Project Sylvanas Documentation
Search Ctrl K

Game Object - Functions
Game Object - Code Examples
Buffs
Spell Book
Spell Book - Raw Functions
Spell helper
Graphics
Graphics - Functions
Graphics - Notifications
Menu Elements
Input
Geometry
Control Panel
Vectors
Vector 2
Vector 3
Libraries
Spell Prediction
Combat Forecast Library
Health Prediction Library
Unit Helper Library
Target Selector
PvP Helper Library
PvP UI Module Library
Inventory Helper
Dungeons Helper
Custom UI
Custom UI Functions ▯
Barney's Basic Guide (With

🏠 > 👨‍💻 Scripting Reference > Documentation > Dev Docs > Libraries > Unit Helper Library

Unit Helper Library

Overview

The **Unit Helper** module provides a collection of utility functions for working with game units in Sylvanas. This module simplifies tasks such as checking unit states, retrieving unit information, and working with groups of units. Below, we'll explore its core functions and how to effectively utilize them.

Including the Module

As with all other LUA modules developed by us, you will need to **import** the unit helper module into your project. To do so, you can use the following lines:

```lua
---@type unit_helper
local unit_helper = require("common/utility/unit_helper")
```

> ⚠ WARNING
>
> To access the module's functions, you **must** use `:` instead of `.`
>
> For example, this code is **not correct**:
>
> ```lua
> ---@type unit_helper
> local unit_helper = require("common/utility/unit_helper")
>
> local function is_target_dummy(unit)
>     return unit_helper.is_dummy(unit)
> end
> ```
>
> And this would be the **corrected code**:
>
> ```lua
> ---@type unit_helper
> local unit_helper = require("common/utility/unit_helper")
>
> local function is_target_dummy(unit)
>     return unit_helper:is_dummy(unit)
> end
> ```

Functions

Unit Classification Functions 📋

`is_dummy(unit: game_object) -> boolean`

Returns `true` if the given unit is a training dummy.

```lua
local is_training_dummy = unit_helper:is_dummy(unit)
if is_training_dummy then
    core.log("The unit is a training dummy.")
end
```

`is_blacklist(npc_id: number) -> boolean`

Returns `true` if the `npc_id` is inside a blacklist. For example, incorporeal beings which should be ignored by the target selector.

```lua
local is_blacklisted = unit_helper:is_blacklist(npcID)
if is_blacklisted then
    core.log("The NPC is blacklisted.")
end
```

`is_boss(unit: game_object) -> boolean`

Determines if the unit is a boss with certain exceptions.

`is_valid_enemy(unit: game_object) -> boolean`

Determines if the unit is a valid enemy with exceptions.

`is_valid_ally(unit: game_object) -> boolean`

Determines if the unit is a valid ally with exceptions.

Combat State Functions 🛡️

`is_in_combat(unit: game_object) -> boolean`

Determines if the unit is in combat with certain exceptions.

Health and Resource Functions ❤️

`get_health_percentage(unit: game_object) -> number`

Returns the health percentage of the unit in a format from `0.0` to `1.0`.

```lua
local health_pct = unit_helper:get_health_percentage(unit)
core.log("Unit's health percentage: " .. (health_pct * 100) .. "%")
```

`get_health_percentage_inc(unit: game_object, time_limit?: number) -> number, number, number, number`

Calculates the health percentage of a unit considering incoming damage within a specified time frame. This function uses the Health Prediction Module

> ⓘ NOTE
>
> This function returns 4 values:
>
> 1- Total -> The value that you will want usually. It's the future health percentage that you willhave according to the incoming damage.
> 2- Incoming -> The amount of incoming damage.
> 3- Percentage -> The current HP percentage.
> 4- Incoming Percentage -> The HP percentage taking into account just the incoming damage and not current health.

```lua
local total, incoming, percentage, incoming_percent = unit_helper:get_health_percentage_inc(unit, 5)
core.log("Health after incoming damage: " .. (total * 100) .. "%")
```

> 💡 TIP
>
> Generally, you will use this function as follows:
>
> ```lua
>     if unit_helper:get_health_percentage_inc(ally_target) < 0.45 then
>         is_anyone_low = true
>     end
> ```
>
> Just taking into account the first value.

`get_resource_percentage(unit: game_object, power_type: number) -> number`

Gets the power (resource) percentage of the unit. See PowerType Enum for `power_type` values.

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

local get_local_player_energy_pct(local_player)
    local energy_percentage = unit_helper:get_resource_percentage(local_player, enums.power_type.ENERGY)
    core.log("Unit's Energy percentage: " .. (energy_percentage * 100) .. "%")
    return energy_percentage
end

```

Role Determination Functions 🏹

`get_role_id(unit: game_object) -> number`

Determines the role ID of the unit (Tank, DPS, Healer).

`is_healer(unit: game_object) -> boolean`

Determines if the unit is healer.

> ⚠ WARNING
>
> Might not work in open world (if the target is not a party member).

`is_player_in_arena() -> boolean`

Determines if the local player is in arena.

`is_player_in_bg() -> boolean`

Determines if the local player is in BG.

`is_tank(unit: game_object) -> boolean`

> ⚠ WARNING
>
> Might not work in open world (if the target is not a party member).

Determines if the unit is in the tank role.

> 💡 TIP
>
> Below, an example on how to retrieve the tank from your party:
>
> ```lua
> ---@type unit_helper
> local unit_helper = require("common/utility/unit_helper")
>
> ---@param local_player game_object
> ---@returns game_object | nil
> local get_tank_from_party(local_player)
>     local allies_from_party = unit_helper:get_ally_list_around(local_player:get_position(), 40.0, true, tr
>
>     for k, ally in ipairs(allies_from_party) do
>         local is_current_ally_tank = unit_helper:is_tank(ally)
>
>         if is_current_ally_tank then
>             return ally
>         end
>     end
>
>     return nil
> end
>
> ```

Group Unit Functions 👥

> 💡 TIP
>
> See Object Manager for a more in-depth explanation and code examples.

`get_enemy_list_around(point: vec3, range: number, incl_out_combat?: boolean, incl_blacklist?: boolean) -> table<game_object>`

Returns a list of enemies within a designated area. This function is performance-friendly with Lua core cache.

- `point` : The center point to search around.
- `range` : The radius to search within.
- `incl_out_combat` : Include units that are out of combat. Defaults to `false` .
- `incl_blacklist` : Include units that are blacklisted. Defaults to `false` .

`get_ally_list_around(point: vec3, range: number, players_only: boolean, party_only: boolean) -> table<game_object>`

Returns a list of allies within a designated area. This function is performance-friendly with Lua core cache.

- `point` : The center point to search around.
- `range` : The radius to search within.
- `players_only` : Include only player units.
- `party_only` : Include only party members.

Previous
« Health Prediction Library

Next
Target Selector »

Overview
Including the Module
Functions
Unit Classification Functions 📋
`is_dummy(unit: game_object) -> boolean`
`is_blacklist(npc_id: number) -> boolean`
`is_boss(unit: game_object) -> boolean`
`is_valid_enemy(unit: game_object) -> boolean`
`is_valid_ally(unit: game_object) -> boolean`
Combat State Functions 🛡️
`is_in_combat(unit: game_object) -> boolean`
Health and Resource Functions ❤️
`get_health_percentage(unit: game_object) -> number`
`get_health_percentage_inc(unit: game_object, time_limit?: number) -> number, number, number, number`
`get_resource_percentage(unit: game_object, power_type: number) -> number`
Role Determination Functions 🏹
`get_role_id(unit: game_object) -> number`
`is_healer(unit: game_object) -> boolean`
`is_player_in_arena() -> boolean`
`is_player_in_bg() -> boolean`
`is_tank(unit: game_object) -> boolean`
Group Unit Functions 👥
`get_enemy_list_around(point: vec3, range: number, incl_out_combat?: boolean, incl_blacklist?: boolean) -> table<game_object>`