Buffs

Overview

The Lua Buff Class provides a way to represent buffs in Lua scripts. Buffs are temporary enhancements or negative effects (debuffs) applied to game characters (players and npcs). For example, a deffensive cooldown would be a buff and a poison applied to an enemy would be a debuff. This is very basic, but we aim to be beginner friendly and welcome everybody abroad ★

The Way Raw Buffs and Debuffs Work

In WoW, buffs and debuffs are essentially a (usually) large list for each game_object. As you can imagine, checking every frame every buff for every unit in the game is very expensive CPU-wise. So, if you were to use the raw buffs given by Blizzard to check every frame, for example, if you have one buff up that would make you deal more dmg, you would quickly realize how your FPS take a slight hit, since your PC has to also go through all other irrelevant buffs. This scalates with the number of units for which you are checking buffs or debuffs. So, let's say you are an affliction warlock and your script is checking how many units around you have Corruption; or maybe you are a resto druid and you are checking how many dots from you does everyone on your party have: in this case, your FPS will take a big hit.

How To: Retrieve Buffs and Debuffs Information

To get the buffs and debuffs info from a unit, we have 2 choices:

1 - Raw

1 - Using Raw Buffs and Debuffs

For the reasons explained in the previous point, this method is not recommended in most cases. However, you are still allowed to use it (at your own risk).

Acessing to the game_object function get_buffs() we can get the table of buffs. For the debuffs, we can just use get_debuffs(). This function will return a table with the following elements:

.buff_name (string)
.buff_id (integer)
.count (integer)
.expire_time (number)
.duration (number)
.type (integer)
.caster (game_object)

Here is the code to print all (raw) buffs information for a given unit:

```lua
---@param target game_object
local function print_buffs_info(target)
    --- buff_name         - string
    --- buff_id           - integer
    --- count             - number
    --- expire_time       - number
    --- duration          - number
    --- type              - integer
    --- caster            - game_object
    local buffs = target:get_buffs()

    for k, buff in ipairs(buffs) do
        core.log("Buff name: " .. buff.buff_name)
        core.log("Buff id: " .. tostring(buff.buff_id))
        core.log("Buff Stacks: " .. tostring(buff.count))
        core.log("Buff Expire Time: " .. tostring(buff.expire_time))
        core.log("Buff Duration: " .. tostring(buff.duration))
        core.log("Buff Type: " .. tostring(buff.type))
        core.log("Buff Caster: " .. buff.caster:get_name())
        core.log("- - - - - - - - - - - - - - - - - - - - - - - - - -")
    end
end
```

Here is the code to print all (raw) debuffs information for a given unit:

```lua
---@param target game_object
local function print_debuffs_info(target)
    --- buff_name         - string
    --- buff_id           - integer
    --- count             - number
    --- expire_time       - number
    --- duration          - number
    --- type              - integer
    --- caster            - game_object
    local debuffs = target:deget_buffs()

    for k, debuff in ipairs(buffs) do
        core.log("Buff name: " .. debuff.buff_name)
        core.log("Buff id: " .. tostring(debuff.buff_id))
        core.log("Buff Stacks: " .. tostring(debuff.count))
        core.log("Buff Expire Time: " .. tostring(debuff.expire_time))
        core.log("Buff Duration: " .. tostring(debuff.duration))
        core.log("Buff Type: " .. tostring(debuff.type))
        core.log("Buff Caster: " .. debuff.caster:get_name())
        core.log("- - - - - - - - - - - - - - - - - - - - - - - - - -")
    end
end
```

This is what you will be seeing in the console after running the showcased code (in this case, the parameter was local_player)

2 - Buff Manager Module

2 - Using Our Custom-Made Buffs Module

For the reasons already explained, we recommend using this module to check buffs information, since we have a special cache system that reduces FPS impact to almost zero. The usage is very simple, you just have to import 2 modules: the enums module (although this is optional), and the buff_manager module.
Then, we just have to use either the buff_manager:get_buff_data() function or the buff_manager:get_debuff_data() function, depending on if we want to check the information of a buff or a debuff.

So, to get a specific buff information, you could use this code:

```lua
---@type buff_manager
local buff_manager = require("common/modules/buff_manager")
---@type enums
local enums = require("common/enums")

---@param target game_object
local function print_buffs_info(target)
    local buff_info = buff_manager:get_buff_data(target, enums.buff_db.BARSKIN)

    core.log("Is Buff Active: " .. tostring(buff_info.is_active))
    core.log("Buff Remaining: " .. tostring(buff_info.remaining)) -- in MILISECONDS (ms)
    core.log("Buff Stacks: " .. tostring(buff_info.stacks))
    core.log("- - - - - - - - - - - - - - - - - - - - - - - - - -")
end
```

And to get a specific debuff information, you could use this code:

```lua
---@type buff_manager
local buff_manager = require("common/modules/buff_manager")
---@type enums
local enums = require("common/enums")

---@param target game_object
local function print_buffs_info(target)
    local debuff_info = buff_manager:get_debuff_data(target, enums.buff_db.BARSKIN)

    core.log("Is Buff Active: " .. tostring(debuff_info.is_active))
    core.log("Buff Remaining: " .. tostring(debuff_info.remaining)) -- in MILISECONDS (ms)
    core.log("Buff Stacks: " .. tostring(debuff_info.stacks))
    core.log("- - - - - - - - - - - - - - - - - - - - - - - - - -")
end
```

Parameters:
1- target (game_object) (the unit to check the buffs/debuffs)
2- buff_ids (table of integers) (this is a TABLE)
3- custom_cache_duration (number) (the unit to check the buffs/debuffs)

> NOTE
> The parameters are the same, for both get_debuff_data and get_buff_data functions.

Brief Explanation Of The Parameters
1- Target:
This is just the game_object that we want to analyze the buffs or debuffs of.
2- Buff IDs:
This is a table that contains the possible IDs of the same buff or debuff. For example, let's say you have the buff named "Shiny Day". There might be something that alters the ID of this buff, in most cases a spec change. However, the buff is still "Shiny Day", and its functionality might even remain the same. This is a good reason why we are using a table here, so we can catch the buff or debuff even if it has multiple possible IDs. However, the most important reason for us to use a table is so that the buffs are compatible across all game versions, since all of them are expected to be supported in the future. For example, the "Rend" debuff might have a different ID in WoW Classic than in Retail. If the buff or debuff that you are trying to analyze only has one ID, you can just pass a table containing this one ID.
3- Custom Cache Duration:
This is an optional parameter and should usually not be modified. This is useful in some specific cases where you want the cache to renew very quickly (or slowly), for some buffs that only appear a very brief of time, for example. However, this is very rare and take into account that modifying this parameter might affect FPS.

This is how our console would look like after running the previous code passing an active buff id as parameter:

How To: Recommended Workflow With Buffs and Debuffs

The Way We Work
We offer you multiple tools to check all buffs and debuffs information of any unit, in the "Developer Tools" tab, in the main menu, you will find the following tools:

The buttons functionalities are self-explanatory. Everything will be printed to the console uppon pressing. This is useful to find, for example, the ID of a buff or a debuff that is not added to our buffs db (located in enums) and that you might need.

Another option is to use the Debug Panel, located just below "Benchmark Plugin" in the "Developer Tools" menu.

The buttons are, again, self-explanatory. Upon pressing them, a new window (made completely in LUA using our custom GUI. Check Custom UI to learn how to make your own visuals) will appear showing all the information available for the selected unit in the "Mode" combobox.

> TIP
> When you already know all the buffs and the debuffs that you are going to use, and have all of their IDs stored or know that they are in the buffs database, you can begin using them. If you are going to need the same buff or debuff information in multiple places of your code, maybe you should consider sepparating the said buff or debuff information into a function that you can call multiple times.
> For example:

```lua
---@return boolean
---@param target game_object
local function has_hunters_mark(target)
    local buff_data = buff_manager:get_debuff_data(target, enums.buff_db.HUNTERS_MARK)
    return buff_data.is_active
end

--- Or, alternatively, using a custom buff ID table:

---@return boolean
---@param target game_object
local function has_hunters_mark(target)
    local possible_hunters_mark_debuff_ids = {257284, }
    local buff_data = buff_manager:get_debuff_data(target, possible_hunters_mark_debuff_ids)
    return buff_data.is_active
end
```

Then, we can call has_hunters_mark(target) as many times as we want more easily.