugIn
- MMB PlugIns are downloadable from MMB-related resource sites or PlugIn author's sites
- Mad dog Fido still runs around the planet, collecting bones

The end is near, prepare !

...to get on to some serious business, of course ! ;-)

Like any Operator's Manual - this one would be nothing else but just a bunch of pages with pictures & schemes if it's contents ain't transfered into practice !

Move to (more-less, hah) real MMB world now - get those nasty PlugIns you spent download hours on - and open wide these new windows, get fresh air into your project !

Learn. Earn. Enjoy.

Here are a few more words from author of this Manual...

I find this to be a collection of answers to many times asked plugin-related questions. MMB PlugIn developers will greately appreciate if users spend enough time to get familiar with PlugIns through this manual, thus leaving more available time to PlugIn developers.

And who are these MMB PlugIn developers ?

- a bunch of ordinary guys doing PlugIns for a hobby, more on voluntary basis than with commercial attentions

Are they a part of Mediachance team ?

- no, PlugIn developers are not connected to Mediachance dev team

What about PlugIn support ?

- if PlugIn developer doesn't provide his own support system, you can post questions, suggestions & bug reports on MMB Forums. Please be aware of this: PlugIn developers are one-man-bands, having ordinary jobs & lives - so 24/7 support can hardly be expected, especially for older freeware PlugIns. Of course, commercial products mostly

have good support - and for questions on freeware PlugIn subjects you'll find good souls on MMB Forums, ready to help (except in bug report cases) :-)

Are .mbd demos coming with PlugIns ?

- yes, MMB PlugIns usually come with demos, showing how commands mentioned in documentation work in practice. Notice that PlugIn object paths in some demos will point to <SrcDir>\ - it's being used to lower PlugIn package redundancy (not providing DLL's more than once). In this case you'll adjust path to match a folder where PlugIn's DLL resides.
Another case is when DLL comes <Embedded>\ : if you cannot find DLL in the package, just go to Window's Temporary Folder after running PlugIn's demo - there you'll find DLL file and it can be copied wherever you want.

What about legal issues of using PlugIns in commercial products ?

- if specific legal or any other limitation is not mentioned in PlugIn's documentation / readme file, it can be used in any project, commercial or not. The real beauty behind MMB PlugIns is - purchasing of PlugIns is very small investment, compared to their features, power and the fact they'll even sometimes cover hardest parts of your applications, saving you time & giving a good night sleep, using royalty-free concept.

Back to Manual's opening story - your neighbour's now got something to think about ! While he's struggling with variables - you're on event handling ! As he gets to events - your project is already out there, impressing customers around the planet ! Just make sure he doesn't get a copy of manual himself, hah...

Don't forget - this doesn't have to be the last & only one version of MMB PlugIns: Operator's Manual. If MMB users show support towards this tutorial, it will be refreshed with many new examples, updated according to changes in MMB.

And Fido might get into a better mood too, catching those bones !

MMB PlugIns: Operator's Manual

Author of manual hopes you enjoyed reading all these pages, and with some luck learned something (or at least laughed while reading).

Being one of MMB PlugIn authors, I noticed more than once how lack of good reference can delay user's projects, result in lame solutions and eat great amount of time to both MMB users and PlugIn developers (who write explanations and demos over and over again). MMB PlugIns: Operator's Manual should cover both general and specific aspects of MMB PlugIn usage, helping MMB users understand that savage beast hidden in the dark.

While most of MMB users do not know how to get along with PlugIns, making of this Manual was pretty much a natural thing to do.

Author's contact info
e-mail: bokzy@ hi.hinet.hr
web: http://www.bokzy.com
ICQ: 12610160

This manual is free for personal use. If you find it helpful while building commercial products, please support development & maintenance of this manual and donate money either by purchasing author's plugins or by using payment arranged with author of this manual.

MMB PlugIns: Operator's Manual (V1, 01.06.2003.)
by Bojan Oljaca (a.k.a. Bokzy)

Every effort has been made to make this manual as complete and as accurate as

possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this manual.

14.2 MMB Plugins SDK

If you cannot find a specific 3rd party plugin and you are enough skilled in DLL programming, you may try to develop your own plugin for MMB.

At the moment, we support only VisualC++ as a programming environment for MMB plugins (due to the fact that MMB is developed in VC++). But you can use almost any programming language, which is able to produce standalone Win32 DLL files (no, Visual Basic really can't do this, but you may try the alternative basic-like languages like **PureBasic** or **Ibasic**). From other languages you can use Delphi, BorlandC++, Dev-Pascal and others. The only problem is that you will have to translate the MMB plugins header files to your chosen language ;)

Here you can find an official **VC++ Plugin SDK** If the link doesn't work for you, just go to the , where you can find folder called **MMBPlugInSDK.** In this folder is number of files describing the available SDK functions and also some Plugin examples with their sources! Simply examine these sources or just start making your own plugin with using these samples. You can freely modify or completely overwrite these source codes ;)

And finally, as far as we know, there are currently five other (non-VisualC) translations of the MMB SDK. These SDKs should be functional and updated by their authors, but they are not tested and supported (in terms of tech. support) by Mediachance! Here you can find more information about **Delphi**, **Ibasic**, **Borland C++ Builder**, **Dev-Pascal** or **Aurora** SDK translations.

Top Level Intro

This page is printed before a new top-level chapter starts

Part

15 Advanced Objects and Functions

15.1 Load Text

LoadText("TextObject","variable$")
LoadText("TextObject","path")
LoadText("TextObject","<List>")
LoadText("TextObject","<List>Number")
LoadText("variable$","path")
LoadText("variable$","<List>Number")

Important command for string variables.

The load text loads a string, a file from the path, a list or a single item from the list into the text object (Text, Paragraph text, Input Edit)It also loads a file from the path and a list (or item from the list) into the string variable.

Changed in 4.9.5:
If the string variable used as a first parameter will point to an object (text, button, ...) in your project, then the **LoadText** function will replace the text in this object i.e. it will not load the contents of the second parameter into the string variable as previously. This will allow you to dynamically replace the text in a serie of text objects or buttons by simple For..Next loop.

Let's imagine you have an array of 100 buttons and you have to change the title of all of them. Previously you need to use the LoadText function for each of them..it means 100 lines of almost the same code. But now you can use simple For..Next loop like here:

```
For i=1 To 100
  var$='TextBTN'+CHAR(i)
  LoadText("var$","var$")
Next i
```

The list is the **Song list**

This command can load a text into text object (Paragraph Text, Text, Text Button) from String variable or file. The command decides itself if the text in variable is just a plain text or it points to the file. It can expand the string to full path as well.

```
C$='<SrcDir>\MyFile.txt'
LoadText("Object","C$")
*** load a text file from the path directly to the Object.

C$='Whatever text'
LoadText("Object","C$")
*** display just the text directly in the Object.
```

In both cases (direct text or variable), the command will guess if the argument is a pointer to a file or just simple text.

If the file doesn't exist the LoadText will display it as a string.

Direct format
The previous format let MMB guess what we like to do (read text from file in string variable or display the string variable as a text). If you (or MMB) are in doubt, use the direct command before variable:
STRING:variable or FILE:variable

Example:
path$ = '<SrcDir>\myfile.txt'
LoadText("Text","FILE:path$") will load the text from a file specified in the path$
LoadText("Text","STRING:path$") will load the text from the path$ directly - in our case the Text will show the <SrcDir>\myfile.txt

Tip: If you want to split a long string into the multiple lines, then use \n parameter in the string.

Example:
LongStr$ = 'This is a very long string \nsplitted into two rows'
LoadText("Text","LongStr$")

See more about LoadText **here**.

15.2 Semi-Parallel Processes

Once we use **Refresh**() command in the loop we start Semi-Parallel process or what we call Parallel loop.
That means while the loop is still counting we can interact with any objects, run other short scripts and make other actions.

See first example: Normal loop

```
For n=0 To 10000
   a=n+1
Next n
```

Normal loop will pause the MMB (it will be inactive) for the time until loop will finish.
You can't click on any buttons during the looping.

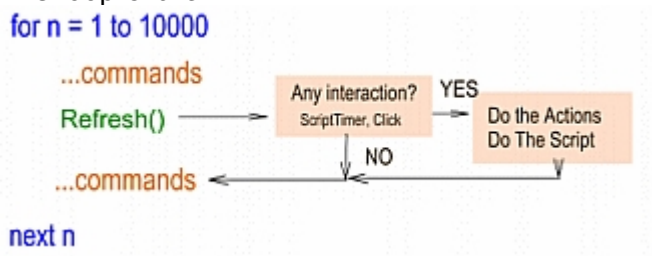
This is Parallel loop:

```
For n=0 To 10000
   a=n+1
   Refresh()
Next n
```

The command **Refresh**() will process the parallel interactions - that means while the loop is still looping we can click on buttons and run short scripts.

You have to remember
Any other (parallel or not) loop will pause the first one until the second will be finished.
See the chart below

The loop chart:



You should use Prallel loop only if you are certain what it does.

Example of using infinite loop.
With Parallel loop we can use also infinite loop option. (But you have to remember to exit)
The script will make the object Circle sticky to the mouse cursor - until some other object doesn't set the stop to 1.

```
stop = 0
For n=0 To Infinity
   MoveObject("Circle"," MOUSEX(),MOUSEY()")
   If (stop=1) Then
      Return()
```

```
    End
    Refresh()
Next n
```

15.3 Embedded Files

 From Menu Project - Embedded Files

You can embed any binary file in the project. It could be exe file, text file, pdf file etc.

Please make sure that you don't use this as a storage for your files. You should embed only if it is necessary.
Please use big files outside the project - don't embed them - they will slow down the startup of your application.

How to access embedded files.
Where normally you use <SrcDir> in case of embedded files you use <Embedded> prefix.

Example:
We embedded setup.exe into the project.

To call the file we use script
Run("<Embedded>\setup.exe","")

What to embed?
Exe files are usually not a good idea to embed, however you can use embedded files for:

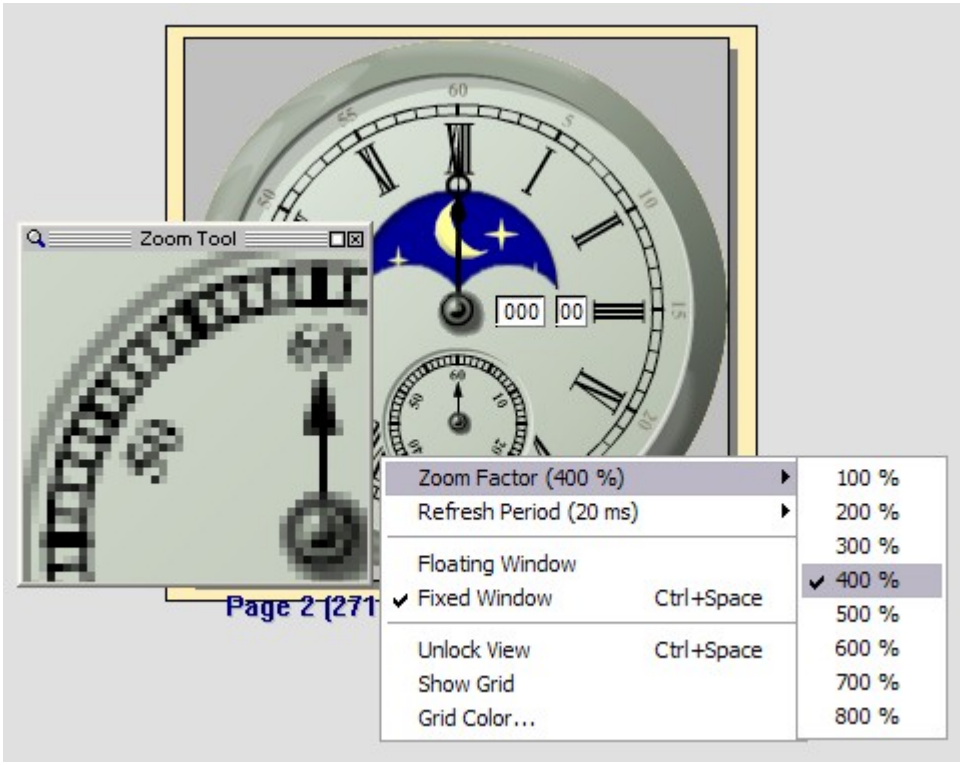
- jpg for **VR panorama**
- html files for **HTML Object**
- text files, etc.
- files for **Binding** (if not big)

Note: Custom Cursors are always embedded as Embedded Files.

Please don't embed AVI, MPG, OGG or some other big files if it is not necessary. These files are usually big and uncompressing them takes additional time.

15.4 Zoom Tool

 Zoom Tool is a tool for magnifying the selected part of MMB workspace under the mouse position. Running this tool will open the "Zoom Tool" window with enlarged part of screen. Right click in the window will appear the Zoom Tool settings menu.



Zoom Factor

Percentage of zoom. You can select between the 100 and 800 %.

Refresh Period

Amount of time used for refresh the Zoom Tool content. Lower value will refresh more often, but it can raise the CPU usage. The optimum value is between 20 - 50ms.

Floating Window

Cause move Zoom Tool window along with mouse.

Fixed Window

Deactivate Zoom Tool window movement (floating). Floating can be also skiped by CTRL+Space shortcut or by clicking on Zoom tool button.

Lock/Unlock View

This option will freeze/unfreeze the Zoom Tool updating when the mouse is

moved. For Lock/Unlock you can use also CTRL+Space shortcut.

Show/Hide Grid

Show assistant Grid inside the Zoom tool window.

Grid Color

Set the color of assitant Grid.

15.5 Clone

Clone object will inherit all graphic properties from its parent, but it can have different actions.
If you change the parent, all clone objects will change the same way.

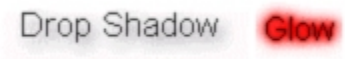
For example, you create the bitmap and then clone another five objects out of it.
If you later load other bitmap into the parent, all other five objects will change.

Clone tool can save a lot of space. Read more in **Clone Bitmap**

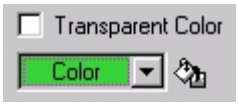
15.6 Glow/Drop Shadow

 This will create a new object based on the parent object (bitmap with one color, transparent, text or rectangle).
The Glow/DS object will be placed behind the parent object.

- **Label**
  String represents the new object.
- **Group with parent**
  New object will be grouped with parent after the new object is created.
- **Opacity**
  0 – 255. You can make Glow/DS more transparent by selecting a value less than 255
- **Color**
  Color of the result Glow/DS object can be set later – see the TIP
- **Orientation – Offset X, Y**
  Glow usually has the position 0,0 relative to the parent, and Drop Shadow is usually shifted a little bit to the right – bottom
- **Feather**
  How much diffuse will the new object have.
  1 – the Glow/DS is almost sharp, Bigger number – more blur
- **Direction**
  Direction of the diffusion



TIP: Glow/DS object is a bitmap object with alpha transparency map.



You can change the color of the Glow/DS object at any later time by opening the properties - changing the color in color box and pressing Fill button.

Don't check the Transparent Color check box !

Wizard – You can easily make an object that will glow if the user moves the mouse over the object. Select the parent object (for example the Text), click the Wizard icon on the Toolbar and select «Glow on mouse move». Now test the page and move the mouse over the object. The object will then glow.

15.7 Clone Bitmap

 If you select any Bitmap Object and go to the menu **Object,** you have enabled the item "Clone Bitmap Object".
This will make a virtual copy of the bitmap object. The new Clone object 'copy' all the visual properties of the parent Bitmap Object.

Now if you change the parent object (Bitmap) for example enabling the Alpha transparency, all the child objects will change as well.

Note: Clone Objects can exist across pages!
On 'Page 1' we copy the Clone Object to the clipboard (Ctrl+C) and here on Page 2 we paste it (Ctrl+V) - we have child of the parent bitmap from Page 1.
If you open the Clone properties, you will see its Parent is 'Page 1::Bitmap'

Knowing the fact the clone objects takes very little space we can make some tricks (animations) with the script and clone objects without creating huge files

Exploring the Clone Object Properties you notice the Clone Objects have some little independency from the parent. They can be independently Shown/Hiden and they each have its own action properties.

See **CloneObject.mbd** sample

15.8 Crop

Cropping cuts away rectangular areas on an image without affecting the resolution or dimension of the area that remains.

To crop an image:
Select image object.
Select Crop from menu Effects
Draw rectangle inside the image.

You can Restore the original image by clicking Restore Original in menu Effects.

If you resize the cropped image, the original image will be restored.
If you are satisfied with your new image then you can apply «Make new original» from menu Effects. After that you can resize the new, cropped image like it was the original image.

15.9 Make new Original

Bitmap images leave original image as a temporary. Whenever you resize the image, the original image is resized. This preserves the best possible quality.
The same works for cropping and Tile – the full image before cropping or tiling is still remembered. If you resize the cropped or tilled image – the original image will resize instead.
Make new Original replace the temporary original with the current image. Then you can resize tilled image or cropped image like you would do with normal image.

15.10 Tile

To tile the image:
Resize image, It should be bigger than the previous.
Click Tile in menu Effects.
To apply Tilling you should use Make new Original.
Otherwise, after resizing the original image will be restored.

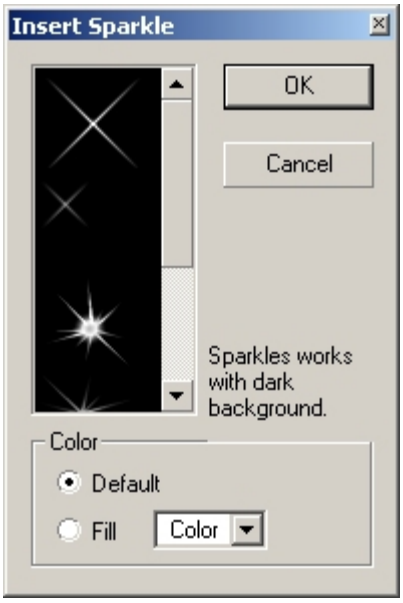
15.11 Reduce Size

Resizing bitmaps and applying effects will create temporary bitmaps inside the project. Those bitmaps are not necessary for distributing and removing them can significantly reduce the size of the project.

You should use this command if you are resizing the bitmaps or using effects before you distribute the files.

15.12 Sparkles

Sparkles works best on dark background.



Top Level Intro

This page is printed before a new top-level chapter starts

Part

16 Articles and Tutorials

16.1 List of MMB Samples

16.2 MM Builder Videos Guide

Written by Lifeson - **lifeson@mindspring.com**

NOTE: this document works perfectly with MMbuild v4.8 – there may be differences with the newer versions but hopefully the method remains basically the same

Introduction

Are you tired of reading endless posts and hunting through countless websites for bits and pieces of information on how to write complex MCI scripts? Do you simply want to use Multimedia Builder the way it was intended, and use the tools that are right there in front of you to build an exciting Video CD? Then this paper is for you !!

Here I list several steps to create a Video CD (not a VCD – that's an entirely different animal !!) using MM Builder – that will display shortcuts to any number of video clips, start the videos, stop them – and show/hide the controls perfectly.

The Goal

To create a CD that autostarts when the user inserts it, and shows a splash screen with a number of links listed to various video clips. When a link is clicked the video starts, plays with controls shown – then when done it vanishes and the controls vanish, taking you back to the original splash screen. The controls for each clip remain hidden until you activate the video, and they will then remain in sight, directly below the video, until it finishes or you click the Stop button – and then the video and controls vanish as they should.

My own Story

I have been playing with video for years now. I own a Sony TRV-900 Digital Camcorder, and a Canopus DV-Raptor capture card (previously I had a Miro DC-30 analog capture card). I captured 43 clips from my favorite movies . . . Planes Trains and Automobiles (the famous "those aren't pillows!!" scene), Wayne's World, Chris Farley, and even two full-length half-hour shows of my favorite vintage TV – The Honeymooners. I have since burned a few of these treasured CD's and handed them out to Family and Friends – they have been a big hit !! People can take these with them while traveling and simply pop it into their laptop on the plane. Depending on the quality of the videos you make (I use Xing to convert my videos from AVI to space-saving MPEG's) you could literally fit a full-length movie onto one 650 MB CD !!!

The tricky part is setting them up so that:

1. the videos and controls are hidden until you activate one of them
2. when the video starts, the controls show up underneath it
3. you only see one set of controls – there will be as many sets of controls as there are videos, so hiding the inactive sets properly is of course important
4. both the video and the controls will disappear when the video has either completed or been stopped manually.

My next project will be a CD resume, which will autostart a brief video Intro of myself on camera, and will contain buttons to click for my resume, projects I have done in

Powerpoint, and a provide link to my website (**http://www.infocellar.com/**) to the prospective employer.

Getting Started

Create a Mmbuilder project, and save it to a sub-folder within your main Mmbuilder application folder. Copy all videos there (buttons & bitmaps are contained within the autorun.exe file when the project is compiled and saved so no need top copy them in). I also copy my project file (.mdb) there as well, though it is not needed on the final CD.

Steps – these are included for you to use "after" you have gone through the detailed steps. It is hard to memorize the steps, and it takes a while to read through the detailed steps. Therefore, after you get the hang of it – print these out and use them as a reference when you make your project. You will go through all the steps once for each Video.

Insert the Video

1. add video, click OK to add still image. The videos will be named in the following order: Video, Video1, Video2, Video3, etc.

Insert and Configure the Controls

2. The video name in Objects Box will now be highlighted (if not, click it) and click Wizard and Add Video Controls – double click the name of the controls in the Object box to open their preferences – click "Hide" and then also rename them to match name of Video (example, for Video2, name your controls to "Controls2") – click OK to save the new name
3. Again - double click the name of the controls in the Object box to open their preferences and then double-click the Stop button entry, click the Video Icon (2nd from the left on bottom) and set Action 1 – leave as is - this should already be set up properly, but if it isn't, set it to Video Stop and select the correct video as the object, and then Action 2 – Hide and select the correct set of controls to as the object
   click OK OK

Configure the Video

4. double-click video name in Objects box, click "Hide Still Screen"
5. click "Run Script - Define" button and use Wizard to Hide the Object that is the controls for that video - click OK OK OK

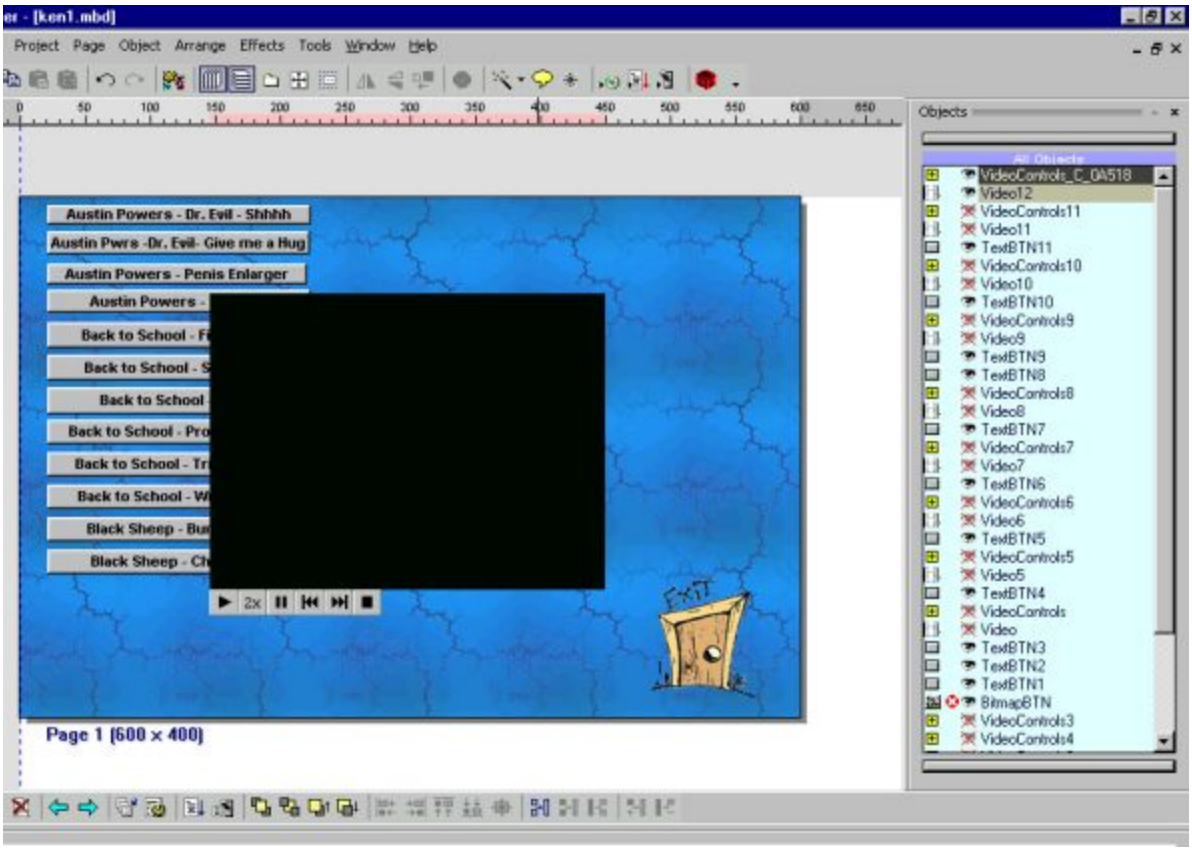
Insert and Configure Text Button

6. if you have an existing button, copy it (to make an identical button) click once to select it, then copy (CRTL-C) and paste it (CTRL-V) and then move it (the pasted button will be directly on top of the one you copied). Or else add text button from the left toolbar by clicking once on it and then clicking anywhere on your page. Double-click it, enter the name of the video in the "Text" field, click the Video Icon on the bottom (2nd from left) and set it so for: Action 1 – Video Play Action 2 – Show Video Controls
7. resize the button and place it where you want it

Details

Step 1 - Add Video - Using the tools on the left side, click the Filmstrip button and then click on your page where you want the video to run – the place you click will be the upper left-hand corner of the video, so keep in mind that it will extend down and to the right – also try to memorize a spot where you will click for all video so that they all run in the same spot. Then select the video from your hard drive. Instantly, a window opens with controls to play the video. At this point you unfortunately *must* add a "still screenshot" of the video, even though you don't need one, since the buttons and/or text will act as the activator (so later we will Hide this "still image").

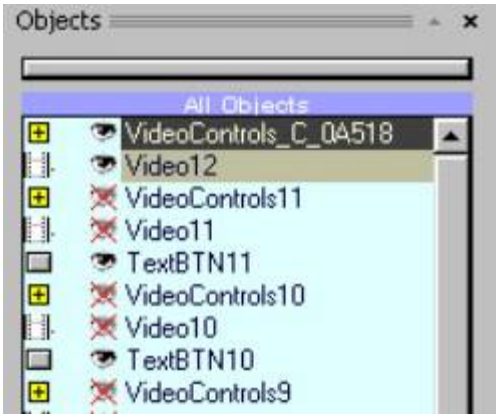
If, on the other hand - you actually DO want to add a still image, play the video and pause it on the image you want.



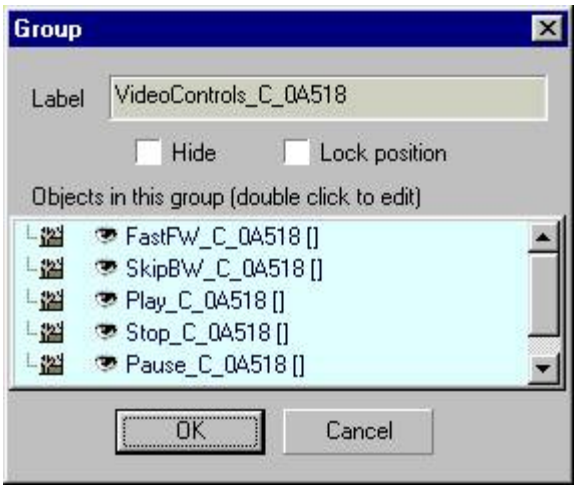
Step 2 - Add a Set of Controls for the Video - Add one set of controls for each video. The video name in Objects Box will now already be highlighted (if not, click it). Then click the Wizard Button (the Magic Wand on top), and select "Insert Video Controls". The video controls will show up below the "still" and will be automatically linked to the video you have selected. In this example, it is the 13th video I have added.

The videos are automatically given the following names in this exact order: Video, Video1, Video2, etc. The controls, however, are given weird names – so I rename them

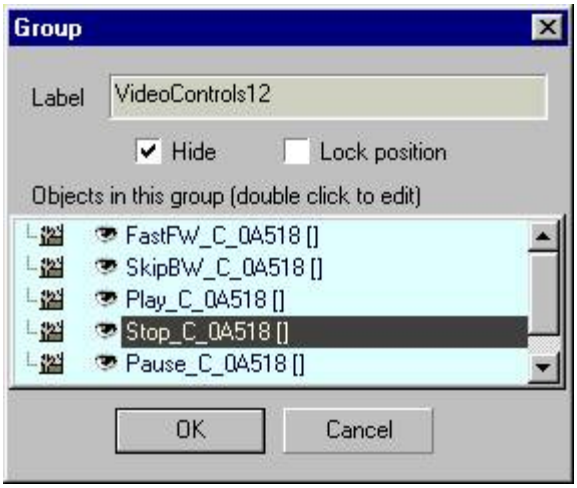
to match the video. In this example my video was automatically named "Video12" (it is the 13th video because the first video is simply named "Video"). The controls for Video12 were given the weird name "VideoControls_C_0A518". Since you may need to work with the project in the future you need to have the controls name be related to the video name, so we will rename it next. Each Video has it's own separate set of controls.



Double click the name of the controls in the Object and you will see this box:



- Click "Hide" so that they do not clutter up the main page
- rename the Label to match name of Video (for my example, for Video12, I renamed the controls to "Controls12") – click OK to save the new name – MAKE SURE TO CLICK OK HERE EVEN THOUGH IT CLOSES THE BOX !! **NOTE:** make sure that you renamed the set of controls in this exact step – not later. If you go in later and rename them to VideoControls12 the control buttons will lose their linkage to the video !! RENAME THEM IMMEDIATELY AFTER CREATING THEM.

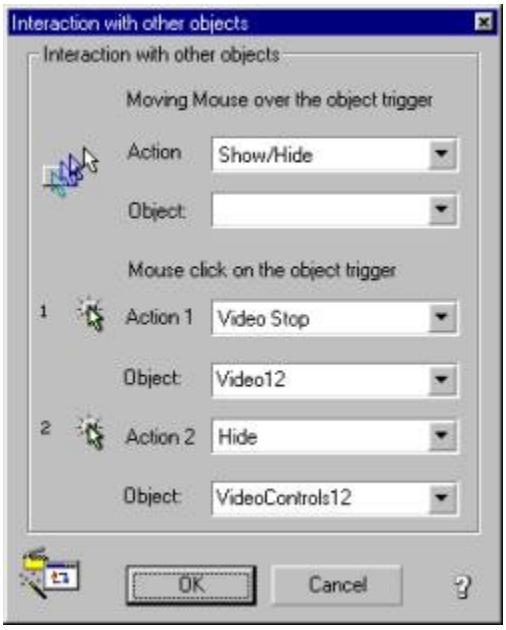


Now reopen the controls box - double click the name of the controls in the Object box to open it. We need to make the Stop button stop the video and also hide the controls. Double-click the "Stop" entry in the list of the various controls to bring up it's own dialog box. Click the Video icon at the bottom (2nd icon from the left on bottom - it will have a partial blue bar underneath, showing it has an action).

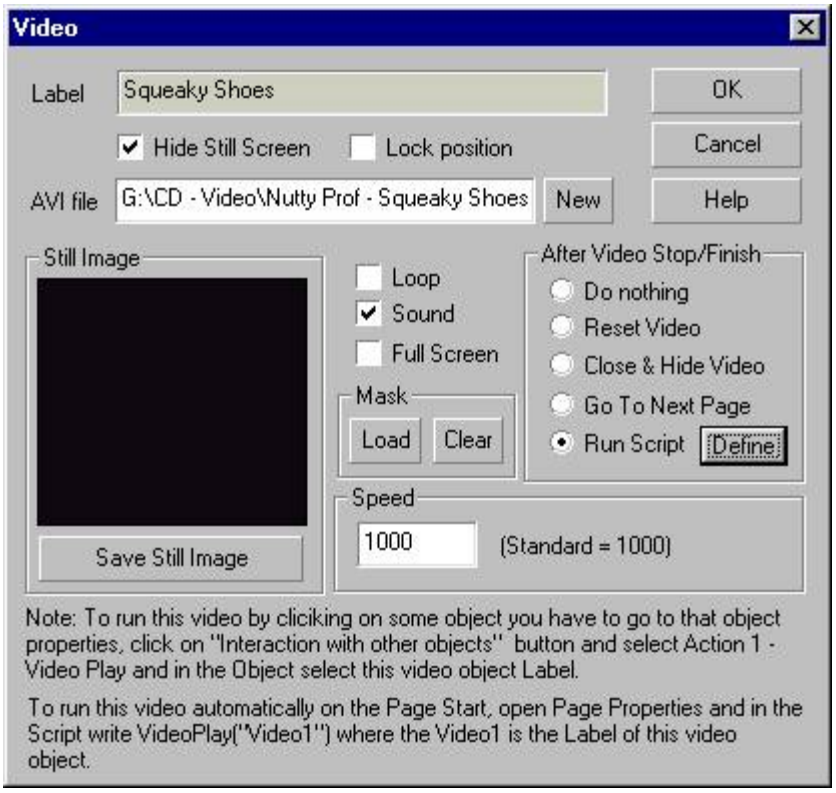
.1. Leave the first action 1 (Video Stop) as is (if you renamed the Label of your Video, you may need to select that Object again)

.2. edit action 2 by selecting "Hide" and then select the Object as the video controls (make sure to select the correct set of controls – there will be many as you add more and more videos and more and more controls).

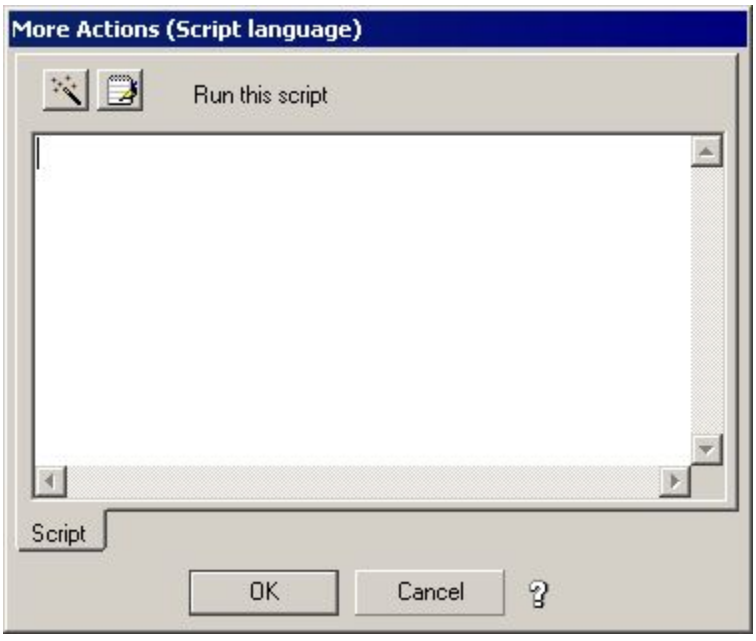
NOTE: when you click the "Stop" button, it does make the controls vanish even without doing Step 2. However, when you click Pause and then click Stop, the controls do not vanish, and Step 2 fixes that.



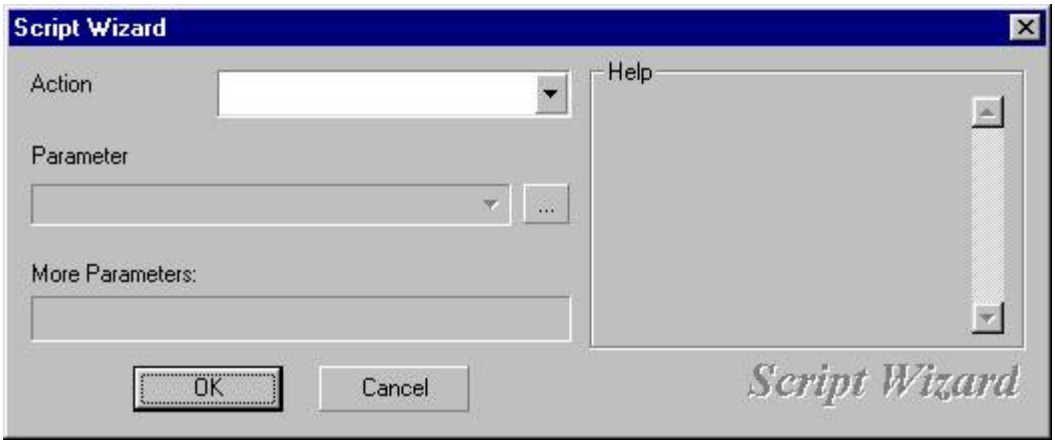
Step 3 - Set up Video parameters and make Controls vanish upon completion of Video - double-click the video in the Objects box to open up it's dialog box, and then rename it's Label to whatever you want, and check "Hide Still Screen".



The video itself automatically vanishes upon completion, but the controls will not. To make that happen, click run script/define, then click the Wizard button (wand) :

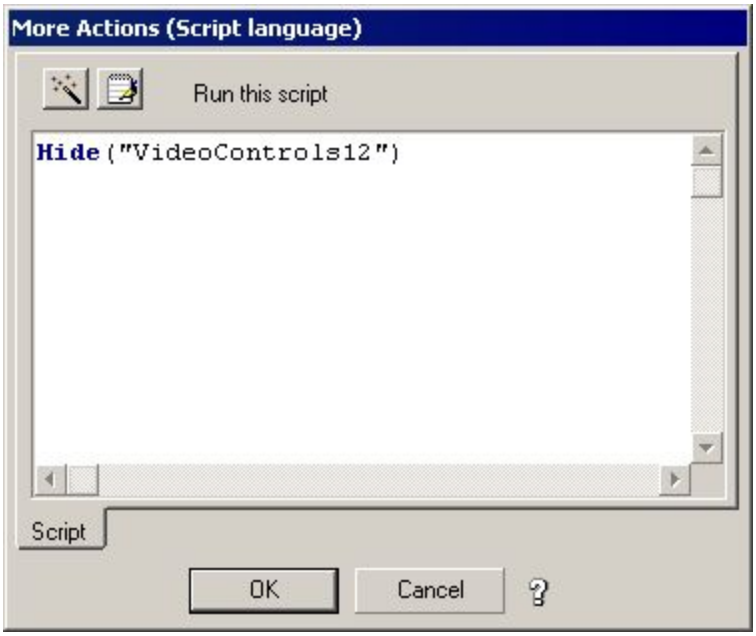


This brings you to the "Script Wizard" box :



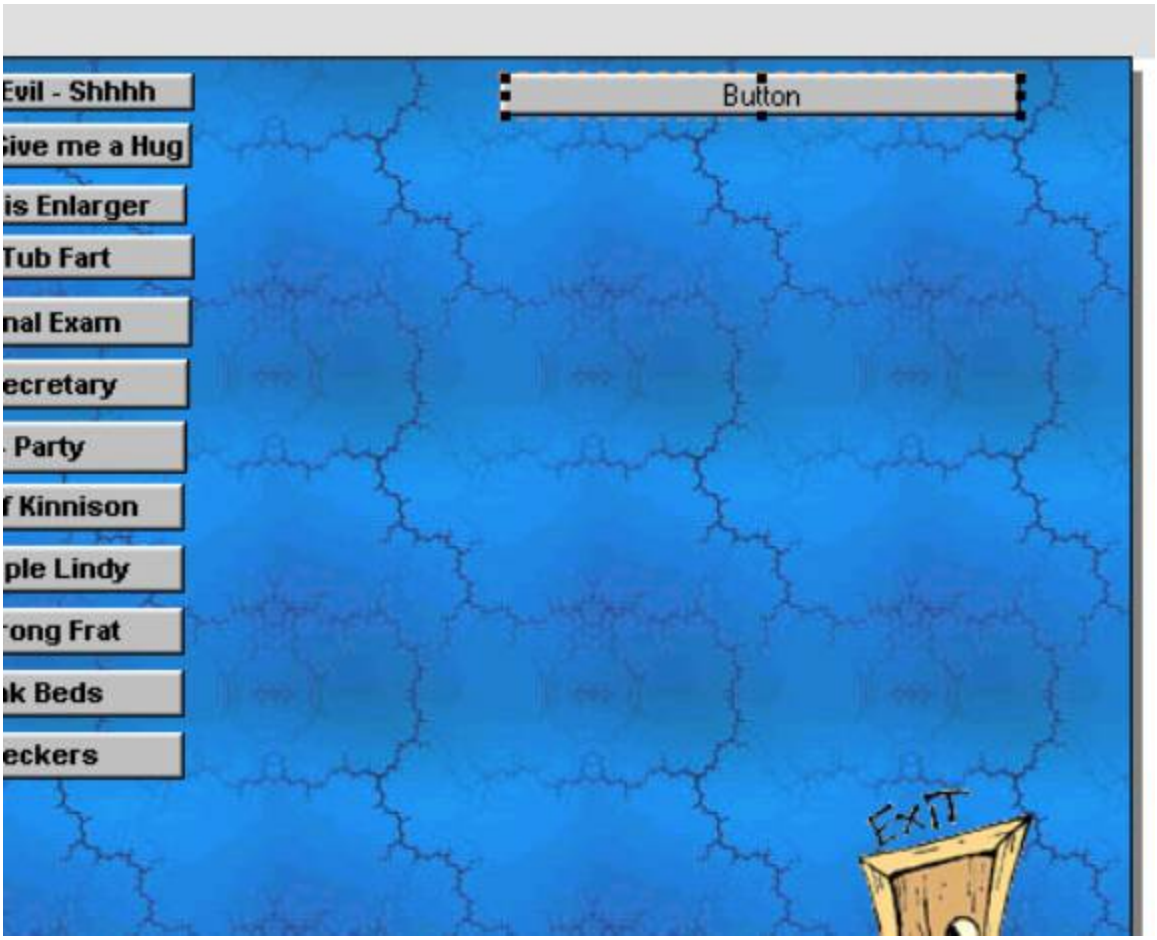
Select the Action "Hide" (use the drop-down arrow), and then also select Object with the *correct set* of controls (VideoControls12) :

When you click OK you will see the script listed:



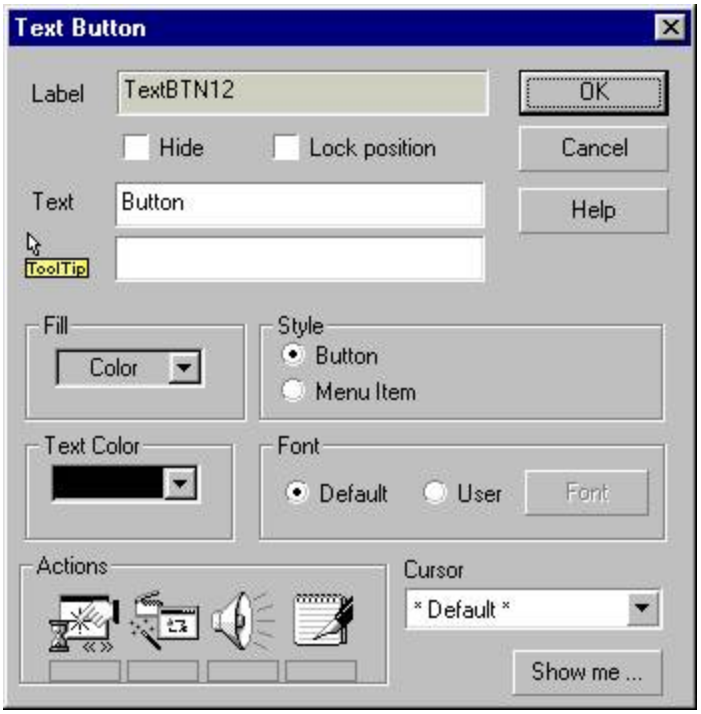
Click OK, and then OK again to get back to the main screen.

Step 4 - Add Bitmap or Text Button to activate Video and "show" controls - if you have an existing button, copy (CRTL-C) and paste it (CTRL-V) and then move it (the pasted button will be directly on top of the one you copied). Otherwise, using the tools on the left side, click either the "Text Button" or the "Bitmap button" and then click where you want the button on your splash screen – this button will serve to activate the video and Show the controls. I used the simpler "Text Button" for this example :

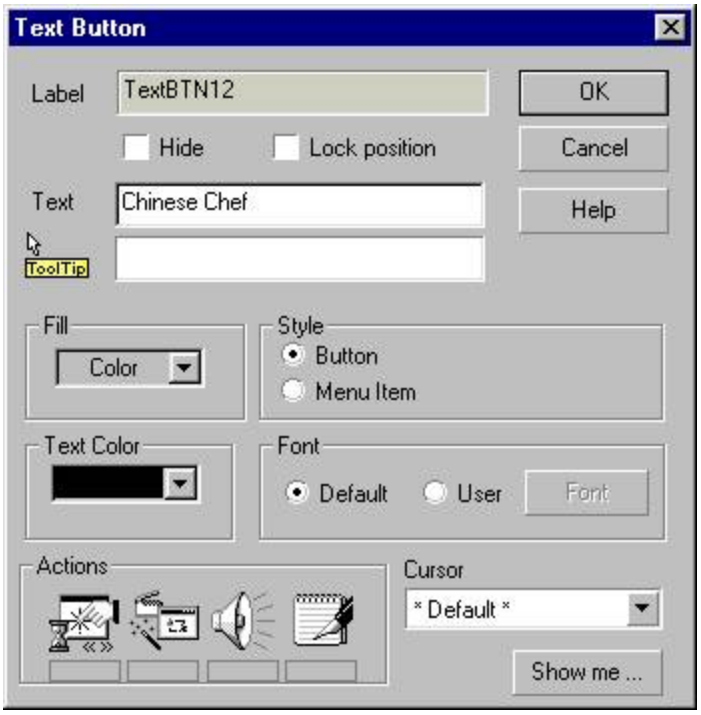


NOTE: Mmbuilder has some great built-in buttons – click the tiny "Library" open document icon to access them – they all have three buttons to act as Auto-Open, Highlight, and Click Image

Resize the button to make it the same size as the others, and then double-click it :



Rename it to the same name as your Video :

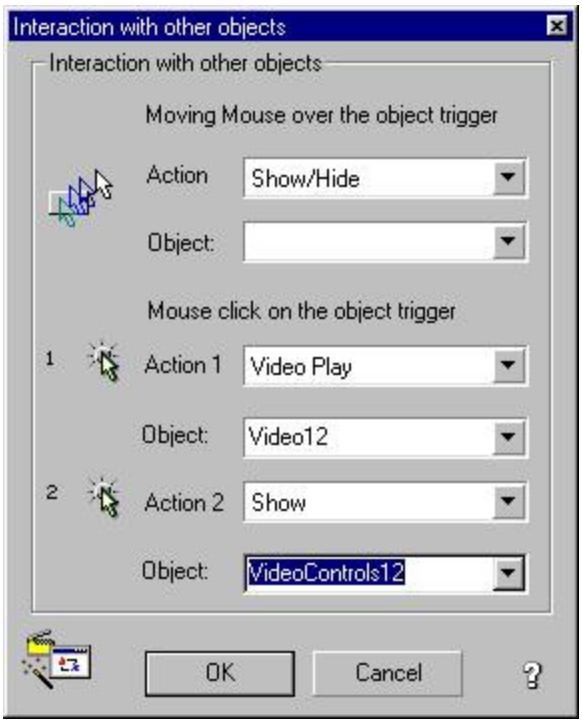


Now – the button you have just selected will serve to Play the Video and also to "Show" the controls, since unfortunately, when you hide the controls, they are also hidden when the video plays – this button will override that.

1. click the Video icon at the bottom (the second of four "Action" icons – the one with

the Wand) and select Action 1 as "Video Play", and select the Object as the name of your video

2. select Action 2 as "Show" and the object as the controls for that video:



Click OK and then test the current page (F5) and run your video !! Test the first couple of videos that you inserts by allowing them to complete, and also make sure th Stop button works properly.

Repeat these Steps once for each Video – when comfortable, just print the page with the steps and use that to refer to. It is difficult to memorize all the steps, and even I use the "Steps sheet" each time I make a video CD.

16.3 Real-DRAW Pro tutorial - Web/Multimedia Button

Application for Multimedia Builder

Multimedia authoring tools, such as Multimedia Builder, use the Multi-Faces images (Buttons) to show interactivity on the multimedia presentation. The **Normal** button is displayed during inactivity. If a user hovers over the button with the mouse, then the second **Over** image will be displayed telling the user that he is over an Active button. If the user presses the button then the third **Down** image will than be displayed. Such kinds of objects are very common for all authoring systems.

Real-DRAW can export the projects as a MBD or MEF project for Multimedia Builder or as a number of separate graphic images plus a Script. Script is a very simple but powerful language allowing you to create a file in **XML**, **HTML**, **Java Script** or any other text based language.

Some of the basic principles are same as described in previous chapter for WEB buttons.

To Export use menu Files: **MM & Script Export**

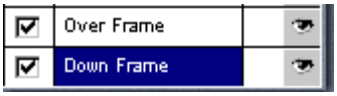
Example - Create interactive page with cool button

1. This is that we designed for a multimedia interface. Note the handle on the right, which we created as a button, so that when user clicks on the handle, it will move down.

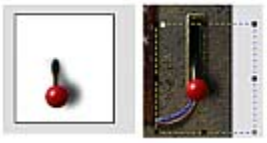
2. Now lets carefully select all the objects we need in the handle and don't forget the shadow. When it's selected, click the **Create package** button. We can now double click on the new package to open it, to see if all objects we want are inside, and they're if not, you can easily pick one from the document, and simply copy it to the opened package view.

3. Let's close the Package for now. Next, when in the main document, select the Package, and in the Web/Multimedia Button window Check, the Over and Down Frame

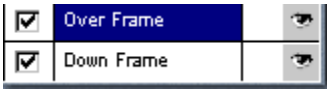
options.



4. We already have Down Frame selected, so let's work on it. Double-click on it to open the Package on this frame. Then re-arrange the objects so the handle is in the down position. Rotate the shadow to get to the right position as well and close the package.



5. We still have Over frame, which you may or may not edit. Let's say we just want the red handle to glow little if the user moves their mouse over it. Select the **Over** frame and the handle should change to its previous position, displaying an Over frame, which is a copy of the Normal frame by default.



6. Open the Package on this frame by double clicking. Change the color of the red sphere to orange or do whatever else you think may be good.

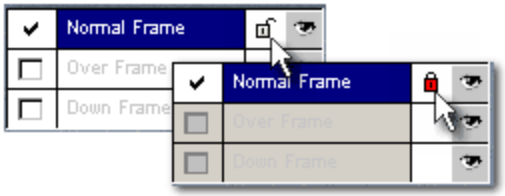
For example, move it to the middle.

7. We're done. Check all of the Normal, Over, and Down buttons to see if the transition is OK and go to menu: File - Web/Multimedia Export.
Leave the Multimedia Builder as the source and export it. Now open the Multimedia Builder and load the file to test it. You should have a great working handle.

- **Note** The Handle is created in a Bitmap Button in MMB, and that the button is set to, Locked in designer, so that you can't accidentally move it with the mouse.

Create Bitmap Object - Solo mode

Sometimes you don't really want to have a button but you want to have a separate object in your multimedia anyway. Maybe you just want to create a special image that will pop-up in your Multimedia applications because of some script or other action. Like a warning or window. But you don't want to have these images active and you don't need any other states. The Solo Mode Lock is just for that. It disables the Over and Down frame, and you are working with the object, just as with any other normal object. However, the Solo Mode Lock is recognized during the Multimedia Export and creates a separate object.



To activate **Solo Mode Lock** you must not have Over nor Down Frame checked. Then a small padlock appear next to Normal Frame. Click on it and the Solo Mode is Locked. (You will not be able to add Over or Down Frame on Locked Objects)

For Multimedia Builder (MMB) users

This creates a separate Bitmap object with an alpha transparency that could be freely hidden or moved within your MMB application.

Let's continue with our example.

8. We decided that as a user moves the handle, the strange device should show some life. We wanted the middle sphere to glow like a bulb.

9. Select the Sphere and Copy it to clipboard (CTRL+C) then Paste it back (CTRL+V). This will create a new copy on the top of the first sphere. We don't want to create a bitmap button this time but only to create a simple Bitmap object in MMB and still keep the original as a background, which is why we need a copy.

The original will stay dull, and because it will have no Multimedia Extension properties set, it will be exported to the background. The new copy should be exported as a single object, so select the sphere and click **Create Bitmap Object** on the Multimedia Extension window. Now add a new Point light to the sphere, which will create an interesting glow. We also need to set the Inset intensity lower, to give a more raised look.

10. Now export to Multimedia Builder again. In Multimedia Builder load, the new project and you'll see that we have an additional object - Bitmap.

Hide it, because we want to show it if the user clicks on the handle. Open the handle properties by double clicking on the Button 0 in the Object list. Click the second button in Actions - Interaction with other objects- and then in the "Mouse click on the object trigger", select Show as Action1 and select Bitmap1 as Object.

That's it, now test it.

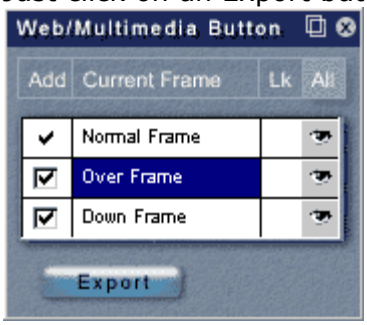
If a user clicks on the handle, the large sphere will glow and looks like a light bulb.

The Multimedia function is not that hard to master, but you must be careful and always know which frame you are editing.

It can definitely be a great helper, because you can design an entire graphics interface for a multimedia program, and in the case of our Multimedia Builder, you will get a working project directly.

Export the 3 Stage Alpha Button (*.3sb)

Just click on an Export button in "Web/Multimedia Button" dialog.



The export will create a **3 Stage Alpha Button** (*.3sb), which can be loaded in Multimedia Builder. After it, you can use it as a usual MMB object (Button).

Export an MMB file

This is very easy. The export will create a single Multimedia Builder file (*mbd) and you can simply load it into our Multimedia Builder program.

The 3-state object will be exported as a Bitmap button. Because its surroundings depend on the position against the background, the position of the button will be

locked in MMB and this is so you can't select it and drag it with a mouse by accident. The rest of the non-interactive objects will be exported as a background. If the Solo Mode was used on any object (Create Bitmap Object), it will create a movable Bitmap object with an alpha transparency.

Meta Export in MEF file

When you've worked with export to MMB long enough, you will realize one weakness. When you load your design to MMB, and then make changes, add script or do additional inter-activity, you may suddenly need to change a few things on the original design. While you can make the changes in Real-DRAW, you won't be able to update the project in MMB without loosing what you did there, or without a large amount of copy and paste, there and back, to save your scripts.

Here comes MEF to save you.

The MEF has simple requirements.

- That all interactive objects you made in Real-DRAW have a unique Label - it will even make them unique for you.
- That you load these objects into MMB with the same Labels.
- That you do not change any of these labels in Real-DRAW, nor MMB. Unless you change them to be both the same.

Basically if you keep the Labels of the same interactive object in Real-Draw and MMB the same, you are fine. The MEF update will also create a new object if you add a new object in Real-Draw and update the ones that remain.
MEF utilize also the new Alpha Button in Multimedia Builder - which creates truly transparent button without boundary problems.

What are the usual steps?

- Create a design in Real-DRAW, the same was as described above.
- Export it into the WEB/MM Export as a Meta Export MEF file.
- Create a new project in MMB and make it the same size as the design in Real-DRAW or continue in an already existing project. Note that the MEF Import in MMB is much like an object Import in a page, so it doesn't change the project size, nor the objects that you already have there.
- Load MEF using File-MEF Import-Load MEF.
- Work on the MMB project as you want
- Create changes in the design in Real-DRAW, and export it again to MEF.
- In the MMB use File-MEF Import-Update from MEF.
- The old objects will be updated, and if you created a new one, it will be added.
- Repeat as many times as you need to.

Note 1: Even if you first exported your project in Real-DRAW as an mbd file, you can later use the MEF update option. Simply create MEF from the Real-DRAW file remembering that the first time it will change Labels in Real-DRAW. Then make the labels in MMB the same as labels of the interactive objects you now see in the Real-DRAW project. You are ready to Update from MEF.

Note 2: The first time you use the MEF Export it will cause all the Labels of interactive objects to be unique - by adding numbers. It will add a number to every interactive object; even if you just have one.

Export to other authoring systems
Simply use: Web/Multimedia Script using JPG files in the MM & Script Export and follow next pages.

16.4 Optimize the Speed

Crop the Images to the size you really need.
Put as much bitmap graphic as you can into the background – the background image redraws much faster than other objects. The tool Combine objects with background in Arrange menu can help you.
Big images with alpha transparency can slow down the application
For 256 colors – you have the palette.bmp file inside the Palette directory.
You can use this image to extract palette inside your graphic editor. All your graphics should use colors from this palette. Convert your pictures with your graphics editor to 8 bit with using this palette.
For Corel Photo Paint there is already palette prepared inside the MMB's Palette directory. Reduce the size of the project.

Click Reduce size from File menu. This will remove all unused temporary bitmaps from the project.
Embed only necessary minimum of files and keep the file size of single MMB project file under or equal 2-3MB. Bigger files can significantly slow down the application start (because of extracting embedded files to temp folder and memory in case of large images). Leave as much files as you can external and load them dynamically via MMB script.

16.5 Embedded Fonts

1. In the first page of your project (oh sure, you could put it on the last page just to be contrary, but do you honestly think it's going to work?) click on "Project"
2. Click on "Embedded Files"
3. Click on "Add"
4. Navigate to the folder containing your font, select it and click open
5. On the "Embedded Files" window, click "OK"
6. At the bottom of the page, click on "Page Properties"
7. In the new window, click on "Script"
8. In the cute little box, type the following
   InstallFont("<Embedded>\thefont.ttf")
9. Save your project
10. Thank all the folks at MMB Forum that are always so kind to share their knowledge with the rest of us.
11. *Make sure that the font is freeware or you have the right to distribute it. (By E. Barry)

16.6 Command Line Support

With Command Line support you can send an unlimited number of parameters to your compiled applications (even to already running apps with a single instance option enabled).
Look at included **cmdline.mbd** example. Simply compile that project and run it with some command line parameters.

Here are some other **example files**:

Rules for passing command line parameters to MMB exe:

- Each parameter passed to command line must be separated by space.
  Example:
  Program.exe *param1 param2* => two parameters passed to exe

- If you want to pass one or more parameters with an empty space inside, then this parameter must be enclosed in double quotes. That apply especially to path to files with some empty spaces!
  Example:
  Program.exe *param1 "param2 param2"* => two parameters passed to exe but the second parameter is split by empty space.

- If you want to pass also a double quotes within the parameter, just double the quotes.
  Example:
  Program.exe *param1 ""param2 param2""* => two parameters passed to exe and the second param is passed with double quotes

All parameters passed to MMB application are stored (as a **string!**) in a predefined string array called ***CmdLineParam$[n]*** where **n** is a number between 0 and the number of passed parameters.

CmdLineParam$[0] contains number of passed parameters and the rest of ***CmdLineParam$[1..n]*** holds the cmd line parameters. With this you can easily enumerate the parameters passed to exe and then process them.

If you pass some numbers to your app and you want to use them as number (i.e. not as string returned by ***CmdLineParam$[n]***) you will have to convert the string to number (by **VAL** predefined function).

EXAMPLES:
If you run MMB exe with 3 parameters the ***CmdLineParam$[n]*** will appear like this:
myprogram.exe param1 param2 param3

CmdLineParam$[0]=>3 **number of passed parameters
CmdLineParam$[1]=>param1 **first parameter
CmdLineParam$[2]=>param2 **second parameter
CmdLineParam$[3]=>param3 **third parameter

Or if you run your exe with two parameters, but one of them is separated by space...
myprogram.exe param1 "param21 param22"

CmdLineParam$[0]=>2 **number of passed parameters
CmdLineParam$[1]=>param1 **first parameter
CmdLineParam$[2]=>param21 param22**second parameter

Here is a complete (but almost useless) example of parsing command line parameters passed to any MMB application.

EXAMPLE:
Insert this code to MouseUp event of any active object (e.g. button), compile app and start it with some parameters. This will show you the parameters in Message box.

```
** CmdLineParam$[0] holds number of passed cmd line parameters
n=VAL(CmdLineParam$[0])
** if number of parameters is > 0 then...
If (n>0) Then
  ** This loop simply enumerate the passed parameters
  For i=1 To n
    ** ..and show the obtained parameters in message box (or do anything
you want)
    Message("Command line parameter:","CmdLineParam$[i]")
  Next i
End
```

However, the above (rather manual) command line processing would be useless without an automated way of processing the passed parameters. If you want to automatically process the command line parameters on time when they arrive to MMB application, just create a new Script object with label **CBK_CMDLINE** (preferably on Master Page/ Layer) and insert your cmd line processing code into this Script object. Each time you will start MMB application with that CBK and with some command line parameters, this CBK_CMDLINE will be invoked and then you can process the passed command line parameters.

With this special script object you can not only process the command line parameters passed to newly started application, but you can also process some new parameters to an already running application with one instance enabled. For example, if the application is already running and you want to pass some new cmdline parameters to this application, simply call the application exe once again with your new cmd line parameters.

NOTE! The dark side of this new command line support (but from our point of view a big plus ;) is the fact that mmb apps no longer automatically process the audio/video command line parameters like in previous MMB versions. In other words, if you want to pass the OGG file to your app and then play it, you will have to make it play via new CBK_CMDLINE script. Previously, MMB automatically started known file formats, like ogg or avi.

The only file format, which is still loaded if you pass it to mmb application is MBD project file. If you pass a MBD file as the first parameter of MMB application, it will be automatically loaded in the application's main window. If you want to avoid this automatic starting of MBD files, and process the project files by yourself, then pass the path to MBD file in the second application parameter (instead of first parameter):

in short, instead of running your application with these parameters...
myprogram.exe "PathToProject.mbd" "AnyParameter"

run it with these...
myprogram.exe "AnyParameter" "PathToProject.mbd"

HINT! With command line support you can now create windows Screen Savers including the settings dialog. For an example **screensaver.mbd** or **rlt4.mbd** example. Just compile these projects with scr file extension (instead of exe) and add them to Windows folder. Then install them via Display Properties dialog (Screen Saver tab).

16.7 FMOD & Video Error numbers

FMOD Errors

Error no.	Error description
-19	An error occured trying to open the specified CD device.
-18	Windows Media Player not installed so cannot play wma or use internet streaming.
-17	Recording is not supported on this machine.
-16	Failed to allocate a new channel.
-15	Tried to use an EAX command on a non EAX enabled channel or output.
-14	An invalid parameter was passed to this function.
-13	The version number of this file format is not supported.
-12	Not enough memory or resources.
-11	Error loading file.
-10	Unknown file format.
-9	File not found.
-8	Error creating hardware sound buffer.
-7	Error setting cooperative level for hardware.
-6	Soundcard does not support the features needed for this soundsystem (16bit stereo output).
-5	Playing the sound failed.
-4	Error initializing output device, but more specifically, the output device is already in use and cannot be reused.
-3	Error initializing output device.
-2	This command failed because FSOUND_Init or FSOUND_SetOutput was not called.
-1	Cannot call this command after FSOUND_Init. Call FSOUND_Close first.
0	No errors.
1	FMOD is using DirectX.
2	FMOD is using WindowsMedia WaveOut.
3	NoSound.

Video (DirectShow) Error numbers:

Run **DXErrors.exe** application from MMB root directory and then enter the error number you got into the "Error Value:" edit field. Then press "Look Up" button and you will get the description of the error number you received via **CBK_Error**.

16.8 ASCII Table

The standard ASCII character set consists of 128 decimal numbers ranging from zero through 127 assigned to letters, numbers, punctuation marks, and the most common special characters. The Extended ASCII Character Set also consists of 128 decimal numbers and ranges from 128 through 255 representing additional special, mathematical, graphic, and foreign characters.

The charts in this section show the default character set for a console application.

To use some of the extended (127-255) or non-printable (0-32) ASCII characters codes inside MMB use **CHR**(dec) function.

The following table lists 0 - 127.

dec	hex	char	description	dec	hex	char	dec	hex	char	dec	hex	char
0	0	NUL	Null	32	20	space	64	40	@	96	60	`
1	1	SOH	Start of Heading	33	21	!	65	41	A	97	61	a
2	2	STX	Start of Text	34	22	"	66	42	B	98	62	b
3	3	ETX	End of Text	35	23	#	67	43	C	99	63	c
4	4	EOT	End of Transmis.	36	24	$	68	44	D	100	64	d
5	5	ENQ	Enquiry	37	25	%	69	45	E	101	65	e
6	6	ACK	Acknowledge	38	26	&	70	46	F	102	66	f
7	7	BEL	Audible Bell	39	27	'	71	47	G	103	67	g
8	8	BS**	Backspace	40	28	(	72	48	H	104	68	h
9	9	TAB**	Horizontal Tab	41	29	)	73	49	I	105	69	i
10	A	LF**	Line Feed	42	2A	*	74	4A	J	106	6A	j
11	B	VT	Vertical Tab	43	2B	+	75	4B	K	107	6B	k
12	C	FF	Form Feed	44	2C	,	76	4C	L	108	6C	l
13	D	CR**	Carriage Return	45	2D	-	77	4D	M	109	6D	m
14	E	SO	Shift Out	46	2E	.	78	4E	N	110	6E	n
15	F	SI	Shift In	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	Data Link Escape	48	30	0	80	50	P	112	70	p
17	11	DC1	Device Control 1	49	31	1	81	51	Q	113	71	q
18	12	DC2	Device Control 2	50	32	2	82	52	R	114	72	r
19	13	DC3	Device Control 3	51	33	3	83	53	S	115	73	s
20	14	DC4	Device Control 4	52	34	4	84	54	T	116	74	t
21	15	NAK	Negative Acknowl.	53	35	5	85	55	U	117	75	u
22	16	SYN	Synchronous Idle	54	36	6	86	56	V	118	76	v
23	17	ETB	End of Trans. Block	55	37	7	87	57	W	119	77	w
24	18	CAN	Cancel	56	38	8	88	58	X	120	78	x
25	19	EM	End of Medium	57	39	9	89	59	Y	121	79	y
26	1A	SUB	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	File Separator	60	3C	<	92	5C	\	124	7C	|

29	1D	GS	Group Separator	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	Record Separator	62	3E	>	94	5E	^	126	7E	~
31	1F	US	Unit Separator	63	3F	?	95	5F	_	127	7F	□

** Values 0-32 and 127 are non-printable characters. They have no graphical representation, but depending on the application, may affect the visual display of text.

The characters that appear in Windows above 127 depend on the selected typeface (i.e. the character code **128=€** or **169=©** in Arial font.

The following table lists 128 - 255.

dec	hex	char cons.	char win.	dec	hex	char cons.	char win.	dec	hex	char cons.	char win.	dec	hex	char cons.	char win.
128	80	Ç ü	€	160	A0	á í	space	192	C0	└ ┴	R	224	E0	α ß	r
129	81	é â		161	A1	ó ú	?	193	C1	┬ ├	Á	225	E1	Γ π	á
130	82	ä à	‚	162	A2	ñ Ñ	?	194	C2	─ ┼	Â	226	E2	Σ σ	â
131	83	å Ç	□	163	A3	ª º	L	195	C3	╞ ╟	A	227	E3	µ τ	a
132	84	ê ë	„	164	A4	¿ ⌐	¤	196	C4	╚ ╔	Ä	228	E4	Φ θ	ä
133	85	è ï	…	165	A5	¬ ½	A	197	C5	╩ ╦	L	229	E5	Ω δ	l
134	86	î ì	†	166	A6	¼ ¡	¦	198	C6	╠ ═	C	230	E6	∞ ø	c
135	87	Ä Å	‡	167	A7	« »	§	199	C7	╬ ╧	Ç	231	E7	ε ∩	ç
136	88	É æ	□	168	A8	░ ▒	¨	200	C8	╨ ╤	C	232	E8	≡ ±	ç
137	89	Æ ô	‰	169	A9	▓ │	©	201	C9	╥ ╙	É	233	E9	≥ ≤	c
138	8A	ö ò	Š	170	AA	┤ ╡	S	202	CA	╘ ╒	E	234	EA	⌠ ⌡	é
139	8B	û ù	‹	171	AB	╢ ╖	<<	203	CB	╫ ╪	Ë	235	EB	÷ ≈	e
140	8C	ÿ Ö	S	172	AC	╕ ╣	¬	204	CC	┘ ┌	E	236	EC	° ∙	ë
141	8D	Ü ¢		173	AD	╗ ╝		205	CD	■ ▬	Í	237	ED	· √	e
142	8E	£ ¥	Ž	174	AE	╜ ╛	®	206	CE	▌ ▐	Î	238	EE	ⁿ ²	í
143	8F	₧ ƒ		175	AF	┐	Z	207	CF	▀	D	239	EF	■ □	î
144	90			176	B0		°	208	D0		Đ	240	F0		d

4			6			8			0		
145	91	'	177	B1	±	209	D1	N	241	F1	d
146	92	'	178	B2	?	210	D2	N	242	F2	n
147	93	"	179	B3	l	211	D3	Ó	243	F3	n
148	94	"	180	B4	´	212	D4	Ô	244	F4	ó
149	95	·	181	B5	µ	213	D5	O	245	F5	ô
150	96	–	182	B6	¶	214	D6	Ö	246	F6	o
151	97	—	183	B7	·	215	D7	×	247	F7	ö
152	98	□	184	B8	¸	216	D8	R	248	F8	÷
153	99	™	185	B9	a	217	D9	U	249	F9	r
154	9A	š	186	BA	s	218	DA	Ú	250	FA	u
155	9B	›	187	BB	››	219	DB	U	251	FB	ú
156	9C	s	188	BC	L	220	DC	Ü	252	FC	u
157	9D		189	BD	?	221	DD	Ý	253	FD	ü
158	9E	ž	190	BE	l	222	DE	T	254	FE	ý
159	9F	z	191	BF	z	223	DF	ß	255	FF	t

□Not supported on the current platform.

Top Level Intro

This page is printed before a new top-level chapter starts

Part

17 FAQ

17.1 General FAQ

- **Min. SW requirements**
- **Min. HW requirements**
- **What OS are supported?**
- **Is there a version for MAC?**
- **Do I have to pay any royalties?**
- **What is the upgrade policy?**
- **Can I distribute the files I create?**
- **Is there a book about MMB?**
- **How can I use 16bit (XP/Vista) multi-icons with MMB?**
- **What is the recommended graphic application to make my artwork?**
- **What is autorun?**
- **What are the lines in autorun.inf?**
- **How to enable Autorun on my computer?**
- **Why Autorun still doesn't work on some computers?**
- **If I want to make an audio CD with digital multimedia track, what is the format and software to do it?**
- **Why I can't play Audio Tracks from my CD-ROM?**
- **How to Shut-Down computer from script?**
- **Why my video doesn't play on other computer?**
- **Why my video doesn't play using MCI commands or MCI object while it plays fine in Windows Media Player?**
- **Why the text looks different on other computers?**
- **No sound in Flash (Video, Binding, MCI, external application or second instance of MMB application)**
  MMB crashes on Win9x/ME on page with Flash/Video or audio running over a 3rd party PlugIn.
  Another sound problems & crashes on Win9x/ME systems
- **Text object or Circle placed over Flash (HTML, ListBox, Binding,...) have an ugly white rectangle around it or transparent parts of images are not transparent**.

- **The Video object doesn't create preview (still) image.**
- **Slow loading of big/lengthy audio (mainly OGG) files**

  Incorrect time reported by some (VBR) audio files
- **I got a virus alert while scanning MMB folder or exe file made by MMB.**

Min. SW requirements:

Windows 98, 2000, XP, Vista (Windows ME is not recommended but usable;)
DirectX 6 or higher (for Video playback)
Windows Media Player 6.2 or higher (and appropriate codecs) - mainly if you use WMA, WMV, ASF or MPEG4 files

Min. HW requirements:

CPU (x86 comp.) - Intel/AMD 300MHz - (550MHz or better is recommended)
32MB RAM (playback) - 64MB RAM (development) - (128MB is recommended)
Sound card - it's not required if you don't use sound
Enough disk space for unpacking the embedded files and swapping.

What OS are supported?

MMB and the applications you create with it works on Windows Platform. It works on Windows 95, Windows 98, Windows 2000 and Windows XP. MMB can works also with Windows NT4 and Windows ME, but we do not recommend to use these OS because of two reasons...
- Windows NT is simply too old and doesn't fully support DirectX (required for Sound&Video)
- Windows ME is not very reliable OS, especially the HW drivers are not well optimized. In other words, Windows ME often crashes (from our own experience).

Is there a version for MAC or Linux?

No there is no version for MAC or Linux.
MMB is for Windows platform only.

Do I have to pay any royalties?

No. You need to just register once and you can make as many applications as you want.

What is the upgrade policy?

Currently the upgrades are free of charge. You simply install any new version over the previous and it should pick-up your registration.

Can I distribute the files I create?

Yes. If you are registered user, you can distribute your files free of any royalties etc...

Is there a book about MMB?

We are not aware of any book about MMB. However, MMB may be mentioned in books about multimedia generally. The best place to learn MMB is looking at a many samples on this or many others web pages dedicated to MMB. Also Discussion Board is a great help.

How can I use 16bit (XP/Vista) multi-icons with MMB?

There is currently no way to add the 16bit true color (XP/Vista) icon to your MMB project via Compile dialog. But you can replace the 16/256color icon after compiling the project. There is a tool to do this right in MMB installation folder called "". Simply run this tool, chose your compiled project, your XP/Vista multi-icon and simply press [Replace] button.

What is the recommended graphic application to make my artwork?

We recommend **Real-DRAW Pro** because it is the only program where you can make a fully functional page with buttons and then export it to MMB. However you can use other tools as a PaintShop Pro, PhotoShop etc...

What is autorun?

When a CD-ROM disk is inserted into the drive the Windows examines the disk looking for file **autorun.inf** .The autorun.inf is a simple ASCII text file and could be edited just with the notepad and it tells windows what program to run and what icon to show for the drive when it appears in the My Computer folder.

What are the lines in autorun.inf?

Inside the autorun.inf file you will see lines:
[autorun]
OPEN=autorun.exe
ICON=autorun.exe,0

The autorun.inf file must be on the root of the CD. However, you can even put that file to the root of a harddrive and then your icon of the harddrive will change.
Note: You can use for the icon directly icon file if you want so the third line will look like: ICON=myico.ico

How to enable Autorun on my computer?

Select Start Button, then Settings > Control panel. Doubleclick on System. Select Device Manager tab, expand the CD-ROM entry, click on appropriate CD-ROM drive

and click Properties. Select Settings tab and finally check the "Auto insert notification" box.a
Trick: If you as an user don't want the AutoRun application to start when you inserting new CD, hold down the Shift key while you insert the CD!

Why Autorun still doesn't work on some computers?

Make sure the that the Autorun is enabled on that computers. If you are sure that autorun is enabled on these computers, but the CD still won't auto-start, or MMB reports an error (something like "cannot find the file.."), the most probably reason of this problem is a long file name of exe or file name with empty spaces. Simply keep the length of file name of compiled exe in 8+3 format (8 characters for name and 3 for extension) and don't use empty spaces inside the file name! If you still wish to use empty spaces in the program file name, you need to edit the Autorun.inf and enclose the program name in double quotes. So instead of this..
OPEN=autorun.exe
use this..
OPEN="autorun.exe"

If I want to make an audio CD with digital multimedia track, what is the format and software to do it?

The industry standard format is **CD Extra**. In this format, the audio tracks are recorded in the first session and the second session is the data. You can create CD-Extra with most of the mastering software, Adaptec Easy CD Creator for example.
There is a also an older format named **Mixed Mode CD**. On such CD the audio and data are recorded in the same session where the data are as a track 1 with following audio tracks. The disadvantage of this is that a CD player can play a CD from track 2 skipping the track 1.

Why I can't play Audio Tracks from my CD-ROM?

You can't play Redbook audio (regular CD audio tracks) and read files off the CD at the same time. Redbook audio tracks are streamed off the CD at 1x speed. If your drive is faster than 1x (let's hope so), it actually slows down to play Redbook.
Most games do one of two things:

- Copy all game files to the hard disk during install. This ensures the fastest possible loading times, brisk play, and huge installs (Baldur's Gate install is over 500MB).
- Read all necessary data off the CD before starting the Redbook track. If your level/map/whatever data is small enough to fit entirely in RAM, you can get away with this.

MMB actually offers a third strategy: embed your data. This means your data gets unzipped to a temporary directory each time you run. If possible, I would recommend *not* using this option unless absolutely necessary.
I would probably use OGGs, since they are already so nicely supported by MMB. This sort of depends on what kind of machines you need to run on and how CPU intensive the rest of your project is. OGGs can eat some clock cycles, Redbook

audio is free.

How to Shut-Down computer from script?

...or Open Sound properties or... **Here** is a description file which allows you to do a lot of stuff with rundll32 command.

Why my video doesn't play on other computer?

Welcome to the real world of video on PC. The video on PC is a complex issue so we add an extended **Video FAQ**.

Why my video doesn't play using MCI commands or MCI object while it plays fine in Windows Media Player?

If the format is fine (previous Question) then make sure the video file has no space in its name. Since MCI internally comunicates using string, the space in the file name may be interpreted as beginning of another command. As such **My Video.mpg** may be wrongly interpreted, the **myvideo.mpg** should work. The best would be if you can rename the file using the simple 8.3 dos type and make sure there are no spaces or any strange characters in the name.

Why the text looks different on other computers?

You have to remember that not every body has the same font as you have. To ensure the font will be displayed properly use standard font such Arial or Times New Roman . You can use any other font, but then you have to include the ttf file on the CD and use InstallFont("<SrcDir>\myfont.ttf") command. Be aware that generally many font's should not be distributed - they are copyrighted. For more info about copyrights you can try FontMagic. To ensure that the Text object will appear the same on any systems you can just simple convert it to bitmap: Select the text object then from Menu - Object select Convert to Bitmap. This will create a bitmap object and hide the original text (in case you want to change something later).

Tip: If you don't need to show/hide text during the runtime you can simply combine all the text objects with the background to save space and memory. Menu Arrange - Combine - Objects with background

No sound in Flash (Video, Binding, MCI, TTS, external application or second instance of MMB application)

MMB crashes on Win9x/ME on page with Flash/Video or audio running over a 3rd party PlugIn.

Another sound problems & crashes on Win9x/ME systems

Unfortunately, we found, that some systems (Win9x/ME) with older or obsolete

VXD sound drivers cannot play multiple sounds at a time or initialize and use audio HW by multiple sources. In other words, on those systems cannot be run for example WinAmp and Flash/Video file with audio track. And the same problem you can get with Flash/Video/embedded application running from MMB application. Some users also reprots crashes on WinME. All these problems have one comon denominator...obsolete sound card drivers.

The only cure as we know (except disabling FMOD audio library) is to reinstall older VXD type of sound card drivers (often available on W9x/ME systems) with the latest WDM drivers.

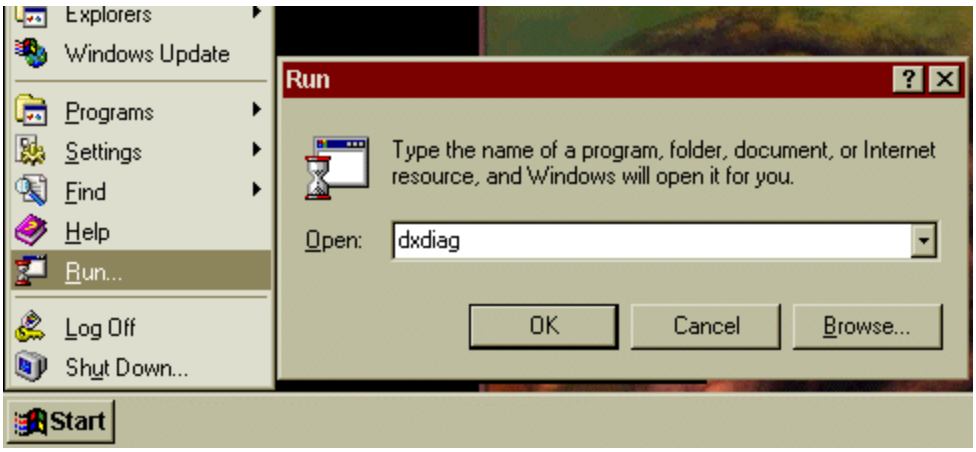
Go to the sound card manufacturer's website, and download and install the latest drivers from there. If you are not sure about the WDM compatibility of their drivers, just ask the manufacturer.
TIP: If the drivers offered on the manufacturer's web site are marked as Win98/ ME/Win2k/WinXP compatible (or packed in one file), then they are most probably WDM compatible.

For SoundBlaster users could be useful this link:
http://us.creative.com/support/downloads/

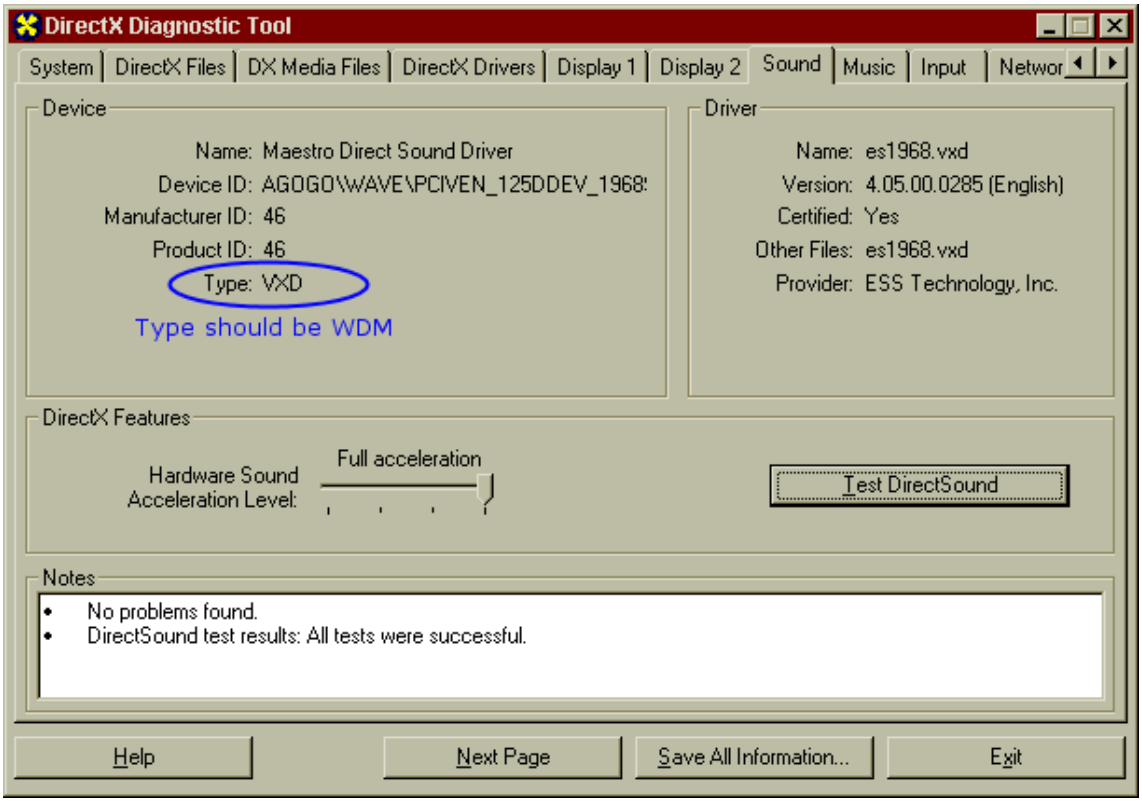
If your sound card is an on-board one, then check the driver CD that you got with PC. Or go to the motherboard manufacturer's website and download the drivers from there.

Also very good source of drivers (many of them are marked as WDM compatible) can be found here:
http://www.driverguide.com/

If you are not sure about your sound card manufacturer or driver type (VXD od WDM), then try the below steps:
Click on **Start**, then on **RUN**, then type **DXDIAG** and click **OK**..



Then click the **Sound** tab...



In "Driver" section you can find the driver name/version and in "Device" section you can check the type of already installed driver: VXD (older..the problematic one) or WDM (new one). The VXD type of driver is the type, which cause the mentioned problems with sound under some Win9x/ME systems.

Reinstall your current drivers with WDM ones and the sound problems will go away! If you don't like to tell the users of your product to reinstall their drivers, then there are also another two options:

1) Compile your project with **NoFMOD player**, but this will completely disable internal MMB audio playback, therefore AudioCommands and some other Sound related commands will no longer work.

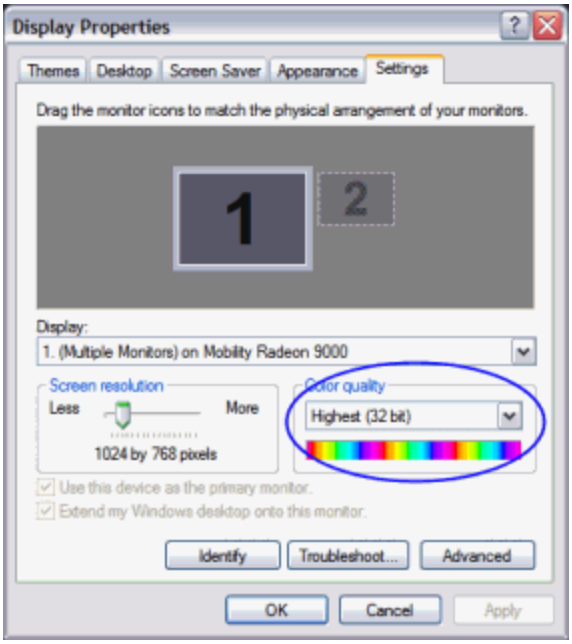
2) Use **FMODConfig("3")** command at application startup. This command will temporarily disable FMOD audio library, then you will able to play Sound from an external source (Flash, Video, TTS, etc...). But this will also disable the internal MMB audio playback. However, against the compiling your project with NoFMOD, in this case you will able to turn the FMOD audio library back ON and play the MMB internal audio.

Text object or Circle placed over Flash (HTML, ListBox, Binding,...) have an ugly white rectangle around it (transparent parts are not transparent)

This is a know limitation of all windowed objects like an HTML, Flash, ListBox, Binding and some other. Unfortunately, this limitation cannot be removed..at least not without significant CPU and memory usage and potential MMB slowdown.

The Video object doesn't create preview (still) image.

This is because the color depth of the loaded video file is too low (most probably 8bpp or lower). Unfortunately, creating preview image for video file like this is not currently supported.
Another cause of this issue can be too low display color depth - 16bit (High Color) or lower. Just increase color depth to 24 or 32 bit (True Color) and it will work as expected.



Slow loading of big/lengthy audio (mainly OGG) files

Incorrect time reported by some (VBR) audio files

This is because all streamed audio files (ogg, wma, asf) are loaded (by default) in a precise mode. In this mode MMB always loads entire file to get the correct total time of loaded file. This is useful mainly for VBR encoded files, which can returns wrong time.

For speeding up the reading of audio files you can turn this precise mode OFF. In a Designer you can turn it OFF in "*Tools>>Designer settings*" menu (uncheck "*MPEG Accurate...*" mode. For compiled apps you can uncheck the same option in "*File>>Compile*" menu. And finally, you can also turn this option ON/OFF in a runtime by **FMODConfig** script function.

Unfortunately, if you turn this option OFF, most of all VBR files will return incorrect total time. There is no way how to avoid this (it's an FMOD audio library feature). You have to carefully decide if you need fast loading of all (even big) audio files or precise total time of all (mainly VBR) files.

I got a virus alert while scanning MMB folder or exe file made by MMB.

In most cases, this is a false positive alert. Try update your antivirus application (mainly the virus definition file) and check the file again. If the problem persist, try to send the file to the author of antivirus for a detailed inspection. They can let you know if the file is infected or not.

17.2 RunDLL FAQ

This FAQ describes how to run Control Panel (and other) tools in Windows from MMB script.

Rundll32.exe is used by Windows to launch actions defined in DLLs. One of its capabilities is running Control Panel tools (applets).
Using Rundll32.exe, you can launch the applets from a CPL file, and you can cause the applet to open on a particular page.

Basic Script Syntax:
Run("rundll32.exe","shell32.dll,Control_RunDLL filename.cpl")

General syntax:
Run("rundll32.exe","shell32.dll,Control_RunDLL filename.cpl,@n,t")
where filename.cpl is the name of one of Control Panel's *.CPL files, n is the zero based number of the applet within the *.CPL file, and t is the number of the tab for multi paged applets.

To select a particular applet within the file, append a comma, an @ sign, and the number of the applet.
Launch the keyboard applet, which is the second applet within Main.cpl, you would use:
Run(" rundll32.exe","shell32.dll,Control_RunDLL main.cpl,@1")

Some multipage applets are designed so that they can be opened at a particular page. You must specify the applet number (@0 for a single-applet file). Then append a comma and the page number.
This line launches the Display Properties dialog and opens to the Appearance page (the third page; number two):
Run("rundll32.exe","shell32.dll,Control_RunDLL desk.cpl,@0,2")

Control panel tools and applets:
Accessibility Options access.cpl
Add New Hardware sysdm.cpl
Add new hardware Add/Remove Programs appwiz.cpl
Date/Time Properties timedate.cpl
Display Properties desk.cpl
FindFast findfast.cpl
Fonts Folder fonts
Internet Properties inetcpl.cpl
Joystick Properties joy.cpl
Keyboard Properties main.cpl keyboard
Microsoft Exchange mlcfg32.cpl
Microsoft Mail Post Office wgpocpl.cpl
Modem Properties modem.cpl
Mouse Properties main.cpl
Multimedia Properties mmsys.cpl
Network Properties netcpl.cpl (In Windows NT 4.0, Network properties is Ncpa.cpl, not Netcpl.cpl)
Password Properties password.cpl
PC Card main.cpl pc card
Power Management (Windows 95) main.cpl power

Power Management (Windows 98) powercfg.cpl
Printers Folder printers
Regional Settings intl.cpl
Scanners and Cameras sticpl.cpl
Sound Properties mmsys.cpl
Sounds System Properties sysdm.cpl

Examples:

Date/time applet, Time Zone tab:
Run("rundll32.exe","shell32.dll,Control_RunDLL TIMEDATE.CPL,@0,1")
Desktop applet, Screensaver tab:
Run("rundll32.exe","shell32.dll,Control_RunDLL DESK.CPL,@0,1")
Network applet, Protocols tab:
Run("rundll32.exe","shell32.dll,Control_RunDLL NCPA.CPL,@0,2")
Network applet, Adapters tab:
Run("rundll32.exe","shell32.dll,Control_RunDLL NCPA.CPL,@0,3")
System applet, Environment tab:
Run("rundll32.exe","shell32.dll,Control_RunDLL SYSDM.CPL,@0,2")
Sound
Run("rundll32.exe","shell32.dll,Control_RunDLL mmsys.cpl")
Joystick
Run("rundll32.exe","shell32.dll,Control_RunDLL joy.cpl")
Users Tool
Run("rundll32.exe","shell32.dll,Control_RunDLL inetcpl.cpl users")
Networks properties, Services tab
Run("rundll32.exe",shell32.dll,Control_RunDLL ncpa.cpl,,1")

Other tricks:

Format Disk
Run("rundll32.exe","shell32.dll,SHFormatDrive")

Copy Disk
Run("rundll32.exe","diskcopy.dll,DiskCopyRunDll")

Exit Windows
Run("RUNDLL32.EXE","USER.EXE,ExitWindows")

Restart Windows
Run("RUNDLL32.EXE","USER.EXE,ExitWindowsExec")

Logoff Windows 98 and run Explorer after relogon:
Run("RUNDLL32.EXE",SHELL32.DLL,SHExitWindowsEx 4")

Start "Add New Printer" wizard:
Run("RUNDLL32.EXE","SHELL32.DLL,SHHelpShortcuts_RunDLL AddPrinter")

Show Windows 9*'s "System setting changed, do you want to reboot now?" dialog
Run("RUNDLL32.EXE","SHELL.DLL,RestartDialog")

Open a file with Windows' "Open as" dialog:
Run("RUNDLL32.EXE","SHELL32.DLL,OpenAs_RunDLL filename")

Swap your mouse to left handed:
Run("RUNDLL32.EXE","USER.EXE,SwapMouseButton"
Windows NT: Run("RUNDLL32.EXE","USER32.DLL,SwapMouseButton")

Start DialUp Network:
Run("RUNDLL32.EXE","RNAUI.DLL,RnaDial exact name of dialer entry")

Install ScreenSaver
Run("RUNDLL32.EXE","DESK.CPL,InstallScreenSaver filename.scr")

Display the Fonts Folder in Explorer view
Run("RUNDLL32.EXE","shell32.dll,SHHelpShortcuts_RunDLL FontsFolder")

Displays the Create New Shortcut dialog.
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL apwiz.cpl,NewLinkHere path")

Displays the Install/Uninstall tab selected
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL appwiz.cpl,,1")

Send the HTML file to the printer
Run("RUNDLL32.EXE","mshtml.dll,PrintHTML htmlfile.html")

Open the URL with an associated HTML browser.
Simple script:
Run("www.mediachance.com")

If you want to make the browser window TOP/TOPMOST use this code (**Win2k/XP only**):
Run("<System>\rundll32.exe","TOP <System>\shell32.dll,ShellExec_RunDLL www.mediachance.com")

..or this (**Win2k/XP**)
Run("<System>\rundll32.exe","<System>\url.dll,FileProtocolHandler www.mediachance.com")

..or this (**Win9x**)
Run("<Windows>\rundll32.exe","<System>\url.dll,FileProtocolHandler www.mediachance.com")

Create new email in an associated email client (with predefined address and subject)
The same as for URL path, but with email address.

Simple script:
Run("mailto:your@address.com")

If you want to make the email client window TOP/TOPMOST use this code (**Win2k/XP only**):
Run("<System>\rundll32.exe","TOP <System>\shell32.dll,ShellExec_RunDLL mailto: your@address.com?Subject="This is a test mail"")

..or this (**Win2k/XP**)
Run("<System>\rundll32.exe","<System>\url.dll,FileProtocolHandler mailto: your@address.com?Subject="This is a test mail"")

..or this (**Win9x**)
Run("<Windows>\rundll32.exe","<System>\url.dll,FileProtocolHandler mailto: your@address.com?Subject="This is a test mail"")

Displays Internet Properties, General Tab
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL inetcpl.cpl,,0")

Microsoft Exchange Profiles general property page
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL mlcfg32.cpl")

Microsoft Postoffice Workgroup Admin property page
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL wgpocpl.cpl")

Multimedia/Audio property page
Run("RUNDLL32.EXE","shell32.dll,Control_RunDLL mmsys.cpl,,0")

17.3 Video FAQ

Avi or MPEG ?
The AVI format is supported in all versions of Windows. The MPEG format is not supported directly by OS and it require a MPEG software player installed which supports MCI.
New Windows Media Player can play many formats and MPEG as well. All new Windows 98 installation should have MPEG support already installed (or not...). For older versions of Windows you have to carry the Windows Media Player installation or DirectShow 6 and newer with you.
MPEG format has better compression and quality than standard AVI, however it needs more CPU to play the video.
To make it more difficult not all AVI's are the same. AVI uses Codec's to compress its data. (And in fact it can also use MPEG codec).

What is CODEC ?
You can use uncompressed AVI, but you will end up with hundreds of MB of data just for few minutes. Instead you should use compression/decompression software. CODEC is such software component. In order to be able to play the video encoded with any CODEC the component must be installed on the computer.
As always there are many Codecs, producing better or worse video quality. You can choose any one, but remember if you go wild, you will have to also install the CODEC on the user computer before he will be able to play your video.
There are already few Codecs installed on every Windows95, 98, 2000 by default: Cinepak, Indeo 3.2, Indeo 4, Indeo 5, Microsoft Video 1.
Each codec is identified by 4 letters code in AVI file (for example Cinepak is CVID)

- **Microsoft Video 1**

The original CODEC shipped with ancient Video for windows.
Quality: Bad, it is CODEC from times when 256 colors was a luxury.
Tip: The best is to avoid this CODEC.

- **Cinepak**

Cinepak was originally developed by Radius to play small movies on 386 from CD ROM.
Quality: Many years ago this was amazing, however today it doesn't compare well with newer CODECs
Tip: All Windows Versions will be able to play AVI encoded with Cinepak. To play the video, the CODEC need just very little CPU.
If you want to have video play back on any Windows machine Cinepak would be your choice.

- **Indeo 3.2**

Developed by Intel in 80's
Quality: Not much to say, the quality is in the range of Cinpak - which means 'medium' quality. It doesn't like fast movement.
Tip: All Windows versions should be able to play AVI encoded with Indeo 3.2. The time needed for compressing is less than Cinepak, however it needs faster computer to play back than Cinepak. (Today's Pentiums are far fast enough anyway)

- **Indeo 4 and 5**

Some name but different technology than Indeo 3.2.
Quality: Produces better results than Cinepak or Indeo 3.2, but needs fast Pentium

Tip: Most of the Windows 95 and 98 will have this CODEC already installed. However it requires a fast Pentium to play it back

● MPEG-1
MPEG-1 provides excellent quality (audio and video)
This format is very popular and most of the MPEG files you download from Internet are encoded in MPEG-1.
In addition, it is the format for VideoCD (VCD).

● MPEG-2
MPEG-2 is designed for broadcast quality digital audio and video. The image quality is outstanding. This is the format DVD uses. To play MPEG-2 videos you need fast Pentium or special hardware.

● MPEG-4
MPEG-4 is designed for high quality web streaming video. It is similar to H.263 (video conferencing CODEC)
The quality is excellent even at low data rates. Microsoft has its own implementation in the new ASF format **So which CODEC?**
Well again, that's your choice. There is no one for all. If you look at the previous paragraphs, it looks that if you want to play video on **any windows computer** you **use AVI** format encoded with **Cinepak**. The quality is not the best, but it will play on all computers for sure. If your video has low motion (talking head) you can also try **Indeo 3.2 CODEC**. Then if the quality is important use **MPEG,** but you have to let user install **Microsoft Windows Media Player** first (it is free)

DivX:) and other hacks
DivX is a simply Microsoft source code for the MPEG-4 from their SDK and hacked very slightly so it could be used in AVI format. DiviX as a codec is the same as used in ASF, however ASF is build as streaming format (you don't have to download the whole thing to play it - unlike AVI) so it has different structure. Because inside DiviX is a Microsoft code it may hapened that they will legaly go against DiviX and other copycats. Hard to say at this point if using DiviX today will bring any legal problems in the future... so read news if you decide to use this (or similar) codec. The other codecs go with various names such as AngelPotion etc.. but they all use the same code.

So which CODEC?
Well that's again your choice. There is no one for all. If you look at the previous paragraphs it looks that if you want to play video on any windows computer you use AVI format encoded with Cinepak. The quality is nothing to write home about, but it will play on all computers for sure. If your video has low motion (boring talking head) you can also try Indeo 3.2 CODEC. Then if the quality is important use MPEG, but you have to let user install Microsoft Windows Media Player first (it is free)

Top Level Intro

This page is printed before a new top-level chapter starts

Part

18 About...

Why I started making my own multimedia program?

What do we have for creating autorun menus for our software or front-end for Mixed-mode audio CD's in reasonable price?
There are many authoring tools on the market. Some of them are really good, but you don't want to spend $1000 to create autorun menu or CD browser. Some of them are cheaper, but you don't want to spend a month to learn how to use it.
I started making MMB to be able in 30 minutes create professional looking autorun browser. And here it is !
It couldn't be done without help of other people on the internet I never met.

So, what is MMB?
It's an authoring tool with great features and one of the best price in the industry!

It is great for small projects, autorun menus, cd browsers, multimedia applications for Mixed-mode CD's, toolbar applications etc.
However, if you going to make Encarta size multimedia, you should invest a bit more. Yet, you can create an autorun for it!

Happy Authoring!

Roman Voska & Mediachance Slovakia guys:
Peter Misik (aka orol)
Pavel Kudrys (aka odklizec)
Pavel Kotucek

Top Level Intro

This page is printed before a new
top-level chapter starts

Part

19 Credits

Thanks to all of you who sent me any suggestions or just nice words.

Special Thanks to:

Edd Twilbeck - for his ideas and encouragement and of course for the logo.
Derrick Yohn - for making this web space reality not just my dream.
Artur Svarc - for his beta testing.
Ronnie Toon - for sending me pages and pages of great suggestions.
Pierre Labelle - for Helping me with the Help File.
Mandryka, Gotlib and Sconi for their Bird and Worm Flash examples.

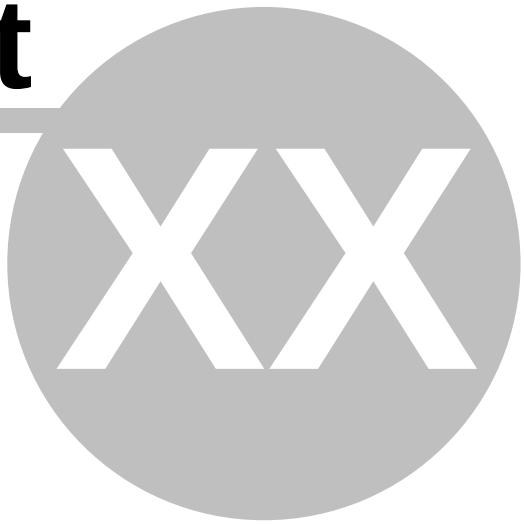
Files Mosquito Killer.fla/swf, Clock.fla/swf and Keyframing.fla/swf are taken from "Macromedia Flash 5" and "Macromedia Flash MX" samples. According the Macromedia documentation, you may reuse these files as you want, but they are not officially supported as part of the product.

My great wife Zuzana - for her endless patience (she really hates computers now!) and many other people...

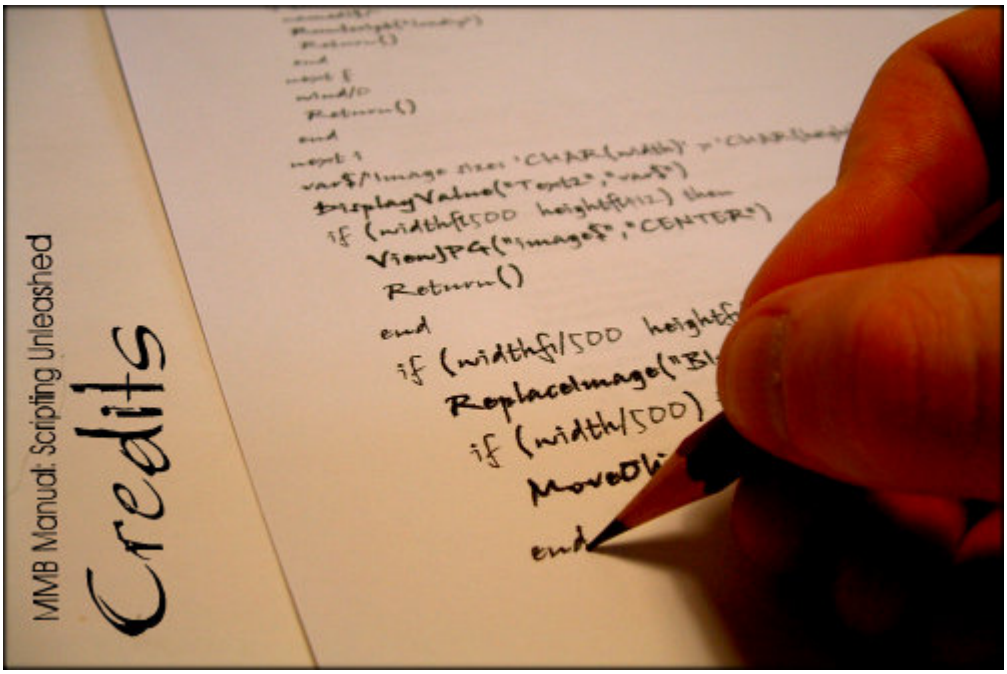
Top Level Intro

This page is printed before a new top-level chapter starts

Part



20 Scripting Unleashed Credits



***Scripting Unleashed* manual for Mediachance Multimedia Builder**

written by:

bokzy

bokzy@hi.htnet.hr

ICQ:12610160

Topics:

- Introduction
- Constants
- Variables
- Script Flow Functions
- Project Commands
- Object Commands
- Time & Script Commands
- Dialog Box Commands
- System Commands
- Audio Commands
- Image Commands
- Print Commands
- Video Commands
- MMB PlugIns: Operator's Manual

Mebirok

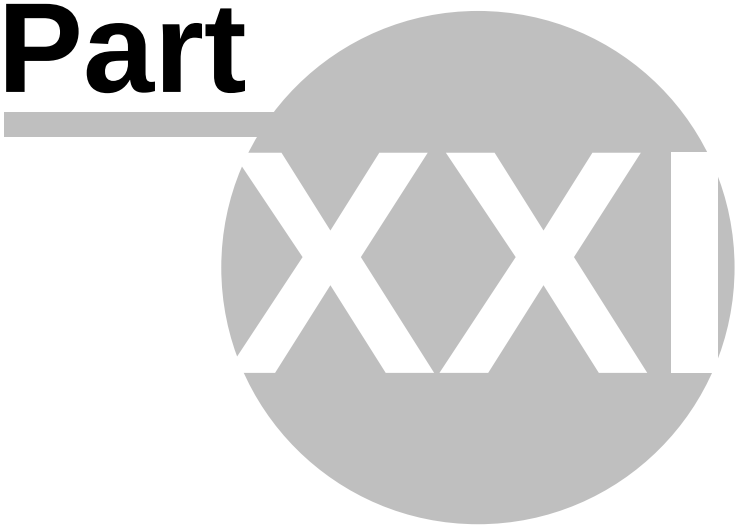
Topics:
- Matrix Commands
- Image Object

support@mobmultimedya.com

Mediachance Home Page: **http://www.mediachance.com**
MMB Forums: **http://www.mmbforums.com**

Top Level Intro

This page is printed before a new top-level chapter starts



21 Getting technical support

Thank you for purchasing and using Multimedia Builder. We hope you enjoy using it as much as we have enjoyed developing it.

Before contacting technical support, we suggest that you check the **FAQ** section of the Multimedia Builder help. The troubleshooting section lists several frequently asked questions (FAQs) regarding technical issues. It's also a good idea to go to the MMB community forum at **http://www.mmbforums.com**.
MMB forum is a very good (online) source of helpful people, ideas and solutions. It's probably the fastest way how to find a solution to your problem ;)

If you still cannot find the answer to your problem or you experienced an unexpected MMB behavior or crash, send us an email to **support@mmbgroup.com** .

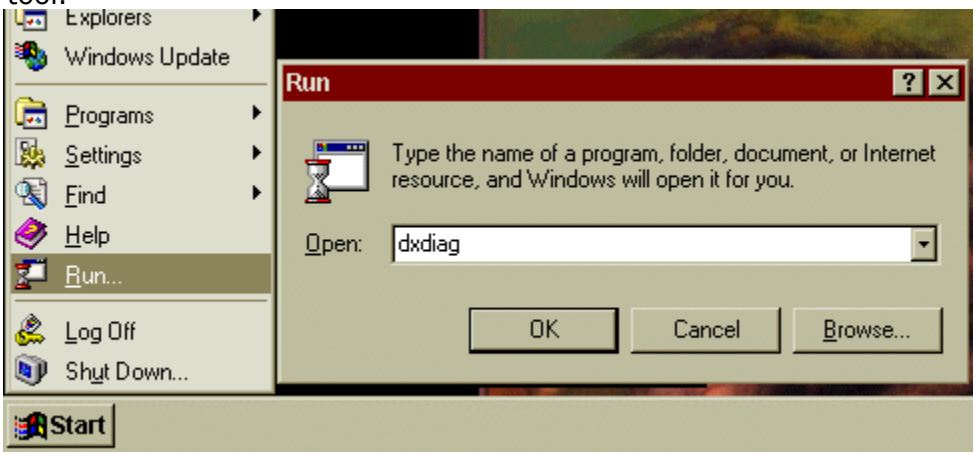
You can also contact us via Skype Chat

To make sure that you receive a fast and efficient answer from us it is important that you send us detailed description of the steps already taken and description of your computer system. The more we know about your problem and system the better we can help you.

The DXDIAG log from your computer would be really helpful. To obtain this system diagnostic file, follow the instructions below:

Important note: Please reboot your system prior to creating the DXDIAG file!

1. Click on Start button (button in windows task bar).
2. Click on Run menu item.
3. Type in DXDIAG and press Enter or click OK, this will open the DirectX diagnostic tool.



4. Run each available DirectX test on the Display Tab and on the Sound Tab.
5. Click on the **"Save All"** Information Button (save this file somewhere on HDD as txt file and name it dxdiag.txt).
6. Click on the **"More Help"** tab and open the Microsoft system information tool (MSInfo).
7. Click on File and choose Export (save this file somewhere on HDD as txt file and name it msinfo.txt). Please make sure to save this file as a .txt file and **NOT** as a .

nfo file.

8. Compress (with winzip, winrar or something else) both txt files and attach the compressed file to your tech. support email.

MMB project and screenshots describing the problem are also welcomed, but please, compress them (with winzip or winrar, etc..) before sending them to us. Also, please, make the mbd projects as simple as possible. In other words, reduce them to necessary objects and scripts needed for reproducing your problem. Huge projects with many buttons and scripts don't help us to find the matter of problem.