Multi Purpose GUI

Reference Manual

Author: Gruftikus
This Manual refers to v1.11 of MPGUI

I. Introduction

MPGUI (Multi Purpose GUI) is a graphical frontend for various command-line based tools such as Lightwaves *TESAnnwyn* or *TES4qLOD*. However, in order to be more flexible for possible future new tools, MPGUI does not use hard coded elements, but it is based on a scripting language (Multi Purpose Batch, or just "batch").

This manual describes the syntax of the batch files (`*.mpb`), and is therefore oriented towards modders, which like to use this GUI for their own projects, or in order to adapt MPGUI to other 3rd party tools.

If you think that features are missing, please feel free to contact me.

II. General syntax of the batch and start sequence

a. Batch files

The batch files are text files, therefore any text editor can be used for creating and changing these files.

Each batch file is structured in sections, as usual marked in squared brackets

```
[sectionname]
```

followed by a list of commands. Section names might start with an underscore (_), these sections are for temporary use and are not stored into a batch file, if one uses "File" → "Save batch".

The sections are encapsulated, sections which are never called because they are either used internally or connected to other commands, are ignored. It is, however, interesting to know that a section can appear multiple times, in this case all sections with the same name are merged.

Comments might be added to the batch file by ";" (like in *.ini) or with "#" (like in unix).

Each command might be followed by options. Options are starting with "-" and assign a value with "=". The value itself *can* be included in quotation marks, this is a *must* if the value has spaces.

A typical example for demonstration is to type a "hello world":
```
GUIConsoleEcho -text="Hello World!"
```

The batch file syntax supports flags (i.e. variables). This becomes rather important when converting the GUI elements, like checkboxes, drop down menu, etc., to command-line arguments, and to include conditions to GUI elements.

Each command might be preceded by a test on a flag condition, which is invoked by a "@" followed by the flag itself. A simple test on the validity of a flag is:
```
@flagname MyCommand...
```

A test if the flag is disabled (negation) is indicated by an additional "!", like
```
@!flagname MyCommand...
```

The test for a specific value of a flag can be realized as follows:
```
@flagname=value MyCommand...
```

Flag names are globals. Choose a unique name, in case that somebody opens more then one batch file.

b. The mpgui_autoload.mpb

After the start of MPGUI (also by double clicking on a mpb file), the first step is to search for the auto load configuration file ("`mpgui_autoload.mpb`"). The search order is as follows: MPGUI looks first in the same path (working directory, "wdir"), than in wdir\ini, than in wdir\ini\mpgui, and finally in the installation path of MPGUI. This allows to drop the auto load configuration file in the path of your mpg file, and override the standard setting.

Although this is a special batch file, because it is loaded always prior to all other batch files, the syntax is the same as for normal batch files.

c. The start sequence

As mentioned above, the first step is that MPGUI loads the auto load batch file. Here it does not matter if MPGUI is started via double-clicking on a *.mpb file, or opened itself. In the latter case, MPGUI does not allow any action until a normal batch file has been loaded.

After the batch file has been read, MPGUI will examine its section `[_gamemode]`. This allows to extent or overwrite settings done in the auto load file.

In the following, MPGUI tries to evaluate the game mode, by comparing the current working directory to the defined search pattern defined with `SetGameSearchPattern` (see section III for more details).

The search for the game mode is disabled, if one adds the lines

```
[_gamemode]
GameMode -name="..."
```

in the batch file. **This can be helpful if MPGUI is not able to determine the game mode.**

The next step is to search for the section `[MPGUI]` and executes the commands listed there. This aim of this section is to extent the GUI with tabs. Keep in mind that, if the user opens several batch files, these sections are merged. Therefore, the use of "doonce" variables is advised.

d. The GUI elements

The batch file also describes which GUI elements are shown, and their content. In most of the GUI

commands, a unique name has to be defined. This name is also used as a flag name. Hence, by defining a GUI element, one defines a corresponding flag. These two items are correlated: by changing the GUI content, the flag value is changed, and vice versa.

Also the tabs itself have unique names, which is defined in the `[MPGUI]` section with the `GUITab` command. Here, the name is connected to a specific section with the same name, where the elements of the tab are defined.

e. The tab sections

The tab sections itself are evaluated directly after the batch file is loaded, and the `GUITab` command connects the tab to its section. But it is important to know, that the section is re-evaluated each time, after GUI elements have been changed by the user.

So lets us discuss a complete example[1]. Your application should have a tab named "My Application", and the section name should be `[myapplication]`. The GUI should have a single check box, and a drop down menu for a hypothetical resolution. The drop-down menu should be enabled only, if the check box is marked.

For the meaning of the various commands, see section III.

(N.B. that the line breaks which appear below are not allowed in the real file)

```
[MPGUI]

GUIRequestVersion -value="0.99"

@!hello_doonce GUIConsoleEcho
               -text="This is my hello world example"
@!hello_doonce GUITab
               -help="Options for the world"
               -name="myapplication" -text="My application"
SetFlag -name=hello_doonce -hidden

[myapplication]

SetOption -noskipinfo

GUITextBox -text="Select the options" -name=hello_intro -vdist=10

GUICheckBox -name=hello_option1
            -text="Use the resolution"
        -help="If this is enabled, you can select a resolution"

GUIDropDown -name=hello_option2
            -help="Select a resolution" -vdist=3
GUIDropDownItem -parent=hello_option2 -name=hello_option3
                -text="Low"
GUIDropDownItem -parent=hello_option2 -name=hello_option4
                -text="High" -select
GUIEnable -name=hello_option2
```

1 Available in the template directory of the MPGUI installation.

```
@!hello_option1 GUIDisable -name=hello_option2
```

It should be noted here, that the `GUIEnable` is needed, because the section is evaluated in loop mode, otherwise the drop down menu will never be re-enabled.

III. Command list

a. Game management

`AddGame -n=... -name=...`

Adds a new game to the list of known games (mainly for `mpgui_autoload.mpb`).

`-n`	Game number
`-name`	The unique name of the game. MPGUI will use this name for determining the game path from the Windows Registry. So it should match the official name.

`SetGamePluginFile -n=... -value=...`

Defines the location of the plugin file (for `mpgui_autoload.mpb`).

`-n`	Game number
`-value`	The full path of the plugin file.

`SetGameSearchPattern -n=... -value=...`

Defines a search pattern to determine the game mode (for `mpgui_autoload.mpb`).

`-n`	Game number
`-value`	A string which is used to be compared to the working directory.

`SetGameStdWorldspace -n=... -value=...`

Default world space selection (for `mpgui_autoload.mpb`).

`-n`	Game number
`-value`	World space

`GameMode -name=...`

Forces MPGUI to use a specific game mode. Can be used in a selected `mpgui_autoload.mpb`

or as one of the first commands in the `[_gamemode]` section of the user batch file.

-name	Name of the game (as defined with `AddGame`)

SetPath -value=...

Set the search path when using the File → Batch open menu. Can be used in a selected `mpgui_autoload.mpb` or later. N.b. that MPGUI always switches to the game path after the plugins have been read.

-value	Full directory path

---

b. Miscellaneous batch commands

SetOption [option list]

Enables a specific option.

-noskipinfo	Avoids the "skip" info messages if a flag test has been failed.

SetFlag -name= [-value="..."] [-hidden]

Set a flag to a new value.

-name	The name of the flag. Flag names should use alphanumeric characters only and the underscore "_" (like in C)
-value	[Optional] String value of the flag. If no value is set, the flag is a boolean flag.
-hidden	[Optional] Hides this flag from the `_flaglist`, and prevents that the flag is written to a batch files. Can be used for temporary flags (like "doonce"-flags).

---

c. Miscellaneous GUI commands

GUIRequestVersion -value=...

Tests MPGUI for a version. A messagebox will appear if the MPGUI version of the user is too old.

-value	The version (must be a number)

GUIConsoleEcho -text=...

Dumps a text string on the console tab. Can include flag names.

-text	The text to be written.

Example:

```
GUIConsoleEcho -text="Hello World!"
GUIConsoleEcho -text="Flag dummy is: $dummy"
```

GUIMessageBox -title=... -text="..."

Makes a pop-up message box.

-title	The title of the box
-text	The content of the box.

GUIExec -exe="..."

This command finally starts the hosted tool. One should keep in mind, that this command should be never written in the batch without a prior flag condition. Please write your batch such that only one `GUIExec` can be executed for a single evaluation of the section. Otherwise it will happen, that the GUI will either stop the previous command (if you are lucky), or the GUI can freeze. This is because the GUI uses multiple threads to execute the commands.

After the `GUIExec` is called, the GUI opens a shell, and executes the command defined in "`-exe`". The output is dumped to the console, and all other tabs are disabled, to prevent any interference of the user with the settings, and to prevent multi-execution (e.g. by clicking again on a button).

-exe	The command to be executed. The default working directory is the game directory
-dumpbatch	[Optional] Dumps the complete batch to stdout (stdin for the command). Needed for tes4ll.

d. Graphics GUI commands

GUITextBox -name=... -text="..." -help="..."
[-vdist=...] [-width=...] [-height=...] [-sameline]
[-input|-fileinput]

Adds a single-line textbox to the tab

-name	Unique name of the GUI element.
-text	The content of the textbox. For input text boxed, this also the default value.
-input	Defines an input text field (i.e. editable by the user).

-fileinput	Defines an input text field (a file open menu appears when clicking on it).
-help	Adds a tooltip for this element
-vdist	Vertical distance to the previous GUI element (in pixel)
-width	Width of the element (in fraction of the window)
-height	Height of the element (in pixel)
-sameline	Make no line break and draw the element in the same line

---

GUICheckBox -name=... -text="..." -help="..." [-vdist=...] [-width=...] [-height=...] [-sameline] [-select]

Adds a single-line textbox to the tab

-name	Unique name of the GUI element.
-text	The text written after check box.
-select	Check box is marked by default, and flag is enabled.
-help	Adds a tooltip for this element
-vdist	Vertical distance to the previous GUI element.
-width	Width of the element (in fraction of the window)
-height	Height of the element (in pixel)
-sameline	Make no line break and draw the element in the same line

---

GUIDropDown -name=... -help="..." [-vdist=...] [-width=...] [-height=...] [-sameline]

Adds the frame (parent) for a drop-down menu

-name	Unique name of the GUI element.
-help	Adds a tooltip for this element
-vdist	Vertical distance to the previous GUI element.
-width	Width of the element (in fraction of the window)
-height	Height of the element (in pixel)
-sameline	Make no line break and draw the element in the same line

---

GUIDropDownItem -name=... -parent=... -text="..." [-select]

Adds a new item to the drop-down menu "`-parent`"

-name	Unique name of the GUI element.

-parent	The parent GUIDropDown element
-text	The text of this item (must be unique in the parent)
-select	[Optional] Selects this item by default

GUIButton -name=... -text="..." -help="..." [-vdist=...]

Adds a button to the tab. The corresponding flag "`-name`" is disabled. After the button is clicked, the flag is enabled, and the tab sections are executed. After the single execution, the flag is again disabled.

-name	Unique name of the GUI element.
-text	The text written inside the button.
-help	Adds a tooltip for this element
-vdist	Vertical distance to the previous GUI element.

GUIEnable -name=... [-select]

This enables the element "`-name`" (i.e. make it accessible for the user)

-name	Name of the GUI element. Can be one of the tab elements, or of the tabs itself.
-select	[Optional] Enables at the same time the flag. In addition, for a checkbox, the mark is set. *The handling of drop down menu items is not yet implemented*

GUIDisable -name=... [-unselect]

This disables the element "`-name`" (i.e. it will be grayed out)

-name	Name of the GUI element. Can be one of the tab elements, or of the tabs itself.
-unselect	[Optional] Disables at the same time the flag. In addition, for a checkbox, the mark is unset. *The handling of drop down menu items is not yet implemented*

IV. Internal flags

Internal flags are always starting with "_"

$_modlist

The complete comma-separated list of plugin files, as selected in MPGUI

`$_flaglist`

The complete comma-separated flaglist, without hidden flags, and without "_modlist"

`$_worldspace`

The worldspace, as selected in the "worldspace tab"

`$_worldspaceid`

The worldspace-ID (decimal number), as selected in the "worldspace tab"

`$_worldspaceidhex`

The worldspace-ID (hexadecimal number), as selected in the "worldspace tab"

`$_gamemmode`

The selected game mode, with the name as defined in the autoload batch file.

`$_gamedir`

The game directory (with the trailing "\Data", or "\Data Files" for Morrowind)

V. Remark

Suggestions? Features missing?

Please contact me (gruftikus@tesnexus, @bethsoft, @tesalliance, @scharesoft)