Welcome to GML+ - Essential Extensions for the Way You Work

At XGASOFT, we love GML! The GameMaker Markup Language is both easy for beginners and powerful for veterans. It's like eating your cake and having it too. And over more than 20 years of development, it's only gotten better and better. But 20 years of organic growth can leave a few holes behind.

Unified. Simplified. Amplified.

That's where GML+ comes in: GML+ is a collection of useful functions and built-in variables designed to fill the gaps in vanilla GML and supplement it with quality-of-life enhancements it should've had all along.

Like GML itself, GML+ is not set in stone and will continue to grow with its parent language. What's more, most GML+ functions come with few external dependencies. You can pick-and-choose only the functions you need for your project. Now *that's* the best of both worlds!

Once you go GML+, you won't want to go back!

It features...

- Automatic integration into your project--**just import and start using it!**
- **Built-in variables and macros**, such as:
  - **Extended mouse variables**, unifying behavior with instances
  - **Frametime constants** (easy `delta_time`)
- **Dozens of useful functions** you always wished were part of vanilla GML, such as:
  - **Extended array functions**, unifying behavior with data structures
  - **Extended data structure functions**, unifying behavior among different types
  - **Extended sprite functions**, unifying behavior with new GMS2 features

    - **Extended angle functions** (robust `lengthdir`)
    - **Interpolation with easing**, including custom bezier curves (robust `lerp`)
    - **Timer functions** (robust `alarm`)
    - **Filesystem functions** (robust `file_find_*`)
    - **Even/odd number functions**
    - **Hex color functions**
- **... And more!** See the complete documentation for details!

To get started, choose a topic from the navigation menu to learn more.

Version History

1.0.0

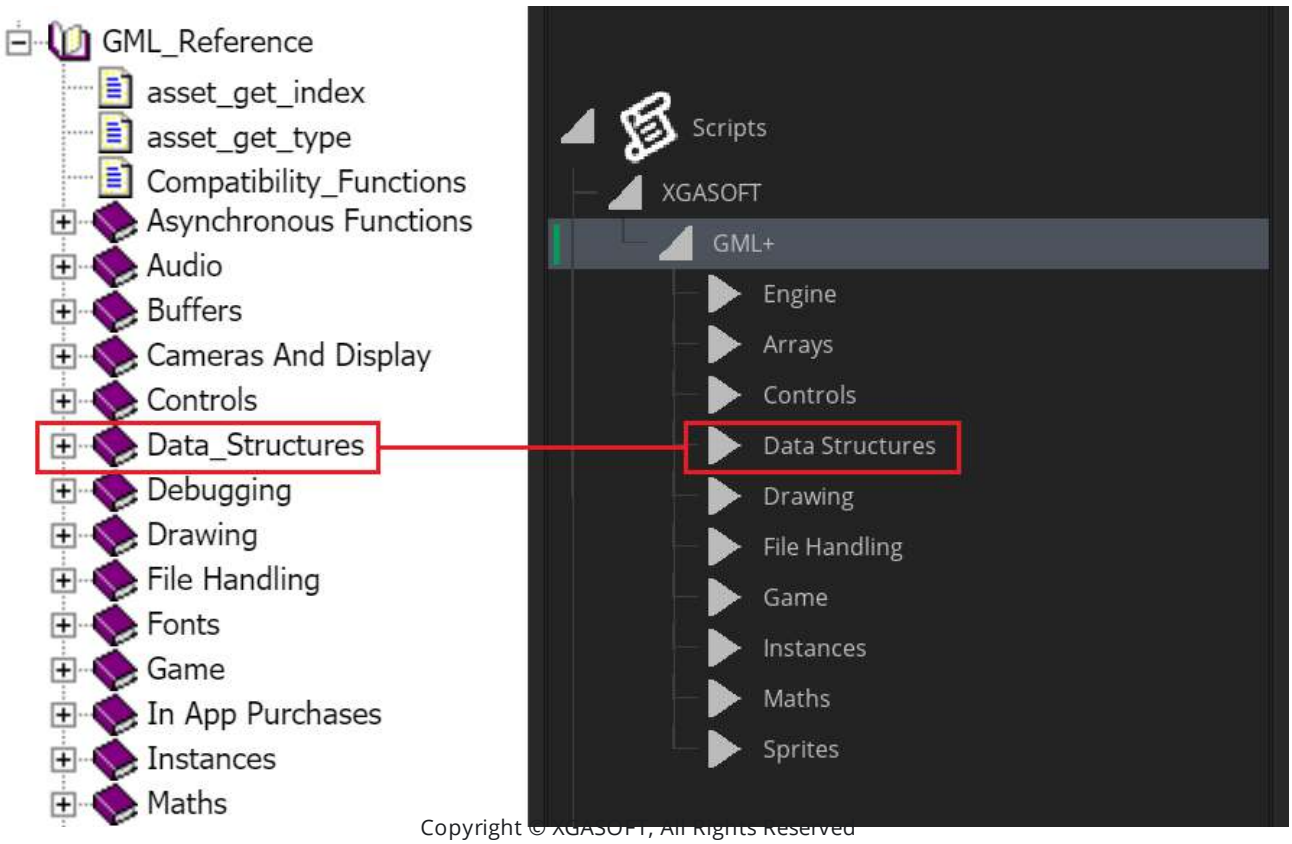
- Initial release

GML+ Reference Guide

GML+ is a collection of useful functions and built-in variables designed to fill the gaps in vanilla GML. This gives the package an unusually broad scope that may seem daunting at first. But don't worry! GML+ is organized to be as easy to navigate as possible.

Vanilla-Compliant Folder Structure

Most of GML+ resides in your project's Scripts folder. There, you'll find GML+ functions are organized in a structure that mimics the vanilla GML reference guide.

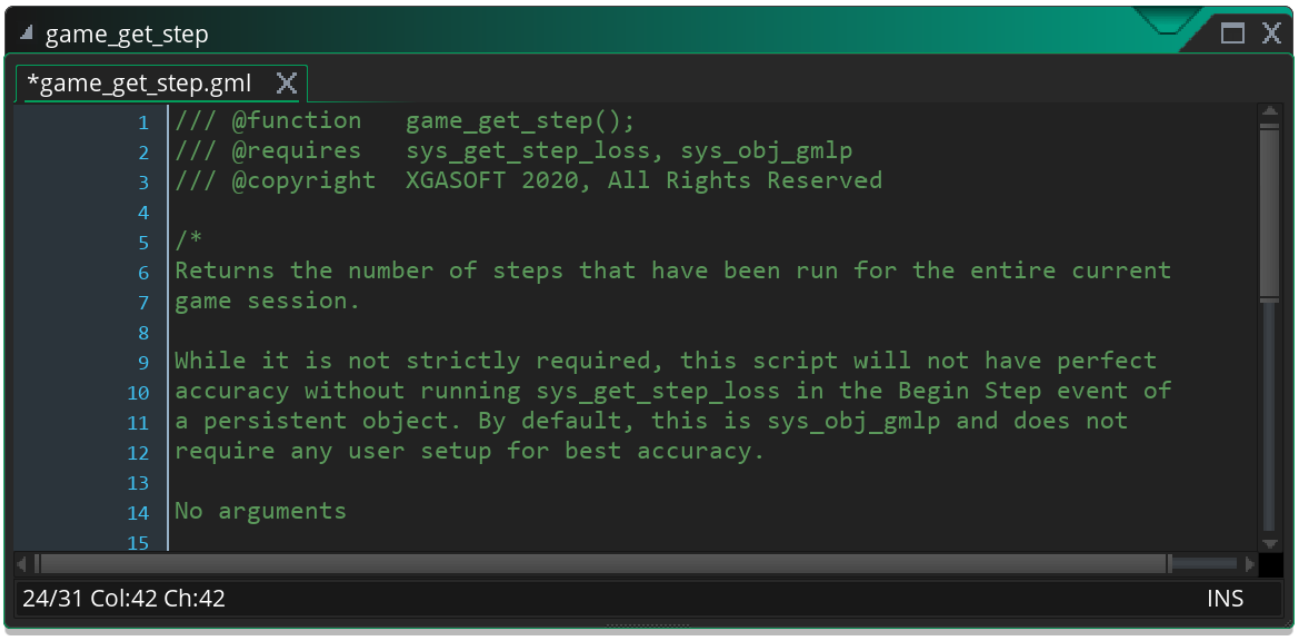


Self-Integrating Manager Object

Besides scripts, some GML+ components (such as built-in variables) rely on an object called `obj_gmlp`. However, you don't need to place this object in any of your rooms--**ever**! So long as it's in your project, it will self-create in the first available room and persist until the game ends. No further interaction is required.

Documented Dependency Requirements

But how do you know if a function requires the manager object--or other GML+ functions--to work? Just open it to find out! Every GML+ function uses JSDOC headers to define syntax and list other properties, as well as more detailed explanations and usage examples below. Look for a line beginning with `@requ ires`for a list of external dependencies.



If this line is absent, it means there are no external dependencies and the function can be used completely independently. This way, you can pick and choose only the functions your project needs!

Detailed Reference Guide

Now that you understand the GML+ folder and dependency structure, you're all set to dive straight into the full reference guide, where we'll examine each function and built-in variable in detail.

Built-in Variables & Constants

GML+ includes a number of built-in variables and constants which can be accessed in your own code. These can provide useful information or behaviors that are commonly needed, but have no implementation in vanilla GML.

All built-in variables are provided by the `obj_gmlp` object. This object must be present in the current project, but does not need to be added to any rooms to function.

Mouse Variables

Variable	Type	Description
`mouse_hspeed`	real	Stores the current horizontal mouse speed, as a value of pixels.
`mouse_vspeed`	real	Stores the current vertical mouse speed, as a value of pixels.
`mouse_speed`	real	Stores the current directional mouse speed, as a value of pixels.
`mouse_direction`	real	Stores the current mouse direction of travel, as a value of degrees.
`mouse_xstart`	real	Stores the mouse X coordinate upon starting the game. If no mouse is used, -1 will be returned. Like the instance `xstart` variable, this variable is not read-only and can be redefined if needed.
`mouse_ystart`	real	Stores the mouse Y coordinate upon starting the game. If no mouse is used, -1 will be returned. Like the instance `ystart` variable, this variable is not read-only and can be redefined if needed.
`mouse_xprevious`	real	Stores the mouse X coordinate from the *previous* step. Like the instance `xprevious` variable, this variable is not read-only and can be redefined if needed.
`mouse_yprevious`	real	Stores the mouse Y coordinate from the *previous* step. Like the instance `yprevious` variable, this variable is not read-only and can be redefined if needed.
`mouse_visible`	boolean	Shows or hides the mouse, while preserving cursor state and/or sprite. Set to `true` by default. Replaces `cr_none`, which can no longer be used while `obj_gmlp` is present.

NOTE

GameMaker Studio 2 uses two separate functions for setting different types of cursors. For system cursors, use `window_set_cursor([cursor])`. For sprite cursors, use GML+'s `window_set_cursor_sprite([sprite])`. Unlike vanilla GML, GML+ will automatically switch the cursor type when either command is used.

Time Constants & Variables

Constant	Type	Description
`frame_target`	real	Stores the amount of time a single frame *should* take to render at the current game speed (FPS), as a value of milliseconds.
`frame_time`	real	Stores the *actual* time the previous frame took to render, as a value of milliseconds.
`frame_delta`	real	Stores the difference (or delta) *between* the target and actual frame time, as a multiplier.

While the use-case for these constants and variables may not be immediately obvious, in reality they're some of the most important properties in your entire project. Much game logic occurs over time, and while it may be tempting to rely on FPS as a unit of time, this will cause a host of problems down the road. FPS-based logic can never run at another framerate without altering its real-world speed--frustrating for users on high-end systems that could run the game faster, and frustrating for low-end users that can't hit the target framerate to begin with.

A better unit of time is *the time it took to render the previous frame*, or simply `frame_time`. Frame time-based logic will always run at the same real-world speed

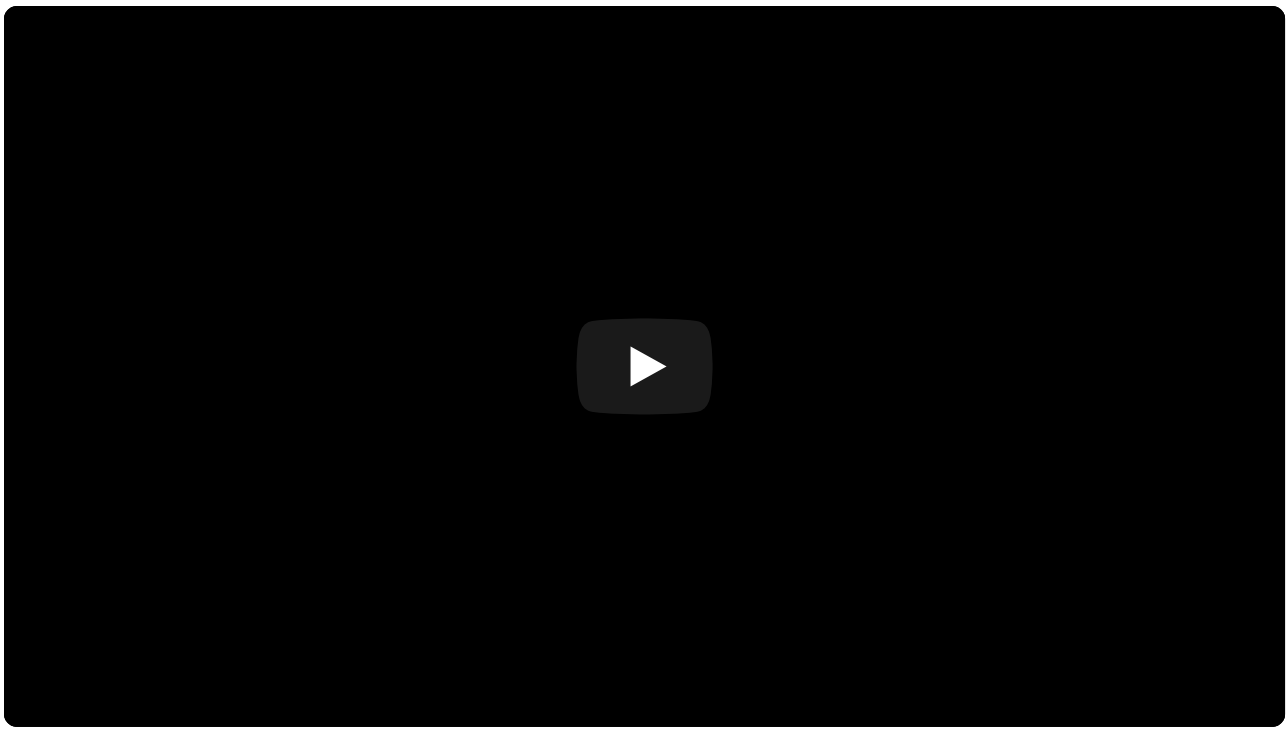
regardless of framerate. This both compensates for lag on low-end systems and allows high-end systems to run to their full potential.

You may have previously heard of this concept referred to as **delta time**. However, this is a bit of a misnomer. Strictly speaking, delta time is *not* the previous frame time, but rather *the diŸ erence betw een* the current frame time and the previous frame time. Though the term is often misused, `frame_delta` is nonetheless a very useful property to be aware of. In fact, `frame_time` and `frame_delta` are like two sides of the same coin, only forming a whole when used together.

Which available time constant or variable you should use in your logic depends on the kind of logic itself. For time-based logic (e.g. clocks, timers, etc.), use `frame_time` as a unit of milliseconds. For distance-based logic (e.g. movement, rotation, etc.), use `frame_delta` as a multiplier of pixels, degrees, and so forth.

Example

For further explanation of how to implement frame time and delta time in your projects, see the video tutorial below:



The "array_create_2d" Function

Syntax

```
array _create_2d(w idth, height, [v alu e]);
```

Argument	Type	Description
`w idth`	real	Sets the number of horizontal cells in the array
`height`	real	Sets the number of vertical cells in the array
`[v alu e]`	real/string	*Optional:* Sets a value to assign to all new cells (default 0)

Description

Creates and returns a 2D array of the given with and height, optionally assigning a custom default value to each index.

Theoretically, this can be achieved in vanilla GML by declaring a 2D arrray with a single value assigned to the maximum desired index in either axis. However, many export modules do not recognize this form of declaration, resulting in crashes. By contrast, GML+ uses a safe method to initialize every cell in order to avoid errors.

Example

```
my _array = array _create_2d(10, 5);
my _array = array _create_2d(10, 5, -1);
```

The "array_fill_2d" Function

Syntax

```
array _fill_2d(id, [v alu e]);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to modify
`[v alu e]`	real/string	*Optional:* Sets a value to assign to all new cells (default 0)

Description

Unlike `ds_grid`, 2D arrays have non-uniform columns, meaning each can have a different height. This is often problematic, and it would be preferable to have empty cells rather than no cells at all.

This script takes an existing 2D array and fills the vertical gaps, optionally assigning a custom default value to each index.

It is **highly** recommended to run this script before `array _w rite`r looping through 2D array contents if the height is not known to be uniform.

Example

```
array _fill_2d(my _array );
array _fill_2d(my _array ,1);
```

The "array_find_col" Function

Syntax

```
array _find_col(id, v alu e);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to search
`v alu e`	real/string	The value to search for

Description

Searches an array for the given value and returns the **column** index. If the value cannot not found, `-1` will be returned instead.

This script should be used with **2D** arrays only, as 1D arrays will always return `0`. To find the index of a value in a 1D array, use `array _find_row`

Example

```
//Find index of the target string
v arx x  = array_find_col(my_array, "Items held:");
v ary y  = array_find_row(my_array, "Items held:");

//Draw the string and the corresponding value held in the following
column
draw_text(25, 25, my_array [x x , y y ] + string(my_array [x x1,+y y ]));
```

The "array_find_row" Function

Syntax

```
array _find_row(id, v alu e);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to search
`v alu e`	real/string	The value to search for

Description

Searches an array for the given value and returns the **row** index. If the value cannot not found, `-1` will be returned instead.

This script can be used to search both 1D and 2D arrays.

Example

```
//Find index of the target string
v arx x  = array_find_col(my_array, "Items held:");
v ary y  = array_find_row(my_array, "Items held:");

//Draw the string and the corresponding value held in the following
column
draw_text(25, 25, my_array [x x , y y ] + string(my_array [x x1,+y y ]));
```

The "array_width" Function

Syntax

```
array _w idth(id);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to check

Description

Returns the number of horizontal cells present in a 2D array. While this script can be used on a 1D as well, it will always return `1`.

Vanilla GML array length and height functions reverse the X and Y values relative to data structures like `ds_grid`. This is confusing. By contrast, GML+ uses the same dimension for width and height as other data structures.

Example

```
v arlast_cell = array _w idth(my _array ) 1;
```

The "array_height" Function

Syntax

```
array _height(id);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to check

Description

Returns the number of vertical cells present in a 1D or 2D array. If the 2D array has non-uniform height among different columns, the tallest column count will be returned.

IMPORTANT

Just because a 2D array has a certain height does not guarantee that data exists at the highest column index of every row. Attempting to access data in shorter columns will result in an error. To avoid this behavior, it is recommended to use `array _fill_2d`.

Vanilla GML array length and height functions reverse the X and Y values relative to data structures like `ds_grid`. This is confusing. By contrast, GML+ uses the same

dimension for width and height as other data structures.

Example

```
v arlast_cell = array _height(my _array ) 1;
```

The "array_read" Function

Syntax

```
array _read(string);
```

Argument	Type	Description
`string`	string	A previously-encoded string to decode as an array

Description

Takes a string encoded with `array _write` and returns the contents as a 1D or 2D array.

This script also accepts strings encoded with `ds_grid_write` and can be used to convert `ds_grids` to arrays.

Example

```
my _array = array _read(str_array );
```

The "array_write" Function

Syntax

```
array _write(id);
```

Argument	Type	Description
`id`	array	The index of a previously-created array to encode as a string

Description

Takes a 1D or 2D array and returns the contents as an encoded string for use with `array _read`.

The resulting string can also be decoded with `ds_grid_read` and can be used to convert an array to a `ds_grid`.

Example

```
my _array  =0[ 1, 2, 3, 4];
str_array  =array _write(my _array );
```

The "window_set_cursor_sprite" Function

Syntax

```
window_set_cursor_sprite(sprite);
```

Argument	Type	Description
`sprite`	sprite	The sprite to assign as a mouse cursor

Description

Assigns a sprite to the cursor using consistent syntax with the primary `window_set_cursor` function. Note that the sprite origin point will be used as the cursor hotspot.

While it is not required, it is highly recommended to include `obj_gmlp` in the project when using this script. The `obj_gmlp` object will handle automatically switching between system and sprite cursor types, as well as provide additional mouse coordinate and visibility variables. (See Built-in Variables for a complete list.)

To disable the cursor sprite, use `cr_none`. However, if `obj_gmlp` is present, `cr_none` will be ignored, and you should use `mou se_v isible = false` to hide the cursor, or `w indow _set_cu rsor(cr_defau lt)` to restore the system cursor instead.

Example

```
w indow _set_cu rsor_sprite(spr_cu rsor);
```

The "window_get_cursor_sprite" Function

Syntax

```
window_get_cursor_sprite();
```

Argument	Type	Description
`N/A`	N/A	No parameters

Description

Returns the current sprite assigned to the cursor, if any, using consistent syntax with the primary `window_get_cursor` function. If no sprite is currently assigned, `cr_none` will be returned instead.

Example

```
if (window_get_cursor_sprite() == cr_none) {
    window_set_cursor_sprite(spr_cursor);
}
```

The "device_mouse_check_region" Function

Syntax

```
dev ice_mou se_check_regi(dev ice,x, y, rot, w idth, [height],
[halign, v align]);
```

Argument	Type	Description
`dev ice`	integer	The mouse or touch point to check, where 0 is first
`x`	real	The horizontal room coordinate for the region to check
`y`	real	The horizontal room coordinate for the region to check
`rot`	real	The rotation of the region to check, if rectangular
`w idth`	real	The width (or radius) in pixels of the region to check
`[height]`	real	*Optional:* The height in pixels of the region to check (use none for circular)
`[halign]`	constant	*Optional:* The horizontal alignment of the region to check (use none for `fa_center`)
`[v align]`	constant	*Optional:* The vertical alignment of the region to check (use none for `fa_middle`)

Description

Checks whether the mouse is currently within a certain region relative to room coordinates and returns `tru e`or `false`. If only a region width is specified, it will be interpreted as a radius and the region to check will be circular. For other shapes, a rotated rectangle can be used to cover most areas.

By default, the region to check will be aligned to the center, but this can be changed by specifying optional `halign` and `v align`values using font alignment constants such as `fa_left` and `fa_top`. Alignment **must** be input as a pair.

Example

```
//Rectangular region
if (dev ice_mou se_check_regio(0, 640, 480, 0, 512, 384)) {
    //Action
}

//Diamond region
if (dev ice_mou se_check_regio(0, 640, 480, 45, 256, 256)) {
    //Action
}

//Circular region with custom alignment
if (dev ice_mou se_check_regio(0, 256, 128, 0, 512, fa_left,
fa_top)) {
    //Action
}
```

The "device_mouse_check_region_gui" Function

Syntax

```
dev ice_mou se_check_region_gui(dev ice,x, y, rot, w idth, [height],
[halign, v align]);
```

Argument	Type	Description
`dev ice`	integer	The mouse or touch point to check, where 0 is first
`x`	real	The horizontal GUI coordinate for the region to check
`y`	real	The horizontal GUI coordinate for the region to check
`rot`	real	The rotation of the region to check, if rectangular
`w idth`	real	The width (or radius) in pixels of the region to check
`[height]`	real	*Optional:* The height in pixels of the region to check (use none for circular)
`[halign]`	constant	*Optional:* The horizontal alignment of the region to check (use none for `fa_center`)
`[v align]`	constant	*Optional:* The vertical alignment of the region to check (use none for `fa_middle`)

Description

Checks whether the mouse is currently within a certain region relative to GUI coordinates and returns `tru e`or `false`. If only a region width is specified, it will be interpreted as a radius and the region to check will be circular. For other shapes, a rotated rectangle can be used to cover most areas.

By default, the region to check will be aligned to the center, but this can be changed by specifying optional `halign` and `v align`values using font alignment constants such as `fa_left` and `fa_top`. Alignment **must** be input as a pair.

Example

```
//Rectangular region
if (dev ice_mou se_check_region_gu(0, 640, 480, 0, 512, 384)) {
    //Action
}

//Diamond region
if (dev ice_mou se_check_region_gu(0, 640, 480, 45, 256, 256)) {
    //Action
}

//Circular region with custom alignment
if (dev ice_mou se_check_region_gu(0, 256, 128, 0, 512, fa_left,
fa_top)) {
    //Action
}
```

The "ds_grid_delete_col" Function

Syntax

```
ds_grid_delete_col(id, col);
```

Argument	Type	Description
`id`	`ds_grid`	The data structure to remove a column from
`col`	integer	The index of a column to remove

Description

Removes a column from the specified `ds_grid` while preserving other data.

When complete, `ds_grid_w idth`will be reduced by 1. For this reason, a column should only be deleted when there are at least two columns in the grid, otherwise the entire grid will be destroyed.

Example

```
if (ds_grid_w idth(my _grid) >1) {
    ds_grid_delete_col(my _grid,1);
}
```

The "ds_grid_delete_row" Function

Syntax

```
ds_grid_delete_row(id, row );
```

Argument	Type	Description
`id`	`ds_grid`	The data structure to remove a row from
`row`	integer	The index of a row to remove

Description

Removes a row from the specified `ds_grid` while preserving other data.

When complete, `ds_grid_height` will be reduced by 1. For this reason, a row should only be deleted when there are at least two rows in the grid, otherwise the entire grid will be destroyed.

Example

```
if (ds_grid_height(my _grid) >1) {
    ds_grid_delete_row(my _grid,1);
}
```

The "ds_grid_insert_col" Function

Syntax

```
ds_grid_insert_col(id, col, [v alu e]);
```

Argument	Type	Description
`id`	`ds_grid`	The data structure to insert a row into
`col`	integer	The index of the new column to insert
`[v alu e]`	real/string	*Optional:* A value to assign to all new cells (default 0)

Description

Adds a new column to a `ds_grid` at the given index, shifting any columns that follow. Optionally also sets a value for empty new cells in the grid (default value is 0).

When complete, `ds_grid_w idth` will be increased by 1. For this reason, the new column index can be input as less than *or equ al to* the current value of `ds_grid_w idth`.

Example

```
ds_grid_insert_col(my _grid,ds_grid_w idth(my _grid) -2);
ds_grid_insert_col(my _grid,ds_grid_w idth(my _grid) -2, -1);
```

The "ds_grid_insert_row" Function

Syntax

```
ds_grid_insert_row(id, row , [v alu e]);
```

Argument	Type	Description
`id`	`ds_grid`	The data structure to insert a row into
`row`	integer	The index of the new row to insert
`[v alu e]`	real/string	*Optional:* A value to assign to all new cells (default 0)

Description

Adds a new row to a `ds_grid` at the given index, shifting any rows that follow. Optionally also sets a value for empty new cells in the grid (default value is 0).

When complete, `ds_grid_height` will be increased by 1. For this reason, the new row index can be input as less than *or equ al to* the current value of `ds_grid_height`.

Example

```
ds_grid_insert_row(my _grid,ds_grid_height(my _grid) -2);
ds_grid_insert_row(my _grid,ds_grid_height(my _grid) -2, -1);
```

The "ds_list_combine" Function

Syntax

```
ds_list_combine(id, sou rce, [pos]);
```

Argument	Type	Description
`id`	`ds_list`	The data structure to add new data to
`sou rce`	`ds_list`	The data structure to be added
`pos`	integer	*Optional:* The index at which to insert new data (use none for end of list)

Description

Copies the values of one `ds_list` into another `ds_list`. Unlike `ds_list_copy`, `ds_list_combine` does not clear the list of existing values.

By default, this script will insert new values at the end of the list. A different position can be optionally supplied instead, ranging from 0 to `ds_list_siz e(id)`.

Both lists must have already been created before running this script.

Example

```
my _list =ds_list_create();
my _list[|0] = "Hello, ";

my _other_list =ds_list_create();
my _other_list[|0] = "w orld!";

ds_list_combine(my _list, my _other_list);
```

The "ds_list_add_list" Function

Syntax

```
ds_list_add_list(id, sou rce);
```

Argument	Type	Description
`id`	`ds_list`	The data structure to add new data to
`sou rce`	`ds_list`	The data structure to be added

Description

Adds the **contents** of a previously-created `ds_list` to the specified `ds_list`.

Intended only for use with JSON functions. Normally, adding one data structure to another simply stores a **reference** to the data structure, therefore this function is necessary to flag the list value as a data structure itself so its contents are written to the JSON file.

As JSON data is unordered by nature, there is no need to input an index at which to insert the new list.

Example

```
my _list =ds_list_create();
my _other_list =ds_list_create();

ds_list_add_list(my _list, my _other_list);
```

The "ds_list_add_map" Function

Syntax

```
ds_list_add_map(id, sou rce);
```

Argument	Type	Description
`id`	`ds_list`	The data structure to add new data to
`sou rce`	`ds_map`	The data structure to be added

Description

Adds the **contents** of a previously-created `ds_map` to the specified `ds_list`.

Intended only for use with JSON functions. Normally, adding one data structure to another simply stores a **reference** to the data structure, therefore this function is necessary to flag the map value as a data structure itself so its contents are written to the JSON file.

As JSON data is unordered by nature, there is no need to input an index at which to insert the new map.

Example

```
my _list =ds_list_create();
my _map =ds_map_create();

ds_list_add_map(my _list, my _map);
```

The "ds_list_replace_list" Function

Syntax

```
ds_list_replace_list(id, oldlist, new list);
```

Argument	Type	Description
`id`	`ds_list`	The data structure to add new data to
`oldlist`	`ds_list`	The data structure to be replaced
`new list`	`ds_list`	The data structure to be added

Description

Replaces a `ds_list` previously added to another `ds_list` with the **contents** of a new `ds_list`.

WARNING

Because data structures are referenced by numerical values, this script may not behave as you expect! If a numerical entry in the parent `ds_list` happens to match the value of a child `ds_list`, there is no guarantee which value will be replaced!

Intended only for use with JSON functions. Normally, adding one data structure to another simply stores a **reference** to the data structure, therefore this function is necessary to flag the list value as a data structure itself so its contents are written to the JSON file.

Example

```
my _list =ds_list_create();
my _other_list =ds_list_create();
my _new _list =ds_list_create();

ds_list_add_list(my _list, my _other_list);

ds_list_replace_list(my _list, my _other_list, my _new _list);
```

The "ds_list_replace_map" Function

Syntax

```
ds_list_replace_map(id, oldlist, new list);
```

Argument	Type	Description
`id`	`ds_list`	The data structure to add new data to
`oldmap`	`ds_map`	The data structure to be replaced
`new map`	`ds_map`	The data structure to be added

Description

Replaces a `ds_map` previously added to a `ds_list` with the **contents** of a new `ds_map`.

WARNING

Because data structures are referenced by numerical values, this script may not behave as you expect! If a numerical entry in the parent `ds_list` happens to match the value of a child `ds_map`, there is no guarantee which value will be replaced!

Intended only for use with JSON functions. Normally, adding one data structure to another simply stores a **reference** to the data structure, therefore this function is necessary to flag the list value as a data structure itself so its contents are written to the JSON file.

Example

```
my _list =ds_list_create();
my _map =ds_map_create();
my _new _map =ds_map_create();

ds_list_add_map(my _list, my _map);

ds_list_replace_map(my _list, my _map, my _new _map);
```

The "make_color_hex" Function

Syntax

```
make_color_hex("#RRGGBB");
```

Argument	Type	Description
`"#RRGGBB"`	string	A three or six-character hex color code, as a string

Description

Returns an RGB color from a Hex color code.

Note that including a `#` at the beginning of the hex code is optional, and if any unacceptable input is made, the script will return `c_w hite`or the nearest usable color value to the malformed input. It is also acceptable to input only three values, in which case the secondary value will be assumed to match the first.

If you are not familiar with Hex color notation, you can choose a color using the color picker below. Click the notation to cycle between RGB, HSL, and Hex color notations.

Choose a color:
* Requires Chromium browser

Example

```
color = make_color_hex("#0066FF");
color = make_color_hex("0066FF");
color = make_color_hex("#06F");
color = make_color_hex("06F");
```

The "color_get_hex" Function

Syntax

```
color_get_hex(color);
```

Argument	Type	Description
`color`	color	An RGB color to convert to Hex

Description

Returns a Hex color code as a string from an RGB color.

Example

```
color = color_get_hex(c_red);
```

The "file_list" Function

Syntax

```
file_list(dname, attr, [recu rse]);
```

Argument	Type	Description
`dname`	string	The full path of the target directory, including drive letter
`attr`	integer/constant	Enables filtering results as directories (`fa_directory`) or files (anything else)
`[recu rse]`	boolean	*Optional:* Enables including subdirectories in scan results (disabled by default)

Description

Scans a directory and returns the contents as a `ds_list`, including relative paths (if any), filenames, and extensions.

The attribute filter and recursive options can only be used on Windows. All other platforms should set `attr` to `0` and ignore the optional `recu rse`argument.

If the attribute filter is supported and set to `fa_directory`, only directories will be returned. This is the only supported filter, and all other options will return a list of files instead.

> **CAUTION**
>
> To achieve best processing speed, files without extensions are not supported by this script.

> **IMPORTANT**
>
> By default, this script can only be used to scan directories within `w orking_directory` or elsewhere previously granted access via the `get_sav e_filename` function. On desktop platforms, this limitation can be removed by disabling the filesystem sandbox in Game Settings.

Example

```
v ardirs = file_list("C:\\my \\folder", fa_directory, tru e);
v arfiles = file_list("C:\\my \\folder", 0);
```

The "file_move" Function

Syntax

```
file_mov (fname, dest);
```

Argument	Type	Description
`fname`	string	The full path of the file to move, including name and extension
`dest`	string	The destination folder to move file to, not including filename

Description

Moves a file on the disk to a new folder, removing it from the original location. Will also return `tru e` or `false` to indicate if the operation succeeded.

> **CAUTION**
>
> On Android, files must be loaded into memory to be moved. It is not recommended to move very large files that could exceed the RAM capacity of some users' phones.

IMPORTANT

By default, this script can only be used to scan directories within `w orking_directory` or elsewhere previously granted access via the `get_sav e_filename` function. On desktop platforms, this limitation can be removed by disabling the filesystem sandbox in Game Settings.

Example

```
file_mov e(w orking_directory+ "temp.sav ", w orking_directory+
"sav es");
```

The "filename_is_dir" Function

Syntax

```
filename_is_dir(fname);
```

Argument	Type	Description
`fname`	string	The full path to check

Description

Checks if a filename appears to be a directory based on string properties and returns `tru e` or `false`. This is faster than `directory _ex ists`, and is useful for checking paths outside the sandbox.

> ⚠ CAUTION
>
> To achieve best processing speed, files without extensions are not supported by this script.

Example

```
v arfile = file_find_first("C:\\*", fa_directory);

if (filename_is_dir(file)) {
    //Directory  exists!
}
```

The "game_get_step" Function

Syntax

```
game_get_step();
```

Argument	Type	Description
`N/A`	N/A	No arguments

Description

Returns the number of Steps that have been run for the entire current game session.

While it is not strictly required, this script will not have perfect accuracy without including `obj_gmlp` in the current project. The `obj_gmlp` object will automatically track any lost steps (i.e. dropped frames) that occur as a result of system lag, window dragging, and similar events, which will then be accounted for in this script.

Example

```
if (is_ev en(game_get_step())) {
    //Action on even Steps
} else {
    //Action on odd Steps
}
```

The "game_get_time" Function

Syntax

```
game_get_time([ty pe]);
```

Argument	Type	Description
`[ty pe]`	constant	*Optional:* Sets whether to return a value in milliseconds (`gamespeed_fps`) or microseconds (`gamespeed_microseconds`) (use none for milliseconds)

Description

Returns the number of milliseconds or microseconds the entire current game session has been running. If no `[ty pe]` is specified, milliseconds will be returned by default.

Example

```
v arhou rs_play ed =floor(game_get_time(gamespeed_fps)/1000/60/60);
v arms_play ed = (game_get_time(gamespeed_microseconds)/100000);
```

The "instance_find_var" Function

Syntax

```
instance_find_var(var, n);
```

Argument	Type	Description
`var`	string	The variable name to search for, as a string
`n`	integer	The ordinal instance number to return, if multiple results are found

Description

Searches existing instances for a particular variable and returns the ID of the containing instance, or keyword `noone` if not found.

If multiple matching instances exist, you can specify which number to return with the `n` argument, where the first instance is `0`. If the input number is greater than the number of matching instances, the last result will be returned.

IMPORTANT

GameMaker instance order can vary based on many factors, so when multiple instances exist, this script may not always return the same ID each time!

Example

```
var inst = instance_find_var("my_var",0);

if (inst == noone) {
    exit;
}
```

Introduction to Angle Functions

In simple terms, trigonometry is the study of triangles. In programming, it is often used to determine the 2D coordinates of points which have been rotated a certain distance away from another point. You may have a mental image of a line being drawn from point A to point B, creating an angle. While calculating this angle is the objective we're trying to achieve, how we get there is by imagining not just a *line*, but a *triangle* instead--two flat lines following the X and Y axis like normal, while the angle is the triangle's hypotenuse.

Trigonometry demonstrates that it is possible to determine the position, orientation, and length of a triangle's hypotenuse based on its other two sides. While the formulae required are logically quite simple, actually calculating them is not. For programs that heavily rely on trigonometry, having an efficient way to perform these calculations is important. And for newcomers who may not yet be used to working with trigonometry in programming, making them easy to understand is equally so.

GML+ has many angle functions which fundamentally boil down to only three: `rot_prefetch`, `rot_point_x`, and `rot_point_y`. However, by applying the same basic principles in different ways, users may find other angle functions to be easier for their particular use-cases. In this section, we'll examine each one in detail.

The "rot_prefetch" Function

Syntax

```
rot_prefetch(deg);
```

Argument	Type	Description
`deg`	real	The angle to calculate sine and cosine, in degrees

Description

Pre-calculates the sine and cosine of an angle in degrees, which can then be used by future angle functions without calculating them again. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Other angle functions will also prefetch rotation, if supplied, in which case running this script separately is not necessary. However, prefetching rotation manually can still be quite useful in some scenarios (such as calculations spread across multiple events) or simply maintaining clean code.

Example

```
rot_prefetch(90);
x = rot_point_x(5, 10);
y = rot_point_y(5, 10);
```

The "rot_point_x" Function

Syntax

```
rot_point_x(x, y, [deg]);
```

Argument	Type	Description
`x`	real	The horizontal distance from the rotation center point
`y`	real	The vertical distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the X component of a point the given distance away rotated by the given angle in degrees. (Center point is assumed as 0.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = 128 + rot_point_x(64, 64, image_angle);
y = 128 + rot_point_y(64, 64);
```

The "rot_point_y" Function

Syntax

```
rot_point_y(x, y, [deg]);
```

Argument	Type	Description
`x`	real	The horizontal distance from the rotation center point
`y`	real	The vertical distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the Y component of a point the given distance away rotated by the given angle in degrees. (Center point is assumed as 0.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = 128 + rot_point_x(64, 64, image_angle);
y = 128 + rot_point_y(64, 64);
```

The "rot_dist_x" Function

Syntax

```
rot_dist_x(dist, [deg]);
```

Argument	Type	Description
`dist`	real	The distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the X component of a point the given distance away rotated by the given angle in degrees. (Center point is assumed as 0.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = 128 + rot_dist_x(64, image_angle);
y = 128 + rot_dist_y(64);
```

The "rot_dist_y" Function

Syntax

```
rot_dist_y(dist, [deg]);
```

Argument	Type	Description
`dist`	real	The distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the Y component of a point the given distance away rotated by the given angle in degrees. (Center point is assumed as 0.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = 128 + rot_dist_x(64, image_angle);
y = 128 + rot_dist_y(64);
```

The "rot_vec_x" Function

Syntax

```
rot_dist_x(dist, [deg]);
```

Argument	Type	Description
`x 1`	real	The horizontal center point
`y 1`	real	The vertical center point
`x 2`	real	The horizontal distance from the rotation center point
`y 2`	real	The vertical distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the X component of a point the given distance away from the given center point and rotated by the given angle in degrees. (Or in other words, the X component of the tip of a rotated line.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of

angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = rot_v ec_x(128, 128, 64, 64, image_angle);
y = rot_v ec_y(128, 128, 64, 64);
```

The "rot_vec_y" Function

Syntax

```
rot_vec_y(dist, [deg]);
```

Argument	Type	Description
`x 1`	real	The horizontal center point
`y 1`	real	The vertical center point
`x 2`	real	The horizontal distance from the rotation center point
`y 2`	real	The vertical distance from the rotation center point
`[deg]`	real	*Optional:* The angle to calculate sine and cosine, in degrees

Description

Returns the Y component of a point the given distance away from the given center point and rotated by the given angle in degrees. (Or in other words, the Y component of the tip of a rotated line.)

Supplying a rotation is optional. As calculating the sine and cosine of angles is costly to performance, these values are stored in memory for use with further instances of

angle functions based on the same rotation. If no rotation is supplied, the previous angle's sine and cosine will be used. This is highly useful for improving performance when calculating multiple points based on the same rotation.

Example

```
x = rot_v ec_x(128, 128, 64, 64, image_angle);
y = rot_v ec_y(128, 128, 64, 64);
```

The "approx" Function

Syntax

```
approx(v alu e, min, [max ]);
```

Argument	Type	Description
`v alu e`	real	The numerical value to check
`min`	real	The minimum closeness, or alternatively, minimum value to accept
`[max ]`	real	*Optional:* The maximum value to accept (use none for +/- `min`)

Description

Checks if a value is between two numbers and returns `tru e` or `false` .

By default, the input value will be tested plus or minus the `min` value, but an explicit `max` value can also be supplied to set the exact range.

Example

```
if (approx(enemy x - x, 128)) {
    //Enemy  is near play er on left or right
}
```

The "interp" Function

Syntax

```
interp(a, b, amou nt, ease, [bx 1, by 1, bx 2, by 2]);
```

Argument	Type	Description
`a`	real	The starting value to interpolate from
`b`	real	The target value to interpolate to
`amou nt`	real	The amount, or percentage to interpolate between values (variable recommended)
`ease`	integer/macro	Sets the easing method for the interpolation (see options below)
`[bx 1]`	real	*Optional:* X percentage for left control point of a cubic bezier curve (0-1)
`[by 1]`	real	*Optional:* Y percentage for left control point of a cubic bezier curve (0-1)
`[bx 2]`	real	*Optional:* X percentage for right control point of a cubic bezier curve (0-1)
`[by 2]`	real	*Optional:* Y percentage for right control point of a cubic bezier curve (0-1)

Description

Returns a value interpolated between the two input values with optional easing methods to create a smooth start and/or end to animations.

The first input value should equal the original state of the value and the second input the target state of the value. For example, to move an object from `x = 0` to `x = 50`, `0` and `50` would be the two input values here.

The third input value can be thought of as a percentage of completion. Using the same example, an input amount of `0.5` would return `x = 25`.

In order to create animations with this script, the interpolation amount must be input as a variable which is incremented externally.

The fourth and final value is an integer specifying the easing method used during interpolation. True or false can be used here to specify basic in/out interpolation or linear interpolation (i.e. no easing), but in addition to these basic modes there are 30 different easing techniques, featured below. Easing techniques are ordered from shallowest to deepest curve, with a few special techniques added on at the end as well.

For memorability, it is recommended to use an easing macro from the list below in place of an integer value:

Easing Macro	Value		Easing Macro	Value
ease_none	-4		ease_ex po	16
ease_sin	1		ease_ex po_in	17
ease_sin_in	2		ease_ex po_ou t	18
ease_sin_ou t	3		ease_circ	19
ease_qu ad	4		ease_circ_in	20
ease_qu ad_in	5		ease_circ_ou t	21
ease_qu ad_ou t	6		ease_ru bber	22
ease_cu bic	7		ease_ru bber_in	23
ease_cu bic_in	8		ease_ru bber_ou t	24
ease_cu bic_ou t	9		ease_elastic	25
ease_qu art	10		ease_elastic_in	26
ease_qu art_in	11		ease_elastic_ou t	27
ease_qu art_ou t	12		ease_bou nce	28
ease_qu int	13		ease_bou nce_in	29
ease_qu int_in	14		ease_bou nce_ou t	30
ease_qu int_ou t	15		ease_bez ier	31

If the bezier ease mode is selected, four more arguments can be supplied to act as control points for a custom interpolation curve. These values range from 0-1, but Y values can be less or greater to create a rubber-banding effect. See https://cubic-bezier.com/ for an interactive visual example.

Example

```
du ration =5;
time += delta_time/1000000;

x = interp(0, 50, time/du ration, ease_qu art);
y = interp(0, 50, time/du ration, ease_bez ier0.66, -0.33, 0.33,
1.33);
```

The "is_even" Function

Syntax

```
is_ev en(n);
```

Argument	Type	Description
`n`	real	A number to check parity of

Description

Returns `tru e` if a given number is even, and `false` if odd. Invalid inputs will be returned as even.

Example

```
if (is_ev en(v ar_nu m)) {
    show _messag("I'm ev en!");
} else {
    show _messag("I'm odd!");
}
```

The "is_odd" Function

Syntax

```
is_odd(n);
```

Argument	Type	Description
`n`	real	A number to check parity of

Description

Returns `tru e` if a given number is odd, and `false` if even. Invalid inputs will be returned as even.

Example

```
if (is_odd(v ar_nu m)) {
    show _messag("I'm odd!");
} else {
    show _messag("I'm ev en!");
}
```

The "round_to" Function

Syntax

```
rou nd_to(v alu e, mu ltiple);
```

Argument	Type	Description
`v alu e`	real	The value to round
`mu ltiple`	real	The number to round to, as a multiple

Description

Rounds to a multiple of the specified number (rather than to the nearest whole) and returns the result. Unlike normal rounding, rounding to fractional values is supported.

Note that this script uses "banker's rounding", meaning if a value is exactly half the multiplier, it will round to the nearest **even** number.

Also note that the multiplier should **always** be positive. The value to round can be either positive or negative.

Example

```
score = rou nd_to(score, 10);
```

The "timer" Function

Syntax

```
timer(id, [du ration]);
```

Argument	Type	Description
`id`	string	Sets a unique timer ID, as a string
`[du ration]`	real	*Optional:* Sets the duration of time to countdown, in seconds (use none to create timer only)

Description

Sets and/or counts down a timer and returns `false` until the time has expired, after which it will return `tru e`. (To return the actual time value, see `timer_get`.)

The timer ID should be a unique string value. Timers and their IDs are local to the running instance, so multiple timers can use the same ID in different instances. However, the same ID cannot be reused within a single instance. Otherwise, there is no limit on the quantity of timers that can exist at once.

Timer duration is measured in seconds. This value is automatically adapted to framerate and delta time.

Example

```
if (timer("t_alarm", 3)) {
    //Action after 3 seconds
}
```

The "timer_set" Function

Syntax

```
timer_set(id, du ration, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to modify, as a string (or keyword `all` for all local timers)
`du ration`	real	Sets the duration of time to countdown, in seconds
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to modify (use none for `self`)

Description

Overrides the current time in the specified timer, restarting the countdown process. If the timer does not exist, it will be created, but not countdown.

Note that if this script is run in an event that is executed every frame (e.g. Step), the timer will be unable to countdown! If this is required, use an 'if' statement to only set the timer under certain conditions.

Example

```
timer("t_alarm", 5);
timer("t_other", 5, my _other_inst);
```

The "timer_get" Function

Syntax

```
timer_get(id, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to check, as a string
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to check (use none for `self`)

Description

Returns the time remaining for the timer running in the current or specified instance, as a value of seconds. If the instance or timer does not exist, `-1` will be returned instead.

Example

```
v arinst_streetlight = instance_find(obj_streetlight, 0);
v arrace_started = (timer_get("t_streetlight", inst_streetlight) ==
0);
```

The "timer_set_speed" Function

Syntax

```
timer_set_speed(id, speed, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to modify, as a string (or keyword `all` for all local timers)
`du ration`	real	Sets the speed multiplier of time countdown, where 1 is default
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to modify (use none for `self`)

Description

Sets the speed multiplier for the specified timer, increasing or decreasing the countdown rate. The default value of 1 equals real time.

Example

```
timer_set_speed("t_alarm", 0.5);
timer_set_speed(all, 1, my _other_inst);
```

The "timer_get_speed" Function

Syntax

```
timer_get_speed(id, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to check, as a string
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to check (use none for `self`)

Description

Returns the speed multiplier for the timer running in the current or specified instance, where a value of 1 is real time. If the instance or timer does not exist, 0 will be returned instead.

Example

```
v army _timer_speed =timer_get_speed("my _timer");

my _timer_speed += (1 - my _timer_speed)*0.25;

timer_set_speed("my _timer", my _timer_speed);
```

The "timer_set_pause" Function

Syntax

```
timer_set_pau se(id, enable, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to modify, as a string (or keyword `all` for all local timers)
`enable`	boolean	Enables, disables, or toggles the pause state
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to modify (use none for `self`)

Description

Pauses or unpauses the specified timer. Can also toggle pause state if `other` is supplied instead of `tru e` or `false`.

Example

```
timer_set_pau se("t_alarm", other);
timer_set_pau se(all, tru e, my _other_inst);
```

The "timer_get_pause" Function

Syntax

```
timer_get_pau se(id, [instance]);
```

Argument	Type	Description
`id`	string	The unique timer ID to check, as a string
`[instance]`	instance	*Optional:* Sets the object instance containing the timer to check (use none for `self`)

Description

Returns the pause state for the timer running in the current or specified instance. If the instance or timer does not exist, `tru e` will be returned, as the timer is not running.

Example

```
if (timer_get_pau se("my _timer")) {
    timer_set("my _timer", 5);
}
```

The "wait" Function

Syntax

```
wait(duration, [offset]);
```

Argument	Type	Description
`duration`	real	Sets the duration of time to wait, in seconds
`[offset]`	real	*Optional:* Sets the amount of time to offset the timer

Description

Returns `false` for a specified interval, as a value of seconds, after which `true` will be returned for **one frame**. Repeats endlessly.

Note that this script's starting time is based on *instance creation time*, and will always return `true` at the same time for every instance of any object created in the same Step. Sometimes this synchronization is not desirable, in which case an optional offset time can also be supplied. Unlike the main time interval, the offset value can be either positive or negative. For example, to base starting time on global session time, use `-game_get_time()` (must be a variable declared in an event that is not run every Step).

Example

```
//STEP EVENT
if (x != x_prev_iou) or (y != y_prev_iou) {
    if (wait(1)) {
        stamina--;
    }
} else {
    if (wait(2)) {
        stamina++;
    }
}

//LEFT MOUSE PRESSED EVENT
click_time = -game_get_time();

//LEFT MOUSE DOWN EVENT
if (wait(0.15, click_time)) {
    instance_create_layer(x, y, layer, obj_bullet);
}
```

The "sprite_get_index" Function

Syntax

```
sprite_get_index(sprite, [offset]);
```

Argument	Type	Description
`sprite`	sprite	The sprite to retrieve image index of
`[offset]`	real	*Optional:* Sets the number of frames to offset the sprite index

Description

Every instance has a built-in `image_index` variable which tracks the animation frame for the sprite assigned to `sprite_index`, adjusted for `image_speed`. But many objects draw multiple sprites, each of which may have a different speed. This script returns the `image_index` value for any sprite, factoring in both sprite speed and delta time.

Note that this script is based on *global session time*, and will always return the same index at the same time for every instance of the sprite. Sometimes this synchronization is not desirable, in which case an optional offset time can also be supplied, as a value of frames. For example, to base starting time on instance creation

time, use `-sprite_get_index ()`(must be a variable declared in an event that is not run every Step).

Example

```
//CREATE EVENT
v aroffset = -sprite_get_index(my _sprite);

//DRAW EVENT
draw _sprite(my _sprite,sprite_get_index(my _sprite, offset),x + 32,
y - 32);
```

The "sprite_get_speed_fps" Function

Syntax

```
sprite_get_speed_fps(sprite);
```

Argument	Type	Description
`sprite`	sprite	The sprite to retrieve speed of

Description

Returns the target sprite speed as defined in the sprite editor, forcing the results to be interpreted as **frames per-second**.

Example

```
v arspeed_fps = sprite_get_speed_fps(my _sprite);
```

The "sprite_get_speed_real" Function

Syntax

```
sprite_get_speed_real(sprite);
```

Argument	Type	Description
`sprite`	sprite	The sprite to retrieve speed of

Description

Returns the target sprite speed as defined in the sprite editor, forcing the results to be interpreted as **frames per-game frame**.

Example

```
v arspeed_real = sprite_get_speed_real(my _sprite);
```

Special Thanks

Patreon credits

This product is made possible by the generous support of XGASOFT patrons on Patreon. Every contribution counts, no matter how big or small. To all fans and patrons around the globe, thanks for being a part of XGASOFT's story!

Very special thanks goes out to:

Patreon 'Enthusiasts'

Marvin Mrzyglod

Patreon 'Developers'

AshleeVocals

AutumnInAprilArt

Daniel Sato

Darktoz

Dirty Sock Games

Josef Scott

Meyaoi Games

Patreon 'Gamers'

Kampmichi (Forgers of Novelty)

All Other Patreon Supporters

Adam Miller (Actawesome)

Cosmopath

D Luecke

Alex Lepinay

Tarquinn J Goodwin

Creative Credits

XGASOFT is also privileged to work with other creators from around the world, in some cases on the very developer tools used to make XGASOFT products possible.

Special credit goes out to the following talents for their contributions:

VNgen Demo Voiceover

Kanen *(as Miki and Mei)*

End-User License Agreement ("EULA")

NOTE

Last updated: 12/16/2019

We know that reading EULAs isn't very exciting, but this is important. Please take your time to review and ensure you understand the terms of this document before proceeding to use XGASOFT products in your own work.

If you have any questions or concerns about the terms outlined in this document, please feel free to contact us at contact@xgasoft.com or by visiting our Contact & Support page.

License Agreement

This License Agreement (the "Agreement") is entered into by and between XGASOFT (the "Licensor"), and you (the "Licensee"). This agreement is legally binding, and becomes effective when you purchase and/or download a free product from XGASOFT or authorized third-party distributors. If you do not agree to the terms of this Agreement, do not purchase, download, or otherwise use XGASOFT products.

In order to accept this Agreement, you must be at least eighteen (18) years of age or whatever age is of legal majority in your country. Otherwise, you must obtain your parent's or legal guardian's approval and acceptance of this Agreement in your stead. XGASOFT accepts no liability for your failure to meet this requirement.

XGASOFT delivers content through authorized third-party distributors, each of which may require its own separate End-User License Agreement ("EULA"). XGASOFT accepts no liability for the terms of any third-party agreements, nor for your failure to meet them.

Standard Lifetime License

This is a license, not a sale. XGASOFT retains ownership of all content (including but not limited to any copyright, trademarks, brand names, logos, software, images, animations, graphics, video, audio, music, text, and tutorials) comprising digital products and services offered by XGASOFT (the "Property"). All rights not expressly granted are reserved by XGASOFT.

Subject to your acceptance of the terms of this Agreement, XGASOFT grants you a worldwide, revocable, non-exclusive, non-transferable, and **perpetual** license to download, embed, and modify for your own purposes XGASOFT Property solely for incorporation with electronic applications and other interactive media, including both commercial and non-commercial works, wherever substantial value has been added by you.

Any source code included as part of XGASOFT Property must be compiled prior to redistribution as an incorporated work, whether for commercial or non-commercial purposes.

Patreon Limited License

When you register as a recurring financial supporter of XGASOFT through Patreon (Patreon, Inc.), XGASOFT may provide free access to XGASOFT Property as a reward, subject to the terms of each contribution tier. This is a privilege, not a right.

XGASOFT retains ownership of all content (including but not limited to any copyright, trademarks, brand names, logos, software, images, animations, graphics, video, audio, music, text, and tutorials) comprising digital products and services offered by XGASOFT (the "Property"). All rights not expressly granted are reserved by XGASOFT.

Subject to your acceptance of the terms of this Agreement, XGASOFT grants you a worldwide, revocable, non-exclusive, non-transferable, and **temporary** license to download, embed, and modify for your own purposes XGASOFT Property solely for incorporation with electronic applications and other interactive media, including both commercial and non-commercial works, wherever substantial value has been added by you.

Any source code included as part of XGASOFT Property must be compiled prior to redistribution as an incorporated work, whether for commercial or non-commercial purposes.

This license shall remain effective for the duration of your subscription to XGASOFT through Patreon. In the event that you cancel or reduce your contribution to a lower tier not qualifying for free access to XGASOFT Property, this license will be considered revoked and void for any and all public commercial and non-commercial activities. In order to continue using XGASOFT Property publicly, you must purchase a standard lifetime license.

This limitation shall not be applied retroactively, so that any existing, complete, and publicly available commercial and non-commercial properties using XGASOFT Property will not be considered in violation of this agreement. Furthermore, this limitation shall not apply in the event that XGASOFT suspends, revokes, or disables the contribution of financial support to XGASOFT through Patreon. In such case as

contributions are limited or prohibited by XGASOFT (and not the Licensee), the terms of the Standard Lifetime License shall apply to any and all XGASOFT Property granted as rewards for recurring financial support prior to the date of suspension.

Single-User

This Agreement grants one (1) user an applicable license to use XGASOFT Property on unlimited devices. This license may not be transferred, shared with, or sold to other users.

However, you, the Licensee, may use XGASOFT Property along with a team or company of collaborators wherever substantial value has been added by you.

This limitation does not extend a license to other users. For any works unrelated to you, collaborators must purchase separate licenses.

Modifications

In accordance with the terms of this Agreement, you may freely modify, or alter the functionality of XGASOFT Property exclusively for your own use.

Modifying the Property will not terminate your license, however XGASOFT cannot guarantee the quality and functionality of modified versions of the Property, nor its compatibility with other products.

XGASOFT accepts no liability for any loss or damage incurred by the modified Property, and reserves the right to refuse technical support for the modified Property.

Modifications made to XGASOFT Property in no way represent a change of ownership of the Property.

You may not reverse-engineer XGASOFT Property for the purpose of commercial exploitation which may be in competition with XGASOFT.

Mutability

License fees are determined for each product and service on a case-by-case basis, and XGASOFT reserves the right to change fees on the Property with or without prior notice.

XGASOFT reserves the right to modify, suspend, or terminate this Agreement, the Property, or any service to which it connects with or without prior notice and without liability to you, the Licensee.

Liability

By using XGASOFT Property, you agree to indemnify and hold harmless XGASOFT, its employees, and agents from and against any and all claims (including third party claims), demands, actions, lawsuits, expenses (including attorney's fees) and damages (including indirect or consequential loss) resulting in any way from your use or reliance on XGASOFT Property, any breach of terms of this Agreement, or any other act of your own.

This limitation will survive and apply even in the event of termination of this Agreement.

Governing Law

This Agreement shall be governed by and interpreted according to the laws of the United States of America and the State of Kansas.

If any provision of this Agreement is held to be unenforceable or invalid, such provision will be changed and interpreted to accomplish the objectives of such provision to the greatest extent possible under applicable law, and the remaining provisions will continue in full force and effect.

Conclusion

This document contains the whole agreement between XGASOFT and you, the Licensee, relating to the Property and licenses thereof and supersedes all prior Agreements, arrangements and understandings between both parties regarding XGASOFT Property and licenses.