EasyMCM

May 01, 2019

Contents

CHAPTER 1

Components

1.1 Templates

1.1.1 Template

A Template is the top level component in mcmData. It determines the overall layout of the menu. Can be created with a table or a string (name).

Fields:

name (string) The name field is the mod name, used to register the MCM, and is displayed in the mod list on the lefthand pane.

headerImagePath (string) Set headerImagePath to display an image at the top of your menu. Path is relative to `Data Files/`

Optional.

onClose (function) Set this to a function which will be called when the menu is closed. Useful for saving variables, such as `TableVariable`. For example:

```
onClose = (
    function()
        mwse.log("saving config to json")
        mwse.saveConfig(configPath, localConfig)
    end
),
```

Optional.

Example:

```
local template = EasyMCM.createTemplate("My Mod name")
template:saveOnClose(configPath, config)


--with image header
local template = EasyMCM.createTemplate{
    name = "My mod name",
    headerImagePath = "Path/to/image.dds"
}
```

1.2 Pages

1.2.1 ExclusionsPage

An ExclusionsPage is a highly specialised page used for making whitelists/blacklists. It is made up of two lists: The righthand list displays objects that are filtered using custom settings. It could be a list of NPCs, or weapons, or plugins etc. Clicking on an item in the righthand list will transfer it to the lefthand list, and is also added to a config table. Both lists can be searched using searchbars, and buttons can be added for different filters to appear in the lefthand list

Parent Class: Page

Fields:

label (string) The label field is displayed in the tab for that page at the top of the menu.

description (string) Displayed at the top of the page above the lists.

Optional.

toggleText (string) Overwrite the text for the button that toggles filtered items from one list to another.

Optional: defaults to "Toggle Filtered".

leftListLabel (string) Overwrite the label for the lefthand list.

Optional: defaults to "Blocked".

rightListLabel (string) Overwrite the label for the righthand list

Optional: defaults to "Allowed".

showAllBlocked (boolean) When set to true, the left list shows all items in the blocked table, regardless of the filter being used.

variable (Variable) The Variable used to store blocked list entries.

filters (table) A list of filters, which appears as buttons between the two lists. At least one filter is required. See the below example to see the different filter types.

Example:

```
template:createExclusionsPage{
    label = "Exclusions Page",
    description = "Description",
    toggleText = "Toggle", --Optional: default "Toggle Filtered"
    leftListLabel = "Blacklist", --Optional: default "Blocked"
    rightListLabel = "Whitelist", --Optional: default = "Allowed"
```

(continues on next page)

(continued from previous page)

```
    showAllBlocked = false, --OptionalL default = false
    variable = EasyMCM.createTableVariable{
        id = "blocked",
        table = config,
    },

    filters = {

        --Filter by plugins to exclude entire mods
        {
            label = "Plugins",
            type = "Plugin",
        },

        --Filter by object type
        {
            label = "Ingredients",
            type = "Object",
            objectType = tes3.objectType.ingredient,
        },

        --Define object filters for more specific filters
        {
            label = "Helmets",
            type = "Object",
            objectType = tes3.objectType.armor,
            objectFilters = {
                slot = tes3.armorSlot.helmet
            }
        },
        {
            label = "Blunt Weapons",
            type = "Object",
            objectType = tes3.objectType.weapon,
            objectFilters = {
                type = tes3.weaponType.bluntOneHand
            }
        },
        {
            label = "Essential NPCs",
            type = "Object",
            objectType = tes3.objectType.npc,
            objectFilters = {
                isEssential = true
            }
        },

        --define your own callback for a custom filter
        {
            label = "GMSTs",
            callback = (
                function(self)
                    local gmstNames = {}
                    for gmst, _ in pairs(tes3.gmst) do
                        table.insert(gmstNames, gmst)
                    end
                    return gmstNames
```

(continues on next page)

(continued from previous page)

```
                end
            )
        },

    }--/filters
}
```

1.2.2 FilterPage

A FilterPage is a SideBarPage with additional functionality: The components container is a vertical scroll pane with a searchbar. This is expecially useful if you have a large or unknown number of settings.

Parent Class: Page

Fields:

label (string) The label field is displayed in the tab for that page at the top of the menu.

description (string) Default sidebar text when no sidebarComponents is defined

Optional.

sidebar (MouseOverPage) The object that holds components in the sidebar. Add to it like any other component

Example:

```
local filterPage = template:createFilterPage("Filter Page")
filterPage.sidebar:createCategory("Heading")
filterpage.sideBar:createInfo{ text = "Description of my mod." }
```

1.2.3 Page

A Page is a container that holds other components. It acts a bit like a page on your web browser, in that you have tabs across the top of your menu to selecta page to view. Pages must go in the pages table in your template.

The default page is a simple container, it is recommended you use the SideBarPage for basic settings.

Page Subclasses:

- SideBarPage
- FilterPage
- ExclusionsPage

Fields:

label (string) The label field is displayed in the tab for that page at the top of the menu.

Optional: defaults to "Page {number}"

noScroll (boolean) When set to true, the page will not have a scrollbar. Particularly useful if you want to use a ParagraphField, which is not compatible inside of scroll panes.

Example:

```
EasyMCM.createTemplate("My mod")

template:createPage() --sets name to "Page 1"

template:createPage("Page two")

template:createPage() --sets name to "Page 3"

template:createPage{ label = "Page 4" }
```

1.2.4 SideBarPage

A SideBarPage is a special page type that includes an additional container used to display mouseover information for components

Children in the *components* list can have a *description* text field, which will display in the sidebar when that component is moused over. When no component is moused over, the sidebar will either display the text in the *description* field of the page, unless the sidebarComponents table exists, in which case it will display those components instead.

Parent Class: Page

Fields: ——-.

label (string) The label field is displayed in the tab for that page at the top of the menu.

description (string) Default sidebar text when no sidebarComponents is defined

Optional.

sidebar (MouseOverPage) The object that holds components in the sidebar. Add to it like any

noScroll (boolean) When set to true, the page will not have a scrollbar. Particularly useful if you want to use a ParagraphField, which is not compatible inside of scroll panes.

Example:

```
--With sidebar
local template = EasyMCM.createTemplate("My Mod")
local page = template:createSideBarPage()
page:createButton{
    buttonText = "Button",
    description = "When hovering over the button, this text will be shown in the␣
↪sidebar"
}
page.sidebar:createButton{ buttonText = "press button" } }

--With basic description
template:createSideBarPage{
    label = "Sidebar Page",
    description = "Default sidebar text"
}
```

Base class: Page

1.3 Categories

1.3.1 Category

A Category has a header and a list of components. Components within a category are indented. Categories can be nested indefinitely. A category is a good way to organise settings within a page.

label (string) The label field is displayed in the tab for that Category at the top of the menu.

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

Example:

```
page:createCategory("Category name")

page:createCategory{
    label = "Settings",
    description = "A bunch of settings"
}
```

1.3.2 SideBySideBlock

A SideBySideBlock is a category that arranges its settings horizontally instead of vertically. Particularly useful in conjunction with buttons that have no labels.

label (string) The label field is displayed in the tab for that Category at the top of the menu.

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

Example:

```
local category = page:createSideBySideBlock("List of buttons")
category:createButton{ buttonText = "Button A" }
category:createButton{ buttonText = "Button B" }
```

Base class: Category

1.4 Settings

A setting is a component that the player interacts with in some way. Interacting with a setting may update a Variable, or it may call a custom function.

1.4.1 Button

A button is the most basic of settings. You click the button, and it calls the function defined in the callback field. A number of more advanced button classes extend this class.

Parent Class: Setting

Button Subclasses:

- OnOffButton
- YesNoButton
- KeyBinder

Fields:

label (string) Text shown next to the button.

Optional.

buttonText (string) Text shown inside the button.

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

callback (function) Function that is called when the button is pressed.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage (boolean) The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createButton{
    label = "Reset Actors",
    buttonText = "Okay",
    inGameOnly = true,
    callback = (
        function(self)
            tes3.messageBox("Resetting Actors")
            tes3.runLegacyScript({command = "ResetActors"})
        end
    ),
}

--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createButton{
    block,
    {
        buttonText = "Okay"
    }
}
```

1.4.2 KeyBinder

This button allows the player to bind a key combination for use with hotkeys.

The player presses the hotkey button, a prompt asks them to press a key (or key combination using Shift, Ctrl or Alt), and the current key combo is displayed in the popup until they press "Okay" to confirm.

Key combos are stored in the following format:

```
{
    keyCode = tes3.scanCode.{key},
    isShiftDown = true,
    isAltDown = true,
    isControlDown = true,
},
```

Parent Class: Button

Fields:

label (string) Text shown next to the button.

Optional.

description If in a SideBarPage, description will be shown on mouseover.

Optional.

allowCombinations (boolean) If true, keybinds can allow combos of Shift+x, Alt+x or Ctrl+x

Optional.

variable (Variable) The Boolean variable being toggled.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage The message shown if restartRequired is triggered.

Optional.

callback (function) Function that is called when the button is pressed.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createKeyBinder{
    label = "Assign Keybind",
    allowCombinations = true,
    variable = EasyMCM.createTableVariable{
        id = "keybind_1",
        table = config,
        defaultSetting = {
```

(continues on next page)

(continued from previous page)

```
            keyCode = tes3.scanCode.k,
            --These default to false
            isShiftDown = true,
            isAltDown = false,
            isControlDown = false,
        }
    }
}

--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createKeyBinder{
    block,
    {
        label = "Assign Keybind",
        allowCombinations = true,
        variable = EasyMCM.createTableVariable{
            id = "keybind_1",
            table = config,
            defaultSetting = {
                keyCode = tes3.scanCode.k,
                --These default to false
                isShiftDown = true,
                isAltDown = false,
                isControlDown = false,
            }
        }
    }
}
```

1.4.3 OnOffButton

Toggles a Variable between true and false, with the buttonText switching between the strings “On” and “Off”.

Parent Class: Button

Fields:

label (string) Text shown next to the button.

Optional.

description If in a SideBarPage, description will be shown on mouseover.

Optional.

variable (Variable) The Boolean variable being toggled.

callback (function) Function that is called when the button is pressed.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createOnOffButton{
    label = "Turn on and off?",
    variable = EasyMCM.createTableVariable{
        id = "enabled",
        table = config
    }
}

--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createOnOffButton{
    block,
    label = "Turn on and off?",
    variable = EasyMCM.createTableVariable{
        id = "enabled",
        table = config
    }
}
```

1.4.4 ParagraphField

A ParagraphField allows you to enter a multi-line of text. Press Enter to submit or Shift+Enter to enter a new line.

Parent Class: TextField

Fields:

label (string) Text shown above text field.

Optional.

variable (Variable) The variable which stores the setting value.

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

height (number) Fixes the height of the paragraph field to a custom value.

Optional.

sNewValue (string) Text shown when the setting is updated. This can be formatted with a '%s' which will be replaced with the new value.

Optional: default "New value: '%s'"

callback (function) Function that is called when setting is updated.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage (boolean) The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createParagraphField{
    label = "Paragraph input",
    variable = EasyMCM.createTableVariable{ id = "text", table = config },
    height = 150
}


--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createParagraphField(
    block,
    {
        label = "Paragraph input",
        variable = EasyMCM.createTableVariable{ id = "text", table = config },
        height = 150
    }
}
```

1.4.5 Slider

A slider for setting numerical values.

Parent Class: Setting

Fields:

label (string) Text shown next to the button. If left as a normal string, it will be shown in the form: *[label]: [value]*. If the string contains a '%s' format operator, the value will be formatted into it.

Optional.

variable (Variable) The variable which stores the setting value.

min (number) Minimum value of slider.

Optional: default 0

max (number) Maximum value of slider

Optional: default 100

step (number) How far the slider moves when you press the arrows.

Optional: default 1

jump (number) How far the slider jumps when you click an area inside the slider.

Optional: default 5

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

callback (function) Function that is called when the button is pressed.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage (boolean) The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createSlider{
        label = "Time Scale",
        description = "Changes the speed of the day/night cycle.",
        min = 0,
        max = 50,
        step = 1,
        jump = 5,
        variable = EasyMCM:createGlobal{ id = "timeScale"}
    }
}

--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createButton{
    block,
    {
        label = "Time Scale",
        description = "Changes the speed of the day/night cycle.",
        min = 0,
        max = 50,
        step = 1,
        jump = 5,
        variable = EasyMCM:createGlobal{ id = "timeScale"}
    }
}
```

1.4.6 TextField

A TextField allows you to enter a single line of text and submit with a button. You can also force it to only accept numbers with the numbersOnly field.

Parent Class: Setting

TextField Subclasses:

- ParagraphField

Fields:

label (string) Text shown above text field.

Optional.

variable (Variable) The variable which stores the setting value.

description (string) If in a SideBarPage, description will be shown on mouseover.

Optional.

numbersOnly (boolean) If true, the input will only accept numbers

Optional: default false

press (function) A function that happens before the update() function. Can be overriden to add a confirmation message before updating.

sNumbersOnly (string) The text shown in a messageBox when the user entered an invalid input when numbersOnly is true.

Optional: default "Value must be a number."

sNewValue (string) Text shown when the setting is updated. This can be formatted with a '%s' which will be replaced with the new value.

Optional: default "New value: '%s'"

callback (function) Function that is called when setting is updated.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage (boolean) The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createTextField{
    label = "Text input",
    variable = EasyMCM.createTableVariable{ id = "text", table = config },
}


--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createTextField(
    block,
```

(continues on next page)

(continued from previous page)

```
    {
        label = "Text input",
        variable = EasyMCM.createTableVariable{ id = "text", table = config },
    }
}
```

1.4.7 YesNoButton

Toggles a Variable between true and false, with the buttonText switching between the strings "Yes" and "No".

Parent Class: Button

Fields:

label (string) Text shown next to the button.

Optional.

description If in a SideBarPage, description will be shown on mouseover.

Optional.

variable (Variable) The Boolean variable being toggled.

callback (function) Function that is called when the button is pressed.

Optional.

inGameOnly (boolean) If true, this setting is disabled in main menu.

Optional.

restartRequired (boolean) If true, a message will display prompting the user to restart their game when the setting changes.

Optional.

restartRequiredMessage The message shown if restartRequired is triggered.

Optional.

Example:

```
--EasyMCM:
local template = EasyMCM.createTemplate("My mod")
local page = template:createPage()
page:createYesNofButton{
    label = "Yes or No?",
    variable = EasyMCM.createTableVariable{
        id = "enabled",
        table = config
    }
}

--Adding to a non-easyMCM element
block = e:createBlock()
EasyMCM.createYesNofButton{
    block,
    label = "Yes or No?",
```

(continues on next page)

(continued from previous page)

```
    variable = EasyMCM.createTableVariable{
        id = "enabled",
        table = config
    }
}
```

CHAPTER 2

Variables

Variable Classes:

2.1 ConfigVariable

A ConfigVariable fetches a json file from a given path and stores the variable in the `id` field in that file. If no config file of that name exists yet, it will create it.

The ConfigVariable saves to the config file every time the setting is updated. It is generally recommended you use TableVariable and save the config using template.saveOnClose() instead, especially if you are using a setting where updates happen frequently such as with sliders.

Parent Class: Variable

2.1.1 Fields

id (string) Key in the config file used to store the variable.

path (string) Location of the config file relative to `MWSE/config/`.

defaultSetting(any) If `id` does not exist in the config file, it will be initialised to this value.

Optional.

inGameOnly (boolean) If true, the setting containing this variable will be disabled in the main menu.

Optional.

restartRequired (boolean) If true, updating the setting containing this variable will notify the player to restart the game.

Optional.

restartRequiredMessage (string) The message shown if restartRequired is triggered.

Optional.

Example:

```
EasyMCM.createVariable{
    id = "varID",
    path = "MyMod/config",
},
```

2.2 Global

A variable connected to a Morrowind Global.

Base Class: Variable

2.2.1 Global Subclasses:

- GlobalBoolean

2.2.2 Fields

id (string) The id of the Morrowind Global.

Example:

```
EasyMCM.createGlobal("timeScale")
```

2.3 GlobalBoolean

A variable connected to a Morrowind Global. Treats 0 as false and >=1 as true.

Base Class: Global

2.3.1 Fields

id (string) The id of the Morrowind Global.

Example:

```
EasyMCM.createGlobalBoolean("HeartDestroyed")
```

2.4 PlayerData

Stores the variable on the Player reference. This results in the value of the variable being local to the loaded save file. If users may want different values set for different games, this is a good Variable to use.

Settings using PlayerData are in-game only by default, as the Player reference can only be accessed while a game is loaded.

Parent Class: Variable

2.4.1 Fields

id (string) Key of entry used on the Player data table.

path (string) Path to `id` relative to `tes3.player.data`. It's best to at least store all your PlayerData fields in a table named after your mod to avoid conflicts.

defaultSetting (any) If `id` does not exist in on the playerData field, it will be initialised to this value. best to initialise this yourself though, as this will not create the value until you've entered the MCM.

Optional.

restartRequired (boolean) If true, updating the setting containing this variable will notify the player to restart the game.

Optional.

restartRequiredMessage (string) The message shown if restartRequired is triggered.

Optional.

Example:

```
EasyMCM.createPlayerData{
    id = "varID",
    path = "myModName.mcmSettings",
    defaultSetting = true
}
```

2.5 TableVariable

A TableVariable takes a lua table and stores the variable in the `id` field in that table.

The TableVariable can be used to save multiple changes to a config file only when the menu is closed. Load the config file with mwse.loadConfig(), pass it to any TableVariables in your MCM, then save it using the template:onSaveClose() function.

Parent Class: Variable

2.5.1 Fields

class (string) The name of this class.

id (string) Key in the config file used to store the variable.

table (table) The table to save the data to.

defaultSetting(any) If `id` does not exist in the table, it will be initialised to this value.

Optional.

inGameOnly (boolean) If true, the setting containing this variable will be disabled in the main menu.

Optional.

restartRequired (boolean) If true, updating the setting containing this variable will notify the player to restart the game.

Optional.

restartRequiredMessage (string) The message shown if restartRequired is triggered.

Optional.

Example:

```
local configPath = "myMod.config"
local config = mwse.loadConfig(configPath)
if not config then
    config = {}
end
local template = EasyMCM.createTemplate("My Mod")
template:saveOnClose(configPath, config)

EasyMCM.createTableVariable{
    id = "varID",
    table = config,
},
```

2.6 Variable

A Variable is an object which determines the type and location of the value being set by a setting. It could be a field on a table, or inside a config file, etc.

The Variable base class can be used for custom variables by defining the *get* and *set* fields to retrieve and save the value to a specified location. Variable subclasses exist for default behaviour.

2.6.1 Variable Subclasses:

- Global
- PlayerData
- ConfigVariable
- TableVariable

2.6.2 Fields

class (string) The name of this class.

get (function) Function to retrieve the variable value.

set (function) Function to save the variable value.

inGameOnly (boolean) If true, the setting containing this variable will be disabled in the main menu.

Optional.

restartRequired (boolean) If true, updating the setting containing this variable will notify the player to restart the game.

Optional.

restartRequiredMessage (string) The message shown if restartRequired is triggered.

Optional.

Example:

```
EasyMCM.createVariable{
    get = (
        function(self)
            return tes3.getCurrentWeather().index or 0
        end
    ),
    set = (
        function(self, newVal)
            if tes3.player then
                tes3.getWorldController().weatherController:switchImmediate(newVal)
            end
        end
    ),
    inGameOnly = true
},
```

Base class: Variable

CHAPTER 3

Tutorial

3.1 Prerequisites

3.1.1 MWSE

MWSE dev version >2.1 is required for EasyMCM. You can find documentation on MWSE here.

3.1.2 An MWSE Mod

If you have an MWSE mod you want to add a menu to, then great! If not, create a new folder in `Data Files/MWSE/mods`, and call it the name of your mod. Then in your new folder, create a file called `main.lua`. This is where we will create our new MCM.

3.2 Creating an MCM

Add the following to your *main.lua*:

```
local function registerModConfig()
    EasyMCM = require("easyMCM.EasyMCM")
    local template = EasyMCM.createTemplate("Basic MCM")
    local page = template:createPage()
    local category = page:createCategory("Settings")
    category:createButton{ buttonText = "Press" }
    EasyMCM.register(template)
end
event.register("modConfigReady", registerModConfig)
```

An MCM needs, at minimum, a template, at least one page, and probably a setting or two.

Check out the list of Components that you can build your MCM out of, and the types of Variables that can be attached to settings.

Advanced Example:

```
--fetch config file
local confPath = "config_test"
local config = mwse.loadConfig(confPath)
if not config then
    config = { blocked = {} }
end

local function registerModConfig()
    --get EasyMCM
    local EasyMCM = require("easyMCM.EasyMCM")
    --create template
    local template = EasyMCM.createTemplate("Advanced MCM")
    --Have our config file save when MCM is closed
    template:saveOnClose(confPath, config)
    --Make a page
    local page = template:createSideBarPage{
        sidebarComponents = {
            EasyMCM.createInfo{ text = "An info field in the sidebar" },
            EasyMCM.createButton{ buttonText = "Press this button" }
        }
    }
    --Make a category inside our page
    local category = page:createCategory("Settings")

    --Make some settings
    category:createButton({
        buttonText = "Hello",
        description = "A useless button",
        callback = function(self)
            tes3.messageBox("Button pressed!")
        end
    })

    category:createSlider{
        label = "Time Scale",
        description = "Changes the speed of the day/night cycle.",
        variable = EasyMCM:createGlobal{ id = "timeScale" }
    }

    --Make an exclusions page
    local exclusionsPage = template:createExclusionsPage{
        label = "Exclusions",
        description = (
            "Use an exclusions page to add items to a blacklist"
        ),
        variable = EasyMCM:createTableVariable{
            id = "blocked",
            table = config,
        },
        filters = {
            {
                label = "Plugins",
                type = "Plugin",
            },
            {
                label = "Food",
```

(continues on next page)

(continued from previous page)

```
                type = "Object",
                objectType = tes3.objectType.ingredient,
            }
        }
    }

    --Register our MCM
    EasyMCM.register(template)
end

--register our mod when mcm is ready for it
event.register("modConfigReady", registerModConfig)
```

3.3 Creating Components

There are three ways to create a component with EasyMCM.

You can add the component to another EasyMCM object:

```
local page = template:createPage()
page:createButton{ buttonText = "Hello" }
```

You can add the component to a vanilla element. Be warned that easyMCM components require the parent element to have the correct formatting to appear. They tend to work best with menus that utilise widthProportional and autoHeight:

```
local block = e:createThinBorder()
--note the `.` instead of `:`, very important:
EasyMCM.createButton{
    block,
    { buttonText = "Hello" }
}
```

And you can construct the component object without creating the UI elements, then use the create() function later to create the element itself. You can see an example of this method in the advanced Example above, where we define a *sidebarComponents* table with an info and button, but we don't actually create those components yet:

```
local button = EasyMCM.createButton{ buttonText = "Hello" }

--Then create the component element as a child of some vanilla element:
local block = e:createThinBorder()
button:create( block )
```

EasyMCM is a UI Library for Morrowind, allowing modders to quickly generate Mod Config Menus and preconfigured UI elements without touching UI code.

Example:

```
local function registerModConfig()
    EasyMCM = require("easyMCM.EasyMCM")
    local template = EasyMCM.createTemplate("Basic MCM")
    local page = template:createPage()
    local category = page:createCategory("Settings")
    category:createButton{ buttonText = "Press" }
```

(continues on next page)

(continued from previous page)

```
    EasyMCM.register(template)
end
event.register("modConfigReady", registerModConfig)
```

See the Tutorial for a more detailed walkthrough.