

docs.xgasoft.com



Table of contents:

- [Welcome to GML+ - Essential Extensions for the Way You Work](#)
- [Unified. Simplified. Amplified.](#)
- [It features...](#)
- [Version History](#)
 - [1.1.4](#)
 - [1.1.3](#)
 - [1.1.2](#)
 - [1.1.0](#)
 - [1.0.0](#)
- [Compatibility Notes](#)
 - [1.1.4](#)
 - [1.1.3](#)
 - [1.1.2](#)
 - [1.1.0](#)
- [GML+ Reference Guide](#)
 - [Vanilla-Compliant Folder Structure](#)
 - [Self-Integrating Manager Object](#)
 - [Documented Dependency Requirements](#)
 - [Detailed Reference Guide](#)
- [Built-in Variables & Constants](#)
 - [Mouse Variables](#)
 - [Time Constants & Variables](#)
 - [Deprecated Variables](#)
- [The "instance_find_var" Function](#)
 - [Syntax](#)
 - [Description](#)
 - [Example](#)
- [The "instance_link" Function](#)
 - [Syntax](#)
 - [Description](#)
 - [Example](#)
- [The "camera_get_view" Function](#)

- Syntax
- Description
- Example
- The "window_set_cursor_sprite" Function
 - Syntax
 - Description
 - Example
- The "window_get_cursor_sprite" Function
 - Syntax
 - Description
 - Example
- The "ds_grid_empty" Function
 - Syntax
 - Description
 - Example
- The "ds_grid_delete_col" Function
 - Syntax
 - Description
 - Example
- The "ds_grid_delete_row" Function
 - Syntax
 - Description
 - Example
- The "ds_grid_insert_col" Function
 - Syntax
 - Description
 - Example
- The "ds_grid_insert_row" Function
 - Syntax
 - Description
 - Example
- The "ds_list_combine" Function
 - Syntax
 - Description
 - Example

- The "ds_list_add_list" Function
 - Syntax
 - Description
 - Example
- The "ds_list_add_map" Function
 - Syntax
 - Description
 - Example
- The "ds_list_replace_list" Function
 - Syntax
 - Description
 - Example
- The "ds_list_replace_map" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_create" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_copy" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_destroy" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_exists" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_equals" Function
 - Syntax
 - Description

- Example
- The "ds_struct_find_first" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_find_last" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_find_next" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_find_previous" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_find_index" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_find_value" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_set" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_read" Function
 - Syntax
 - Description
 - Example
- The "ds_struct_write" Function
 - Syntax

- Description
- Example
- The "make_color_hex" Function
 - Syntax
 - Description
 - Example
- The "color_get_hex" Function
 - Syntax
 - Description
 - Example
- The "sprite_get_index" Function
 - Syntax
 - Description
 - Example
- The "sprite_get_speed_fps" Function
 - Syntax
 - Description
 - Example
- The "sprite_get_speed_real" Function
 - Syntax
 - Description
 - Example
- The "draw_get_surface" Function
 - Syntax
 - Description
 - Example
- The "surface_read" Function
 - Syntax
 - Description
 - Example
- The "surface_write" Function
 - Syntax
 - Description
 - Example
- The "file_list" Function

- Syntax
- Description
- Example
- The "file_move" Function
 - Syntax
 - Description
 - Example
- The "filename_is_dir" Function
 - Syntax
 - Description
 - Example
- The "device_mouse_check_region" Function
 - Syntax
 - Description
 - Example
- The "device_mouse_check_region_gui" Function
 - Syntax
 - Description
 - Example
- The "game_get_step" Function
 - Syntax
 - Description
 - Example
- The "game_get_time" Function
 - Syntax
 - Description
 - Example
- The "foreach" Statement
 - Syntax
 - Description
 - Example
- Introduction to Angle Functions
- The "angle_reflect" Function
 - Syntax
 - Description

- Example
- The "angle_refract" Function
 - Syntax
 - Description
 - Example
- The "rot_prefetch" Function
 - Syntax
 - Description
 - Example
- The "rot_dist_x" Function
 - Syntax
 - Description
 - Example
- The "rot_dist_y" Function
 - Syntax
 - Description
 - Example
- The "rot_point_x" Function
 - Syntax
 - Description
 - Example
- The "rot_point_y" Function
 - Syntax
 - Description
 - Example
- The "rot_vec_x" Function
 - Syntax
 - Description
 - Example
- The "rot_vec_y" Function
 - Syntax
 - Description
 - Example
- The "approx" Function
 - Syntax

- Description
- Example
- The "emod" Function
 - Syntax
 - Description
 - Example
- The "interp" Function
 - Syntax
 - Description
 - Example
- The "is_even" Function
 - Syntax
 - Description
 - Example
- The "is_odd" Function
 - Syntax
 - Description
 - Example
- The "round_to" Function
 - Syntax
 - Description
 - Example
- The "timer" Function
 - Syntax
 - Description
 - Example
- The "timer_set_time" Function
 - Syntax
 - Description
 - Example
- The "timer_get_time" Function
 - Syntax
 - Description
 - Example
- The "timer_set_pause" Function

- Syntax
- Description
- Example
- The "timer_get_pause" Function
 - Syntax
 - Description
 - Example
- The "timer_set_speed" Function
 - Syntax
 - Description
 - Example
- The "timer_get_speed" Function
 - Syntax
 - Description
 - Example
- The "wait" Function
 - Syntax
 - Description
 - Example
- The "collision_line_meeting" Function
 - Syntax
 - Description
 - Example
- The "string_explode" Function
 - Syntax
 - Description
 - Example
- The "string_implode" Function
 - Syntax
 - Description
 - Example
- The "string_lower_all" Function
 - Syntax
 - Description
 - Example

- The "string_lower_first" Function
 - Syntax
 - Description
 - Example
- The "string_lower_words" Function
 - Syntax
 - Description
 - Example
- The "string_upper_all" Function
 - Syntax
 - Description
 - Example
- The "string_upper_first" Function
 - Syntax
 - Description
 - Example
- The "string_upper_words" Function
 - Syntax
 - Description
 - Example
- The "string_trim" Function
 - Syntax
 - Description
 - Example
- The "string_trim_left" Function
 - Syntax
 - Description
 - Example
- The "string_trim_right" Function
 - Syntax
 - Description
 - Example
- The "array_create_ext" Function
 - Syntax
 - Description

- Example
- The "array_clear" Function
 - Syntax
 - Description
 - Example
- The "array_depth" Function
 - Syntax
 - Description
 - Example
- The "array_fill" Function
 - Syntax
 - Description
 - Example
- The "array_find_dim" Function
 - Syntax
 - Description
 - Example
- The "array_find_index" Function
 - Syntax
 - Description
 - Example
- The "array_shuffle" Function
 - Syntax
 - Description
 - Example
- The "array_read" Function
 - Syntax
 - Description
 - Example
- The "array_write" Function
 - Syntax
 - Description
 - Example
- The "is_empty" Function
 - Syntax

- Description
- Example
- Patreon credits
 - Patreon 'Enthusiasts'
 - Patreon 'Developers'
 - Patreon 'Gamers'
 - All Other Patreon Supporters
- Creative Credits
 - VNgen Demo Voiceover
- End-User License Agreement ("EULA")
 - License Agreement
 - Standard Lifetime License
 - Patreon Limited License
 - Single-User
 - Modifications
 - Mutability
 - Liability
 - Governing Law
 - Conclusion



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!**
- **Additional language features**, such as:
 - **Universal data functions** like `foreach` and `is_empty`
- **Built-in variables and macros**, such as:
 - **Extended mouse variables**, unifying behavior with instances
 - **Extended instance properties**, creating new possibilities for game design

- 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** for trigonometry (robust `lengthdir`), reflections, and more
 - **Interpolation with easing** (robust `lerp`), including custom bezier and user-generated curves
 - **Filesystem functions** (robust `file_find_*`)
 - **Timer functions** (robust `alarm`)
 - **String manipulation functions**
 - **Recursive struct functions**
 - **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

1.14

- Added `collision_line_meeting` for identifying exact coordinates of a collision intersection
- Added `ds_struct_equals` for recursively comparing contents of structs
- Added struct support to `is_empty`
- Added `emod` for calculating remainder with Euclidean division (always positive result)
- Fixed `angle_refract` clipping mirror angle below 0 degrees
- Improved `instance_link` to no longer depend on `obj_server_gmlp`
- Improved accuracy of `game_get_step` when `obj_server_gmlp` is present
- Optimized mouse and frame-time constants
- Optimized `game_get_time`
- Deprecated universal `image_angle_previous`, `image_x/yscale_previous` for performance reasons, as they are no longer needed for `instance_link` (see Compatibility Notes)

1.1.3

- Added `ds_struct_copy` for "deep cloning" structs and all contents (including other structs/arrays)
- Added `ds_struct_find_first`, `ds_struct_find_next`, `ds_struct_find_previous`, `ds_struct_find_last`

- Added support for objects to `foreach` (loops all instances of an object)
- Improved `wait` timer accuracy when application is running below performance target
- Fixed `string_explode` returning an empty first value if the delimiter doesn't exist in the string
 - Strings with no delimiters will now return the string as an array of one value as expected
- Fixed `string_implode` returning a string with a delimiter even if only one array value exists
 - Strings will now only include delimiters when two or more array values exist as expected

1.1.2

- Improved `foreach` syntax and behavior

1.1.0

- GameMaker Studio 2.3.1 support!
 - Completely reorganized code structure, updated to new standards
- Added `foreach` function to iterate the contents of a wide variety of data types, including numbers, strings, arrays, and data structures
- Added `is_empty` function to check the contents of a wide variety of data types, including numbers, strings, arrays, data structures, surfaces, and buffers
- Added `surface_read` and `surface_write` functions for handling surfaces as strings
- Added `draw_get_surface` to restore broken surfaces from cached memory
- Added `camera_get_view` to complement built-in `view_get_camera` function
- Added `ds_struct` functions for interacting with structs like other data structures
 - Supports struct hierarchies, unlike built-in functions!

- Added `ds_grid_empty` function, bringing parity between grids and other data structures
- Added `instance_link` function for grouping objects by position, rotation, and scale
- Added `image_angle_previous`, `image_xscale_previous`, and `image_yscale_previous` built-in variables to all instances via `obj_gmlp`
- Added `angle_reflect` and `angle_refract` to Angle Functions
 - Includes new demo room!
- Added support for animation curve assets to `interp`. Make your own custom curves using the built-in visual editor in GameMaker Studio 2.3!
- Added `string_explode`, `string_implode`, and `string_trim` functions for converting data between strings and arrays
- Added `string_upper_*` and `string_lower_*` functions for manipulating string case on an individual, per-word, or whole-string basis
 - `string_upper_all` and `string_lower_all` are 2x faster than equivalent built-in GML!
- Multiple changes to array functions:
 - Added `array_clear` and `array_shuffle` for greater parity with data structure functions
 - Replaced `array_create_2d` with `array_create_ext`, now supporting any number of dimensions!
 - Replaced `array_fill_2d` with `array_fill`, now supporting any number of dimensions!
 - Replaced `array_width` and `array_height` with `array_depth` to complement the new built-in `array_length` function
 - Replaced `array_find_col` and `array_find_row` with `array_find_index` and `array_find_dim` for better compliance with updated GML behaviors
 - Removed `array_sort`, `array_delete_*`, and `array_insert_*` functions, as they have been replaced by official functions
- Fixed `wait` returning `true` on the first frame of the first cycle
 - Now will wait the specified duration once before first returning `true`

1.0.0

- Initial release



Compatibility Notes

Some updates include certain changes which require existing projects to be modified to retain compatibility with updated versions. This section documents those changes as well as the remedies to any incompatibilities they create.

1.14

- New dependency: `angle_refract` now depends on `emod` for accurate modulo calculation
 - Projects currently using `angle_refract` must also import `emod` when updating.
- Obsolete dependency: `instance_link` no longer depends on `obj_server_gmlp`
 - Projects which included `obj_server_gmlp` only for `instance_link` can now safely remove it.
- Deprecated: universal `image_angle_previous`, `image_x/yscale_previous`
 - These were previously included for full functionality of `instance_link`, which no longer requires them.
 - Universally updating these variables has a measurable performance cost whether or not they are used.
 - This feature is now disabled by default, but can be re-enabled by modifying `obj_server_gmlp`:
 - In the object **Create Event**, under "Configure GML+", change `expanded_image_previous` to `true`.
 - No other modifications to `obj_server_gmlp` should be made.

1.1.3

- Added support for objects to `foreach`
 - Due to the way GameMaker handles pointers, this may potentially disrupt integer `foreach` loops if the integer happens to match an object asset index.
 - If existing code with an integer `foreach` loop now attempts to loop through instances of an object, use a `repeat` loop instead.
- Fixed `string_explode` returning an empty first value if the delimiter doesn't exist in the string
 - Any code that previously handled empty first values must be updated to assume an array length of 1 if no delimiter was found in the string.
- Fixed `string_implode` returning a string with a delimiter even if only one array value exists
 - Any code that previously handled single-value strings with delimiters must be updated to assume no delimiter is present.
- Renamed `obj_gm1p` to `obj_server_gm1p` for consistency with other XGASOFT middleware
 - Any existing references to the old name must be updated to match the new nomenclature
 - (Recommended: Use global search & replace)

1.1.2

- Improved `foreach` syntax and behavior
 - Parentheses now close *before* `call` keyword rather than after
 - Existing instances of this function must be updated to match the new syntax
 - (Recommended: Use regex global search & replace in external code editor)

1.1.0

- Changed `timer_get` and `timer_set` to `timer_get_time` and `timer_set_time` for clarity and consistency with other get and set functions.
 - Existing instances of these functions must be renamed to match the updated syntax
 - (Recommended: Use global search & replace)
- Replaced `array_create_2d` with `array_create_ext`, now supporting arbitrary dimensions
 - Existing instances of the old function must be renamed to match the new function.
 - Instances providing two dimensions and a default value require no further changes.
 - Setting a default value is now required, as additional arguments are treated as new dimensions preferentially. Instances which previously set no default value must be updated to include one.
 - (Recommended: Use global search & replace)
- Replaced `array_fill_2d` with `array_fill`, now supporting arbitrary dimensions
 - Existing instances of the old function must be renamed to match the new function.
 - (Recommended: Use global search & replace)
- Replaced `array_width` and `array_height` with `array_depth` to complement the new built-in `array_length` function
 - `array_width` generally maps to `array_length` and `array_height` generally maps to `array_depth`. However, behaviors may not be identical in all cases due to multidimensional arrays now acting as arrays within arrays. Evaluate existing code and update as needed.
 - (Recommended: Use global search & replace)
- Replaced `array_find_col` and `array_find_row` with `array_find_index` and `array_find_dim`
 - `array_find_col` generally maps to `array_find_index` and `array_find_row` generally maps to `array_find_dim`. However, behaviors may not be identical in all cases due to multidimensional arrays now acting as arrays within arrays. Evaluate existing code and update as needed.

- (Recommended: Use global search & replace)

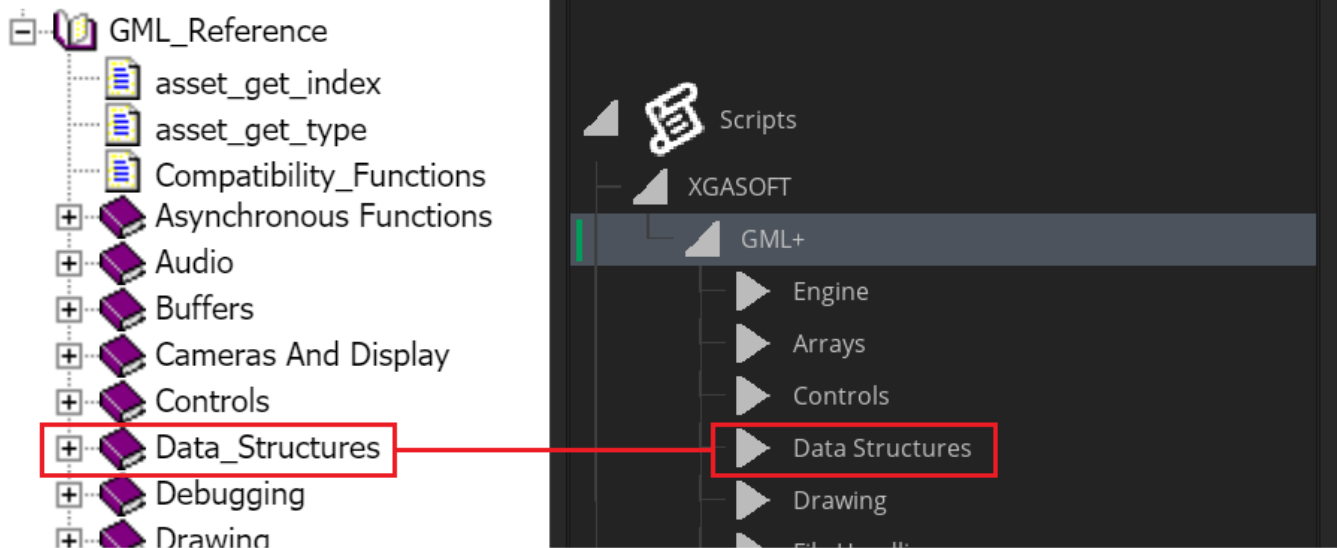


GML+ Reference Guide

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](#).



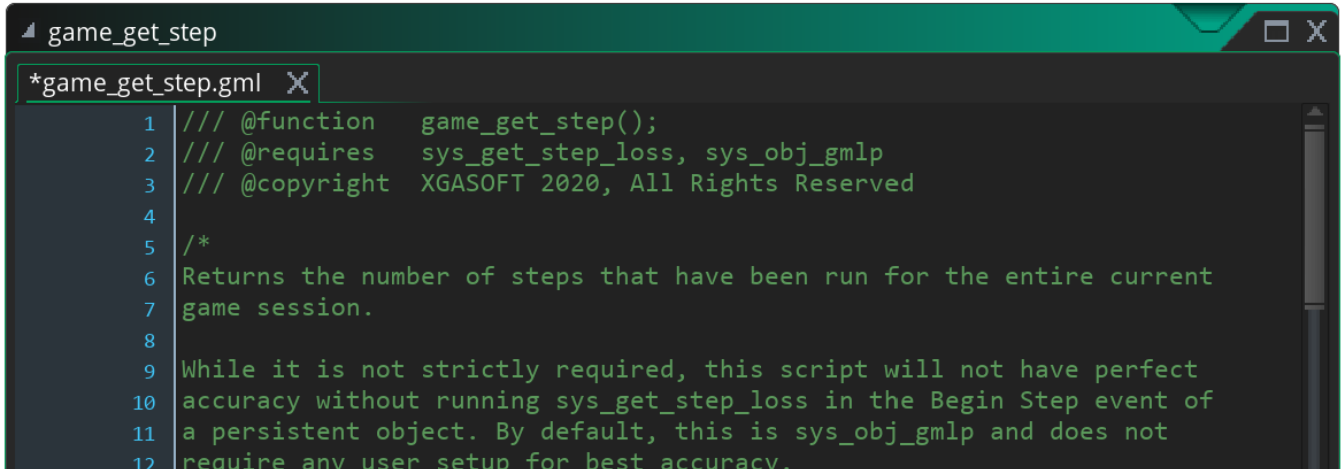
Self-Integrating Manager Object

Besides scripts, some GML+ components (such as built-in variables) rely on an object called `obj_server_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 `@requires` for a list of external dependencies.



```
game_get_step
*game_get_step.gml X
1 /// @function   game_get_step();
2 /// @requires  sys_get_step_loss, sys_obj_gmlp
3 /// @copyright XGASOFT 2020, All Rights Reserved
4
5 /*
6 Returns the number of steps that have been run for the entire current
7 game session.
8
9 While it is not strictly required, this script will not have perfect
10 accuracy without running sys_get_step_loss in the Begin Step event of
11 a persistent object. By default, this is sys_obj_gmlp and does not
12 require any user setup for best accuracy.
```

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_server_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
<code>mouse_hspeed</code>	real	Stores the current horizontal mouse speed, as a value of pixels.
<code>mouse_vspeed</code>	real	Stores the current vertical mouse speed, as a value of pixels.
<code>mouse_speed</code>	real	Stores the current directional mouse speed, as a value of pixels.
<code>mouse_direction</code>	real	Stores the current mouse direction of travel, as a value of degrees.
<code>mouse_xstart</code>	real	Stores the mouse X coordinate upon starting the game. If no mouse is used, -1 will be returned. Like the instance <code>xstart</code> variable, this variable is not read-only and can be redefined if needed.
<code>mouse_ystart</code>	real	Stores the mouse Y coordinate upon starting the game. If no mouse is used, -1 will be returned. Like the instance <code>ystart</code> variable, this variable is not read-only and can be redefined if needed.
<code>mouse_xprevious</code>	real	Stores the mouse X coordinate from the <i>previous</i> Step. Like the instance <code>xprevious</code> variable, this variable is not read-only and can be redefined if needed.
<code>mouse_yprevious</code>	real	Stores the mouse Y coordinate from the <i>previous</i> Step. Like the instance <code>yprevious</code> variable, this variable is not read-only and can be redefined if needed.

Variable	Type	Description
<code>mouse_visible</code>	boolean	Shows or hides the mouse, while preserving cursor state and/or sprite. Set to <code>true</code> by default. Replaces <code>cr_none</code> , which can no longer be used while <code>obj_server_gmlp</code> is present.

i 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
<code>frame_target</code>	real	Stores the amount of time a single frame <i>should</i> take to render at the current game speed (FPS), as a value of milliseconds.
<code>frame_time</code>	real	Stores the <i>actual</i> time the previous frame took to render, as a value of milliseconds.
<code>frame_delta</code>	real	Stores the difference (or delta) <i>between</i> 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 difference between* 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:

Deprecated Variables

The following variables were previously included for full functionality of `instance_link`, which no longer requires them. Universally updating these variables has a measurable performance cost whether or not they are used. This feature is now disabled by default, but can be re-enabled by modifying `obj_server_gmlp`:

- In the object **Create Event**, under "Configure GML+", change `expanded_image_previous` to `true`.
- No other modifications to `obj_server_gmlp` should be made.

Variable	Type	Description
<code>image_angle_previous</code>	real	Stores the instance sprite rotation from the <i>previous</i> step, as a value of degrees.
<code>image_xscale_previous</code>	real	Stores the instance sprite X scale from the <i>previous</i> step, as a value of degrees.
<code>image_yscale_previous</code>	real	Stores the instance sprite Y scale from the <i>previous</i> step, as a value of degrees.

i NOTE

`image_angle` should not be confused with `direction`. While both are built-in to each instance, `image_angle` modifies the *sprite* while `direction` modifies *movement*.



The "instance_find_var" Function

Syntax

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

Argument	Type	Description
<code>var</code>	string	The variable name to search for, as a string
<code>n</code>	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;  
}
```



The "instance_link" Function

Syntax

```
instance_link(parent, child, pos, rot, scale);
```

Argument	Type	Description
<code>parent</code>	instance	The parent instance to link properties <i>from</i>
<code>child</code>	instance/object/array	The child instance or collection of instances to link properties <i>to</i>
<code>pos</code>	boolean	Enables or disables linking child instance position relative to the parent
<code>rot</code>	boolean	Enables or disables linking child instance rotation relative to the parent
<code>scale</code>	boolean	Enables or disables linking child instance scale relative to the parent

Description

Links an object, instance, or array of them to the parent instance so that all child objects match the parent's position, rotation, and/or scale. Links are relative, meaning child objects can still have their own independent position, rotation, and scale as well.

Unlike the child, the parent must be a specific instance. Use `self` or `id` to indicate the running instance. If an object is input as the parent, the first randomly detected active instance of the object will be used.

Must be run in the Step Event for changes to position, rotation, and scale to apply continuously.

 IMPORTANT

This function depends on comparing values across time. It is highly recommended that you do not run `instance_link` inside a conditional statement, as this can lead to incorrect results.

Example

```
instance_link(obj_grid, obj_cell, true, true, true);
```



The "camera_get_view" Function

Syntax

```
camera_get_view(camera);
```

Argument	Type	Description
camera	camera	The camera ID to check

Description

Checks if a camera is currently assigned to a view, and if so, returns the view number (0-7). If the camera is not currently assigned to a view, -1 will be returned instead.

Example

```
var view = camera_get_view(my_cam);  
  
if (view > -1) {  
    camera_set_view_pos(view_camera[view], x, y);  
}
```



The "window_set_cursor_sprite" Function

Syntax

```
window_set_cursor_sprite(sprite);
```

Argument	Type	Description
<code>sprite</code>	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_server_gm1p` in the project when using this script. The `obj_server_gm1p` 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_server_gmlp` is present, `cr_none` will be ignored, and you should use `mouse_visible = false` to hide the cursor, or `window_set_cursor(cr_default)` to restore the system cursor instead.

Example

```
window_set_cursor_sprite(spr_cursor);
```




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 "ds_grid_empty" Function

Syntax

```
ds_grid_empty(id);
```

Argument	Type	Description
<code>id</code>	<code>ds_grid</code>	The data structure to check

Description

Checks whether the specified `ds_grid` is empty and returns `true` or `false`.

CAUTION

This function does **not** check whether or not the grid exists, and checking a non-existent grid will throw an error.

Example

```
var ds = ds_grid_create(1, 0);

if (ds_grid_empty(ds)) {
    ds_grid_insert_row(ds, 0, "init");
}
```



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_width` 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_width(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, [value]);
```

Argument	Type	Description
<code>id</code>	<code>ds_grid</code>	The data structure to insert a row into
<code>col</code>	integer	The index of the new column to insert
<code>[value]</code>	real/string	<i>Optional:</i> 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_width` will be increased by 1. For this reason, the new column index can be input as less than *or equal to* the current value of `ds_grid_width`.

Example

```
ds_grid_insert_col(my_grid, ds_grid_width(my_grid) - 2);  
ds_grid_insert_col(my_grid, ds_grid_width(my_grid) - 2, -1);
```



The "ds_grid_insert_row" Function

Syntax

```
ds_grid_insert_row(id, row, [value]);
```

Argument	Type	Description
<code>id</code>	<code>ds_grid</code>	The data structure to insert a row into
<code>row</code>	integer	The index of the new row to insert
<code>[value]</code>	real/string	<i>Optional:</i> 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 equal 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, source, [pos]);
```

Argument	Type	Description
id	ds_list	The data structure to add new data to
source	ds_list	The data structure to be added
pos	integer	<i>Optional:</i> 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_size(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] = "world!";  
  
ds_list_combine(my_list, my_other_list);
```



The "ds_list_add_list" Function

Syntax

```
ds_list_add_list(id, source);
```

Argument	Type	Description
id	ds_list	The data structure to add new data to
source	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, source);
```

Argument	Type	Description
id	ds_list	The data structure to add new data to
source	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, newlist);
```

Argument	Type	Description
<code>id</code>	<code>ds_list</code>	The data structure to add new data to
<code>oldlist</code>	<code>ds_list</code>	The data structure to be replaced
<code>newlist</code>	<code>ds_list</code>	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, newlist);
```

Argument	Type	Description
id	ds_list	The data structure to add new data to
oldmap	ds_map	The data structure to be replaced
newmap	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 "ds_struct_create" Function

Syntax

```
ds_struct_create();
```

Argument	Type	Description
N/A	N/A	No arguments

Description

Creates a new struct and returns the index (or ID).

A **struct** is a data structure similar to JSON (and also to GameMaker objects themselves!). Sometimes dubbed "lightweight objects", a struct contains a tree of key/value pairs which can themselves contain other structs, methods, or any other data type.

Like a `ds_map`, struct data is unordered. However, unlike *any other data structure*, structs are inherently non-scoped and non-volatile. This means any instance can access the struct by its ID without the need for copying the entire struct from one instance to another. Furthermore, instance variables like `x` and `y` can be given their own independent definitions within a struct.

Once created, a struct incurs no processing overhead and will exist until destroyed manually or until no more references to it exist, at which point it will be purged automatically. In other words, unlike other data structures, structs will typically not memory leak.

With so many positives, it's important to keep one negative in mind: structs are ideal for data with *static, pre-determined trees*. This means you should always know where data is stored in the struct rather than locate it programatically. While GML+ provides functions to do just that, they are potentially quite slow and should be used sparingly. For highly dynamic data, other data structures remain preferable.

Example

```
my_struct = ds_struct_create();
```




The "ds_struct_copy" Function

Syntax

```
ds_struct_copy(id);
```

Argument	Type	Description
id	struct	The source struct to be copied

Description

Copies the contents of a given struct and returns a new struct ID. This script will function recursively, also copying any structs and arrays within the root struct.

If the given input is invalid, `undefined` will be returned instead.

This type of copy operation is also known as a "deep clone", meaning data is truly duplicated in memory rather than merely referenced.

Example

```
my_new_struct = ds_struct_copy(my_struct);
```

The "ds_struct_destroy" Function

Syntax

```
ds_struct_destroy(id);
```

Argument	Type	Description
<code>id</code>	struct	The struct to be destroyed

Description

Destroys a struct, freeing its data from memory.

NOTE

Running this script is not strictly necessary, as structs will typically be purged automatically when all references to them are removed. However, it is good practice to manually remove structs when they are no longer needed to keep memory usage optimized.

Example

```
ds_struct_destroy(my_struct);
```



The "ds_struct_exists" Function

Syntax

```
ds_struct_exists(id, key);
```

Argument	Type	Description
<code>id</code>	struct	The struct to check
<code>key</code>	string	The key (i.e. struct content) to check existence of

Description

Checks if a variable name (as a string) exists within the given struct and returns `true` or `false`. Unlike the built-in `variable_struct_exists` function, this script will check recursively, including any structs-within-structs.

If multiple levels of struct exist, it is possible for the same key to occur multiple times with different values. To increase the speed and precision of this function, you can specify which level to search by prepending the key with any parent structs separated by a period. In this case, the **first parent specified must exist in the root struct**, but deeper levels will be recursed and are optional.

i NOTE

This script is for checking if the **contents** of a struct exist, not a struct itself.

To get the value of the key, if found, use `ds_struct_find_value`.

Example

```
// Both "font" and "text.font" are acceptable and will return the same value  
if (ds_struct_exists(my_struct, "text.font")) {  
    var my_font = ds_struct_find_value(my_struct, "text.font");  
  
    draw_set_font(my_font);  
}
```

The "ds_struct_equals" Function

Syntax

```
ds_struct_equals(var1, var2);
```

Argument	Type	Description
<code>var1</code>	struct	The source struct to be compared
<code>var2</code>	struct	The target struct to compare with

Description

Compares the contents of two structs and returns `true` or `false` depending on whether the contents and their values are equal. This script will function recursively, also comparing any structs and arrays within the root struct.

Note that volatile data structure and method contents cannot be compared due to GameMaker's handling of references for these types, and as such may behave differently than expected:

- Data structures will be compared by reference only and thus will only return `true` if the exact same structure is referenced. Identical copies will return `false` because each copy has a unique (non-matching) index.
- Methods cannot be compared by contents *or* reference and therefore will always return `true` provided a method exists at the same location in each struct.

Example

```
var struct1 = { name: "John Doe", age: 30 };
var struct2 = { age: 30, name: "John Doe" };

if (ds_struct_equals(struct1, struct2)) {
    show_message("Structs are equal!");
}
```




The "ds_struct_find_first" Function

Syntax

```
ds_struct_find_first(id);
```

Argument	Type	Description
id	struct	The struct to check

Description

Returns the name of the first key in the given struct, as a string. Further searches can then be performed with `ds_struct_find_next`.

If the struct is empty, `undefined` will be returned instead.

Note that because structs are a non-ordered data format, data may be returned in a different order than originally declared in code. Changes to the struct may also change the order in which data is returned with this function.

Example

```
var key = ds_struct_find_first(my_struct);  
var val = true;  
  
variable_struct_set(my_struct, key, val);
```



The "ds_struct_find_last" Function

Syntax

```
ds_struct_find_last(id);
```

Argument	Type	Description
<code>id</code>	struct	The struct to check

Description

Returns the name of the last key in the given struct, as a string. Further searches can then be performed with `ds_struct_find_previous`.

If the struct is empty, `undefined` will be returned instead.

Note that because structs are a non-ordered data format, data may be returned in a different order than originally declared in code. Changes to the struct may also change the order in which data is returned with this function.

Example

```
var key = ds_struct_find_last(my_struct);  
var val = true;  
  
variable_struct_set(my_struct, key, val);
```



The "ds_struct_find_next" Function

Syntax

```
ds_struct_find_next(id, key);
```

Argument	Type	Description
<code>id</code>	struct	The struct to check
<code>key</code>	string	The key (i.e. struct content) to begin search from

Description

Returns the name of the next key in the given struct, as a string. Search will begin from the given key (for example, as returned by `ds_struct_find_first`).

If the struct is empty or no further key exists, `undefined` will be returned instead.

Note that because structs are a non-ordered data format, data may be returned in a different order than originally declared in code. Changes to the struct may also change

the order in which data is returned with this function.

Example

```
var key = ds_struct_find_first(my_struct);  
  
key = ds_struct_find_next(my_struct, key);  
var val = true;  
  
variable_struct_set(my_struct, key, val);
```



The "ds_struct_find_previous" Function

Syntax

```
ds_struct_find_previous(id, key);
```

Argument	Type	Description
<code>id</code>	struct	The struct to check
<code>key</code>	string	The key (i.e. struct content) to begin search from

Description

Returns the name of the previous key in the given struct, as a string. Search will begin from the given key (for example, as returned by `ds_struct_find_last`).

If the struct is empty or no further key exists, `undefined` will be returned instead.

Note that because structs are a non-ordered data format, data may be returned in a different order than originally declared in code. Changes to the struct may also change

the order in which data is returned with this function.

Example

```
var key = ds_struct_find_last(my_struct);  
  
key = ds_struct_find_previous(my_struct, key);  
var val = true;  
  
variable_struct_set(my_struct, key, val);
```




The "ds_struct_find_index" Function

Syntax

```
ds_struct_find_index(id, key);
```

Argument	Type	Description
id	struct	The struct to check
key	string	The key (i.e. struct content) to check index of

Description

Checks if a variable name (as a string) exists within the given struct and returns the **index** of the parent struct. The result can then be used with standard struct accessors (see official GameMaker documentation for details). Naturally, this is primarily useful for checking recursively, including any structs-within-structs.

If multiple levels of struct exist, it is possible for the same key to occur multiple times with different values. To increase the speed and precision of this function, you can specify

which level to search by prepending the key with any parent structs separated by a period. In this case, the **first parent specified must exist in the root struct**, but deeper levels will be recursed and are optional.

If the specified key doesn't exist, `-1` will be returned instead. To detect whether the key exists first, use `ds_struct_exists`.

Example

```
var struct_text = ds_struct_find_index(my_struct, "font");  
  
draw_set_font(struct_text.font);  
draw_set_color(struct_text.color);
```



The "ds_struct_find_value" Function

Syntax

```
ds_struct_find_value(id, key);
```

Argument	Type	Description
<code>id</code>	struct	The struct to check
<code>key</code>	string	The key (i.e. struct content) to check value of

Description

Checks if a variable name (as a string) exists within the given struct and returns the value. Unlike the built-in `variable_struct_get` function, this script will check recursively, including any structs-within-structs.

If multiple levels of struct exist, it is possible for the same key to occur multiple times with different values. To increase the speed and precision of this function, you can specify which level to search by prepending the key with any parent structs separated by a

period. In this case, the **first parent specified must exist in the root struct**, but deeper levels will be recursed and are optional.

If the specified key doesn't exist, `undefined` will be returned instead. To determine whether the key exists first, use `ds_struct_exists`.

Example

```
// Both "font" and "text.font" are acceptable and will return the same value  
if (ds_struct_exists(my_struct, "text.font")) {  
    var my_font = ds_struct_find_value(my_struct, "text.font");  
  
    draw_set_font(my_font);  
}
```



The "ds_struct_set" Function

Syntax

```
ds_struct_set(id, key, value);
```

Argument	Type	Description
<code>id</code>	struct	The struct to modify
<code>key</code>	string	The key to assign a value to
<code>value</code>	any	The value (number, string, array, struct, data structure, etc.) to assign

Description

Assigns a value to the specified key within the given struct. The struct must have been previously created (e.g. with `ds_struct_create`).

If multiple levels of struct exist, it is possible for the same key to occur multiple times with different values. To increase the speed and precision of this function, you can specify which level to search by prepending the key with any parent structs separated by a

period. In this case, the **first parent specified must exist in the root struct**, but deeper levels will be recursed and are optional, provided the key already exists. If the key does not exist, any parents will be created as well in the order specified.

To get the value of a key once set, use `ds_struct_find_value`.

Example

```
// Both "font" and "text.font" are acceptable and will assign the same value  
ds_struct_set(my_struct, "text.font", fnt_Arial);
```



The "ds_struct_read" Function

Syntax

```
ds_struct_read(str);
```

Argument	Type	Description
<code>str</code>	string	A previously-written struct string to be converted into a struct

Description

An alias of `json_parse`. Converts a string previously written with `ds_struct_write` back into a struct, including any sub-structs it may contain.

Example

```
var file = file_text_open_read("settings.json");  
my_struct = ds_struct_read(file_text_read_string(file));  
file_text_close(file);
```



The "ds_struct_write" Function

Syntax

```
ds_struct_write(id, [pretty]);
```

Argument	Type	Description
<code>id</code>	struct	The struct to encode as a string
<code>[pretty]</code>	boolean	<i>Optional:</i> Enables or disables formatting the string with linebreaks and indentation

Description

Converts a struct and its contents to a string, optionally with "pretty-print" to separate values by line with proper indenting.

 CAUTION

Note that only single values, arrays, strings, and sub-structs are interpreted by this function. Other data structures, like `ds_map` or `ds_grid`, are stored by reference **only** and will be written as a numerical index rather than their actual contents. To avoid this behavior, first use the respective `ds*_write` function on these data structures so they will be included as strings which can be decoded later with `ds*_read`.

Example

```
var file = file_text_open_write("settings.json");
file_text_write_string(
    file,
    ds_struct_write(my_struct, true)
);
file_text_close(file);
```

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_white` 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
<code>color</code>	color	An RGB color to convert to Hex

Description

Returns a Hex color code, as a string, from an RGB color. Result will be formatted as

```
#RRGGBB .
```

Example

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



The "sprite_get_index" Function

Syntax

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

Argument	Type	Description
<code>sprite</code>	sprite	The sprite to retrieve image index of
<code>[offset]</code>	real	<i>Optional:</i> 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  
var offset = -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
<code>sprite</code>	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

```
var speed_fps = sprite_get_speed_fps(my_sprite);
```



The "sprite_get_speed_real" Function

Syntax

```
sprite_get_speed_real(sprite);
```

Argument	Type	Description
<code>sprite</code>	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

```
var speed_real = sprite_get_speed_real(my_sprite);
```




The "draw_get_surface" Function

Syntax

```
draw_get_surface(surf);
```

Argument	Type	Description
<code>surf</code>	surface	The surface to retrieve from memory

Description

Surfaces are "volatile", meaning their data can be erased from memory under certain common conditions, such as resizing or minimizing the game window.

This function will return the surface from cached memory to ensure it always exists before drawing. Unlike normal `if (surface_exists(surf))` statements, `draw_get_surface` will also preserve surface *contents*, so you never need to regenerate the surface manually. This is especially useful when surfaces are drawn to dynamically and cannot be recreated, such as for blood splatter or tire tracks resulting from unique and unrepeatable player actions.

Though it might seem strange to input and return the same variable, this is necessary because the input surface ID might change if the surface does not exist and must be recreated.

Example

```
my_surf = draw_get_surface(my_surf);  
  
draw_surface_ext(my_surf, x, y, 1.5, 1.5, 25, c_red, 0.5);
```



The "surface_read" Function

Syntax

```
surface_read(surf);
```

Argument	Type	Description
<code>str</code>	string	A previously-written surface string to convert into a surface

Description

Returns a surface from a base64-encoded string previously generated by `surface_write`. Invalid strings will return `undefined`. Especially useful for save files and networking.

Example

```
ini_open(working_directory + "save.dat");  
var my_surf = ini_read_string("data", "screenshot", undefined);  
ini_close();  
  
if (!is_undefined(my_surf)) {  
    draw_surface(my_surf, x, y);  
}
```



The "surface_write" Function

Syntax

```
surface_write(surf);
```

Argument	Type	Description
<code>surf</code>	surface	The surface to encode as a string

Description

Returns a surface as a base64-encoded string. Use `surface_read` to convert back into a surface. Especially useful for save files and networking.

Example

```
ini_open(working_directory + "save.dat");  
ini_write_string("data", "screenshot", surface_write(application_surface));  
ini_close();
```



The "file_list" Function

Syntax

```
file_list(dname, attr, [recurse]);
```

Argument	Type	Description
<code>dname</code>	string	The full path of the target directory, including drive letter
<code>attr</code>	integer/constant	Enables filtering results as directories (<code>fa_directory</code>) or files (anything else)
<code>[recurse]</code>	boolean	<i>Optional:</i> 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 `recurse` 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 `working_directory` or elsewhere previously granted access via the `get_save_filename` function. On desktop platforms, this limitation can be removed by disabling the filesystem sandbox in Game Settings.

Example

```
var dirs = file_list("C:\\my\\folder", fa_directory, true);  
var files = file_list("C:\\my\\folder", 0);
```



The "file_move" Function

Syntax

```
file_move(fname, dest);
```

Argument	Type	Description
<code>fname</code>	string	The full path of the file to move, including name and extension
<code>dest</code>	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 `true` 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 `working_directory` or elsewhere previously granted access via the `get_save_filename` function. On desktop platforms, this limitation can be removed by disabling the filesystem sandbox in Game Settings.

Example

```
file_move(working_directory + "temp.sav", working_directory + "saves");
```



The "filename_is_dir" Function

Syntax

```
filename_is_dir(fname);
```

Argument	Type	Description
<code>fname</code>	string	The full path to check

Description

Checks if a filename appears to be a directory based on string properties and returns `true` or `false`. This is faster than `directory_exists`, 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

```
var file = file_find_first("C:\\*", fa_directory);  
  
if (filename_is_dir(file)) {  
    //Directory exists!  
}
```



The "device_mouse_check_region" Function

Syntax

```
device_mouse_check_region(device, x, y, rot, width, [height], [halign,  
valign]);
```

Argument	Type	Description
<code>device</code>	integer	The mouse or touch point to check, where 0 is first
<code>x</code>	real	The horizontal room coordinate for the region to check
<code>y</code>	real	The horizontal room coordinate for the region to check
<code>rot</code>	real	The rotation of the region to check, if rectangular
<code>width</code>	real	The width (or radius) in pixels of the region to check
<code>[height]</code>	real	<i>Optional:</i> The height in pixels of the region to check (use none for circular)
<code>[halign]</code>	constant	<i>Optional:</i> The horizontal alignment of the region to check (use none for <code>fa_center</code>)
<code>[valign]</code>	constant	<i>Optional:</i> The vertical alignment of the region to check (use none for <code>fa_middle</code>)

Description

Checks whether the mouse is currently within a certain region relative to room coordinates and returns `true` 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 `valign` values using font alignment constants such as `fa_left` and `fa_top`. Alignment **must** be input as a pair.

Example

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

//Diamond region
if (device_mouse_check_region(0, 640, 480, 45, 256, 256)) {
    //Action
}

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



The "device_mouse_check_region_gui" Function

Syntax

```
device_mouse_check_region_gui(device, x, y, rot, width, [height], [halign,  
valign]);
```

Argument	Type	Description
<code>device</code>	integer	The mouse or touch point to check, where 0 is first
<code>x</code>	real	The horizontal GUI coordinate for the region to check
<code>y</code>	real	The horizontal GUI coordinate for the region to check
<code>rot</code>	real	The rotation of the region to check, if rectangular
<code>width</code>	real	The width (or radius) in pixels of the region to check
<code>[height]</code>	real	<i>Optional:</i> The height in pixels of the region to check (use none for circular)
<code>[halign]</code>	constant	<i>Optional:</i> The horizontal alignment of the region to check (use none for <code>fa_center</code>)
<code>[valign]</code>	constant	<i>Optional:</i> The vertical alignment of the region to check (use none for <code>fa_middle</code>)

Description

Checks whether the mouse is currently within a certain region relative to GUI coordinates and returns `true` 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 `valign` values using font alignment constants such as `fa_left` and `fa_top`. Alignment **must** be input as a pair.

Example

```
//Rectangular region  
if (device_mouse_check_region_gui(0, 640, 480, 0, 512, 384)) {  
    //Action  
}  
  
//Diamond region  
if (device_mouse_check_region_gui(0, 640, 480, 45, 256, 256)) {  
    //Action  
}  
  
//Circular region with custom alignment  
if (device_mouse_check_region_gui(0, 256, 128, 0, 512, fa_left, fa_top)) {  
    //Action  
}
```



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_server_gm1p` in the current project. The `obj_server_gm1p` 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_even(game_get_step())) {  
    //Action on even Steps  
} else {  
    //Action on odd Steps  
}
```



The "game_get_time" Function

Syntax

```
game_get_time([type]);
```

Argument	Type	Description
[type]	constant	<i>Optional:</i> Sets whether to return a value in milliseconds (<code>gamespeed_fps</code>) or microseconds (<code>gamespeed_microseconds</code>) (use none for milliseconds)

Description

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

Example

```
var hours_played = floor(game_get_time(gamespeed_fps)/1000/60/60);  
var ms_played = (game_get_time(gamespeed_microseconds)/100000);
```



The "foreach" Statement

Syntax

```
foreach (DATA as "VALUE") call {  
    VALUE  
}
```

Or

```
foreach (DATA as "KEY" of "VALUE") call {  
    KEY  
    VALUE  
}
```

Description

Iterates through a given subject and returns the value of each item to a custom variable, which can be used to perform operations on each item in the subject.

Similar to a standard `for` loop, but with a more convenient syntax that shortcuts common loop operations. `foreach` supports iterating specific data types, including arrays, data structures, integers, objects, and strings.

If it is necessary to know the current index of each iteration, a "key" custom variable can also be supplied **before** the custom value variable. The index will then be stored in the key for future reference.

Value and key variables **must** be input as strings, but will be referenced in custom code by their literal names instead.

Note that `as`, `of`, and `call` are special keywords for this function. Also note that the closing parenthesis comes **before** the `call` statement.

In the case of multidimensional arrays, only the dimension provided will be iterated. Any sub-dimensions will be returned in the custom value variable and can be handled in custom code.

In the case of objects, iteration will be performed through all active instances of an object in the current room.

CAUTION

Note that in some cases this function may not return the expected result due to the way GameMaker handles pointers. This means some types of data can share the same value, and whichever one happens to be first will take priority.

Example

```
var my_array = ["a", "b", "c"];
var my_list = ds_list_create();

// Add each item in the example array to the example list
foreach (my_array as "letter" call) {
    ds_list_add(my_list, letter);
}

// Add each item in the example list to the example array
foreach (my_list as "index" of "letter") call {
    my_array[index] = letter;
}
```



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 the same basic principles in different ways, allowing users to find which is easiest for their particular use-cases. In this section, we'll examine each one in detail.



The "angle_reflect" Function

Syntax

```
angle_reflect(deg, mirror);
```

Argument	Type	Description
deg	real	The angle to reflect, in degrees
mirror	real	The mirror angle to reflect from, in degrees

Description

Returns an angle in degrees reflected from a mirror "line" with the given angle in degrees. A mirror angle of 0 degrees is considered horizontal.



TIP

See the included interactive demo for a visual example of this function!

Example

```
var angle_in = point_direction(mouse_x, mouse_y, mirror_x, mirror_y);
var angle_out = angle_reflect(angle_in, mirror_rot);
var dist = point_distance(mouse_x, mouse_y, mirror_x, mirror_y);

// Draw mirror
draw_line(
    mirror_x - rot_dist_x(64, mirror_rot), mirror_y - rot_dist_y(64,
mirror_rot),
    mirror_x + rot_dist_x(64, mirror_rot), mirror_y + rot_dist_y(64,
mirror_rot)
);

// Draw angle in
draw_arrow(mouse_x, mouse_y, mirror_x, mirror_y, 16);

// Draw angle out
draw_arrow(
    mirror_x, mirror_y,
    mirror_x + rot_dist_x(dist, angle_out), mirror_y + rot_dist_y(dist,
angle_out),
    16
);
```



The "angle_refract" Function

Syntax

```
angle_refract(deg, mirror, outer_index, inner_index);
```

Argument	Type	Description
<code>deg</code>	real	The angle to refract, in degrees
<code>mirror</code>	real	The mirror angle to refract from, in degrees
<code>outer_index</code>	real	The refraction index of the input angle
<code>inner_index</code>	real	The refraction index of the mirror

Description

Refractions are *sided*, meaning the resulting refraction angle depends on which direction the input angle is coming from (relative to the orientation of the mirror "line"). Angles between 0-180 degrees are considered to be *inside* the refractive surface, whereas angles between 180-360 degrees are considered to be *outside*. A mirror angle of 0 degrees is considered horizontal.

How much an angle refracts depends on the difference between the substance it is coming from and the substance it is going into. The refractivity of a substance is called the "**refraction index**". Indices are given for both the area *outside* and *inside* the mirror "line", where, for example, an index of 1.0 is a vacuum, 1.0003 is air, 1.333 is water, and 2.42 is diamond. Mathematically, refraction index typically ranges from 1.0-3.0, no less or greater.

In addition to controlling the *amount* of refractivity, these indices also determine the refraction "**critical angle**", at which point internal refractions will become reflections instead. Higher index equals lower critical angle, refracting sharp angles only and merely reflecting the rest (which is responsible for that diamond glitter!).



See the included interactive demo for a visual example of this function!

Example

```
var angle_in = point_direction(mouse_x, mouse_y, mirror_x, mirror_y);
var angle_out = angle_refract(angle_in, mirror_rot, 1, 3);
var dist = point_distance(mouse_x, mouse_y, mirror_x, mirror_y);

// Draw mirror
draw_line(
    mirror_x - rot_dist_x(64, mirror_rot), mirror_y - rot_dist_y(64,
mirror_rot),
    mirror_x + rot_dist_x(64, mirror_rot), mirror_y + rot_dist_y(64,
mirror_rot)
);

// Draw angle in
draw_arrow(mouse_x, mouse_y, mirror_x, mirror_y, 16);

// Draw angle out
draw_arrow(
    mirror_x, mirror_y,
    mirror_x + rot_dist_x(dist, angle_out), mirror_y + rot_dist_y(dist,
angle_out),
    16
);
```



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.



TIP

See the included interactive demo for a visual example of this function!

Example

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



The "rot_dist_x" Function

Syntax

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

Argument	Type	Description
<code>dist</code>	real	The distance from the rotation center point
<code>[deg]</code>	real	<i>Optional:</i> 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.



TIP

See the included interactive demo for a visual example of this function!

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
<code>dist</code>	real	The distance from the rotation center point
<code>[deg]</code>	real	<i>Optional:</i> 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.



TIP

See the included interactive demo for a visual example of this function!

Example

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



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	<i>Optional:</i> 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	<i>Optional:</i> 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_vec_x" Function

Syntax

```
rot_vec_x(x1, y1, x2, y2, [deg]);
```

Argument	Type	Description
x1	real	The horizontal center point
y1	real	The vertical center point
x2	real	The horizontal distance from the rotation center point
y2	real	The vertical distance from the rotation center point
[deg]	real	<i>Optional:</i> 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_vec_x(128, 128, 64, 64, image_angle);  
y = rot_vec_y(128, 128, 64, 64);
```



The "rot_vec_y" Function

Syntax

```
rot_vec_y(x1, y1, x2, y2, [deg]);
```

Argument	Type	Description
x1	real	The horizontal center point
y1	real	The vertical center point
x2	real	The horizontal distance from the rotation center point
y2	real	The vertical distance from the rotation center point
[deg]	real	<i>Optional:</i> 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_vec_x(128, 128, 64, 64, image_angle);  
y = rot_vec_y(128, 128, 64, 64);
```



The "approx" Function

Syntax

```
approx(value, min, [max]);
```

Argument	Type	Description
<code>value</code>	real	The numerical value to check
<code>min</code>	real	The minimum closeness, or alternatively, minimum value to accept
<code>[max]</code>	real	<i>Optional:</i> The maximum value to accept (use none for +/- <code>min</code>)

Description

Checks if a value is between two numbers and returns `true` 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 player on left or right  
}
```



The "emod" Function

Syntax

```
emod(value, divisor);
```

Argument	Type	Description
value	real	The numerical value to modify
divisor	real	The divisor to apply

Description

Returns the modulo (remainder) of a number with Euclidean division. Unlike the built-in `mod (%)` operator, this function will always return a positive value between zero and the divisor (`div`).

Example

```
image_angle = emod(image_angle, 360);
```



The "interp" Function

Syntax

```
interp(a, b, amount, ease, [bx1, by1, bx2, by2]);
```

Argument	Type	Description
<code>a</code>	real	The starting value to interpolate from
<code>b</code>	real	The target value to interpolate to
<code>amount</code>	real	The amount, or percentage to interpolate between values (variable recommended)
<code>ease</code>	integer/macro	Sets the easing method for the interpolation (see options below)
<code>[bx1]</code>	real	<i>Optional:</i> X percentage for left control point of a cubic bezier curve (0-1)
<code>[by1]</code>	real	<i>Optional:</i> Y percentage for left control point of a cubic bezier curve (0-1)
<code>[bx2]</code>	real	<i>Optional:</i> X percentage for right control point of a cubic bezier curve (0-1)
<code>[by2]</code>	real	<i>Optional:</i> Y percentage for right control point of a cubic bezier curve (0-1)
<code>[curve]</code>	curve	<i>Optional:</i> Animation curve asset to use if <code>ease</code> is set to <code>ease_curve</code>

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. Simple `true` or `false` values can be used here for sine or linear interpolation, respectively, but in addition to these basic modes there are 30 different easing techniques, plus other custom techniques, featured below. Built-in easing techniques are ordered from shallowest to deepest curve.



TIP

See the included interactive demo for a visual example of this function!

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
<code>ease_none</code>	-4	<code>ease_expo_in</code>	17
<code>ease_sin</code>	1	<code>ease_expo_out</code>	18
<code>ease_sin_in</code>	2	<code>ease_circ</code>	19
<code>ease_sin_out</code>	3	<code>ease_circ_in</code>	20
<code>ease_quad</code>	4	<code>ease_circ_out</code>	21
<code>ease_quad_in</code>	5	<code>ease_rubber</code>	22
<code>ease_quad_out</code>	6	<code>ease_rubber_in</code>	23
<code>ease_cubic</code>	7	<code>ease_rubber_out</code>	24
<code>ease_cubic_in</code>	8	<code>ease_elastic</code>	25
<code>ease_cubic_out</code>	9	<code>ease_elastic_in</code>	26
<code>ease_quart</code>	10	<code>ease_elastic_out</code>	27
<code>ease_quart_in</code>	11	<code>ease_bounce</code>	28
<code>ease_quart_out</code>	12	<code>ease_bounce_in</code>	29
<code>ease_quint</code>	13	<code>ease_bounce_out</code>	30
<code>ease_quint_in</code>	14	<code>ease_bezier</code>	31
<code>ease_quint_out</code>	15	<code>ease_curve</code>	32

Easing Macro	Value		Easing Macro	Value
ease_expo	16			

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 cubic-bezier.com for an interactive visual example.

Other types of curves can be created visually in GameMaker 2.3 and newer. To use these curves with `interp`, specify `ease_curve` as the mode and then supply an additional argument pointing to the animation curve asset desired. Note that only the first channel in an animation curve asset will be used.

! IMPORTANT

Only a custom bezier curve *or* a custom curve asset can be input. Both cannot be input at the same time.

Example

```
duration = 5;
time += delta_time/1000000;

x = interp(0, 50, time/duration, ease_quart);
y = interp(0, 50, time/duration, ease_bezier, 0.66, -0.33, 0.33, 1.33);
z = interp(0, 50, time/duration, ease_curve, my_curve);
```



The "is_even" Function

Syntax

```
is_even(n);
```

Argument	Type	Description
<code>n</code>	real	A number to check parity of

Description

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

Example

```
if (is_even(var_num)) {  
    show_message("I'm even!");  
} else {  
    show_message("I'm odd!");  
}
```



The "is_odd" Function

Syntax

```
is_odd(n);
```

Argument	Type	Description
<code>n</code>	real	A number to check parity of

Description

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

Example

```
if (is_odd(var_num)) {  
    show_message("I'm odd!");  
} else {  
    show_message("I'm even!");  
}
```



The "round_to" Function

Syntax

```
round_to(value, multiple);
```

Argument	Type	Description
value	real	The value to round
multiple	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 = round_to(score, 10);
```



The "timer" Function

Syntax

```
timer(id, [duration]);
```

Argument	Type	Description
<code>id</code>	string	Sets a unique timer ID, as a string
<code>[duration]</code>	real	<i>Optional:</i> 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 `true`. (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.



See the included interactive demo for a visual example of this function!

Example

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



The "timer_set_time" Function

Syntax

```
timer_set_time(id, duration, [instance]);
```

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

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.



TIP

See the included interactive demo for a visual example of this function!

Example

```
timer_set_time("t_alarm", 5);  
timer_set_time("t_other", 5, my_other_inst);
```



The "timer_get_time" Function

Syntax

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

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

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

```
var inst_streetlight = instance_find(obj_streetlight, 0);  
var race_started = (timer_get_time("t_streetlight", inst_streetlight) == 0);
```



The "timer_set_pause" Function

Syntax

```
timer_set_pause(id, enable, [instance]);
```

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

Description

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

Example

```
timer_set_pause("t_alarm", other);  
timer_set_pause(all, true, my_other_inst);
```



The "timer_get_pause" Function

Syntax

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

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

Description

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

Example

```
if (timer_get_pause("my_timer")) {  
    timer_set("my_timer", 5);  
}
```


The "timer_set_speed" Function

Syntax

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

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

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
<code>id</code>	string	The unique timer ID to check, as a string
<code>[instance]</code>	instance	<i>Optional:</i> Sets the object instance containing the timer to check (use none for <code>self</code>)

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

```
var my_timer_speed = timer_get_speed("my_timer");  
  
my_timer_speed += (3 - my_timer_speed)*0.25;  
  
timer_set_speed("my_timer", my_timer_speed);
```



The "wait" Function

Syntax

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

Argument	Type	Description
<code>duration</code>	real	Sets the duration of time to wait, in seconds
<code>[offset]</code>	real	<i>Optional:</i> 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 != xprevious) or (y != yprevious) {
    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 "collision_line_meeting" Function

Syntax

```
collision_line_meeting(x1, y1, x2, y2, obj, prec, notme);
```

Argument	Type	Description
<code>x1</code>	real	The X coordinate of the starting point of a line to check
<code>y1</code>	real	The Y coordinate of the starting point of a line to check
<code>x2</code>	real	The X coordinate of the ending point of a line to check
<code>y2</code>	real	The Y coordinate of the ending point of a line to check
<code>obj</code>	object/instance	The object or instance to check for collisions (<i>or</i> Keyword <code>aLL</code> for all active instances)
<code>prec</code>	boolean	Enables or disables checking precise collision masks (if any)
<code>notme</code>	boolean	Enables or disables excluding other instances of the running object from collision checks

Description

Finds the exact point of collision between two sets of coordinates and an input object.

Results are returned as a **struct** containing three keys: the nearest colliding instance `id`, plus the `x` and `y` values of the exact point of collision. If no collision is found, the instance ID will be considered `noone` and the position unchanged from `x2` and `y2`.

The object to check for collisions can be an object ID (in which case all instances of the object will be considered, a single instance ID, or keyword `a11` for all active instances. Set `notme` to `true` to exclude instances of the running object from consideration.

Collisions can be evaluated precisely (per-pixel, with a collision mask) or by bounding box, depending on whether `prec` is `true` or `false`. This setting also depends on the type of collision mask defined in the Sprite Editor (i.e. a precise mask must be created for enabling precise collisions to have any effect).

Example

```
var coll = collision_line_meeting(x, y, x + mov_speed, y, obj_wall, false,
true);

if (coll.id != noone) {
    // Limit movement to point of collision, if any
    x = coll.x;
    y = coll.y;
} else {
    // Otherwise move forward freely
    x += mov_speed;
}
```



The "string_explode" Function

Syntax

```
string_explode(str, delim, [limit]);
```

Argument	Type	Description
<code>str</code>	string	The string to split into an array
<code>delim</code>	string	A repeated substring at which point the string will be split
<code>[limit]</code>	integer	<i>Optional:</i> Sets a limit on the number of times the string will be split

Description

Splits a string into a 1D array, using a delimiter character (or substring) to separate contents. The delimiter will not be included in the resulting strings.

If a limit value is supplied, only that number of delimiter matches will be made, and the final value in the array will contain the remainder of the unsplit string. To exclude the remainder from the array, input the limit value as negative.

Note that the resulting strings will automatically be trimmed, meaning you do not need to include spaces in the delimiter string (unless space itself is the delimiter). Spaces will automatically be removed.

Example

```
var notes = "do | re | mi | fa | so | la | ti | do";  
  
notes = string_explode(notes, "|");  
  
draw_text(x, y, notes[0]);
```



The "string_implode" Function

Syntax

```
string_implode(array, [limit]);
```

Argument	Type	Description
<code>array</code>	array	The array to combine into a string
<code>[delim]</code>	string	<i>Optional:</i> A repeated substring to separate combined items

Description

Combines the contents of a 1D array into a string, optionally separated by a delimiter character (or substring).

Note that all array contents will be treated as strings. If the array contains pointers to other types of data, the pointer will be written literally rather than writing the contents of the data itself.

Example

```
var notes = ["do", "re", "mi", "fa", "so", "la", "ti", "do"];  
  
notes = string_implode(notes, "|");  
  
draw_text(x, y, notes);
```



The "string_lower_all" Function

Syntax

```
string_lower_all(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Converts a string to all lowercase letters. Applies to English characters A-Z only.

! INFO

Like the built-in `string_lower` function, but nearly 2x faster!

Example

```
var mystring = "HELLO, WORLD!";  
  
draw_text(25, 25, string_lower_all(mystring));  
  
// Result: "hello, world!"
```



The "string_lower_first" Function

Syntax

```
string_lower_first(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Returns a string with the first letter of the first word lowercase (can be used for camelCase, for example). Only applies to English characters A-Z.

Example

```
var mystring = "HELLO, WORLD!";  
  
draw_text(25, 25, string_lower_first(mystring));  
  
// Result: "hELLO, WORLD!"
```




The "string_lower_words" Function

Syntax

```
string_lower_words(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Returns a string with the first letter of each word uncapitalized (can be used for camelCase, for example). Only applies to English characters A-Z.

Example

```
var mystring = "HELLO, WORLD!";  
  
draw_text(25, 25, string_lower_words(mystring));  
  
// Result: "hELLO, wORLD!"
```



The "string_upper_all" Function

Syntax

```
string_upper_all(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Converts a string to all uppercase letters. Applies to English characters A-Z only.

! INFO

Like the built-in `string_upper` function, but nearly 2x faster!

Example

```
var mystring = "hello, world!";  
  
draw_text(25, 25, string_upper_all(mystring));  
  
// Result: "HELLO, WORLD!"
```



The "string_upper_first" Function

Syntax

```
string_upper_first(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Returns a string with the first letter of the first word capitalized. Only applies to English characters A-Z.

Example

```
var mystring = "hello, world!";  
  
draw_text(25, 25, string_upper_first(mystring));  
  
// Result: "Hello, world!"
```



The "string_upper_words" Function

Syntax

```
string_upper_words(str);
```

Argument	Type	Description
<code>str</code>	string	The string to modify

Description

Returns a string with the first letter of each word capitalized. Only applies to English characters A-Z.

Example

```
var mystring = "hello, world!";  
  
draw_text(25, 25, string_upper_words(mystring));  
  
// Result: "Hello, World!"
```



The "string_trim" Function

Syntax

```
string_trim(str, [filter]);
```

Argument	Type	Description
<code>str</code>	string	The string to modify
<code>[filter]</code>	string	<i>Optional:</i> A custom list of characters to be trimmed

Description

Removes spaces from either side of the string and returns the trimmed result.

If a filter is supplied, **any** character in the filter string will be trimmed instead. (To also trim spaces, include a space in the filter string.)

Example

```
var notes = " +[do, re, mi, fa, so, la, ti, do] % ";  
  
notes = string_trim(notes, " []+%");  
  
draw_text(x, y, notes);
```



The "string_trim_left" Function

Syntax

```
string_trim_left(str, [filter]);
```

Argument	Type	Description
<code>str</code>	string	The string to modify
<code>[filter]</code>	string	<i>Optional:</i> A custom list of characters to be trimmed

Description

Removes spaces from the left side of the string and returns the trimmed result.

If a filter is supplied, **any** character in the filter string will be trimmed instead. (To also trim spaces, include a space in the filter string.)

Example


```
var notes = " +[do, re, mi, fa, so, la, ti, do] % ";  
  
notes = string_trim_left(notes, " [+");  
  
draw_text(x, y, notes);
```



The "string_trim_right" Function

Syntax

```
string_trim_right(str, [filter]);
```

Argument	Type	Description
<code>str</code>	string	The string to modify
<code>[filter]</code>	string	<i>Optional:</i> A custom list of characters to be trimmed

Description

Removes spaces from the right side of the string and returns the trimmed result.

If a filter is supplied, **any** character in the filter string will be trimmed instead. (To also trim spaces, include a space in the filter string.)

Example

```
var notes = " +[do, re, mi, fa, so, la, ti, do] % ";  
  
notes = string_trim_right(notes, " ]%");  
  
draw_text(x, y, notes);
```



The "array_create_ext" Function

Syntax

```
array_create_ext(size_x, size_y, [size_z], [size_w], ..., value);
```

Argument	Type	Description
size_x	integer	Sets the number of cells in the first dimension of the array
size_y	integer	Sets the number of cells in the second dimension of the array
[size_z]	integer	<i>Optional:</i> Sets the number of cells in the third dimension of the array
[size_w]	integer	<i>Optional:</i> Sets the number of cells in the fourth dimension of the array
...	integer	<i>Optional:</i> Sets the number of cells in any additional dimensions of the array
value	any	Sets a value to assign to all new cells (default 0)

Description

Returns an array of multiple dimensions and assigns a default value to the lowest level cells.

At least two dimensions and a default value must be supplied. Additional arguments will be interpreted as dimension sizes preferentially, with the last argument always being the default value.

Example

```
my_2d_array = array_create_ext(5, 10, 0);  
  
my_3d_array = array_create_ext(5, 10, 3, pi);  
  
my_4d_array = array_create_ext(5, 10, 3, 6, "init");
```



The "array_clear" Function

Syntax

```
array_clear(id);
```

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

Description

Clears an array's data from memory. This also includes any child arrays the input array may hold. To clear a multidimensional array, input the root array. However, it is also possible to clear just one dimension of an array by including any parent arrays in the input value, e.g. `my_array[0][0]`.

Note that this function will not destroy the array itself! To de-reference an array after clearing, simply set the parent variable to 0.

Example

```
array_clear(my_array);  
array_clear(my_array[0][0]);
```



The "array_depth" Function

Syntax

```
array_depth(id);
```

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

Description

Returns the number of dimensions contained in an array. If the input value is not an array, 0 will be returned instead.

Note that array dimensions are not required to have uniform depth. This function returns the **deepest** dimension contained within an array.

Example


```
var my_array = [ 1, 2, 3, [ "a", "b", "c", [ ".", "!", "?" ] ] ];  
  
draw_text(25, 25, string(array_depth(my_array)));
```



The "array_fill" Function

Syntax

```
array_fill(id, [value]);
```

Argument	Type	Description
<code>id</code>	array	The index of a previously-created array to modify
<code>[value]</code>	any	<i>Optional:</i> Sets a value to assign to all new cells (default <code>0</code>)

Description

Finds the longest dimension of a multidimensional array and fills all other dimensions to match the same length, optionally assigning a default value to any new cells created. If a custom value is not specified, `0` will be used by default.

Can be useful for parsing arrays where the parser must assume a certain size for all dimensions of an array. However, note that this function fills array **length** only. Sub-dimensions can still be non-uniform in **depth**, which cannot be solved by this function, as doing so would result in data loss.

Example

```
array_fill(my_array);  
array_fill(my_array, -1);
```



The "array_find_dim" Function

Syntax

```
array_find_dim(id, val);
```

Argument	Type	Description
id	array	The index of a previously-created array to search
val	any	The value to search for

Description

Searches a multidimensional array for the given value and returns the dimension in which it exists, or `-1` if the value is not found in the array. If the input is not an array, but happens to match the search value regardless, `0` will be returned instead.

Example

```
var my_array = [ 1, 2, 3, [ "a", "b", "c", [ ".", "!", "?" ] ] ];  
  
draw_text(25, 25, string(array_find_dim(my_array, "b")));
```



The "array_find_index" Function

Syntax

```
array_find_index(id, val);
```

Argument	Type	Description
<code>id</code>	array	The index of a previously-created array to search
<code>val</code>	any	The value to search for

Description

Searches an array for a value and returns the index, if found, or `-1` if the value does not exist in the array. If the input is not an array, but happens to match the search value regardless, `0` will be returned instead.

To search a multidimensional array, input any parent arrays before the child array to be searched, e.g. `my_array[0][0]`.

Example

```
var my_array = [ 1, 2, 3, [ "a", "b", "c", [ ".", "!", "?" ] ] ];  
  
draw_text(25, 25, string(array_find_index(my_array[3], "b")));
```



The "array_shuffle" Function

Syntax

```
array_shuffle(id);
```

Argument	Type	Description
<code>id</code>	array	The index of a previously-created array to modify

Description

Shuffles the contents of an array, resulting in values being stored in random order.

To shuffle a multidimensional array, input any parent arrays before the child array to be shuffled, e.g. `my_array[0][0]`.

Note that for development builds, GameMaker will use the same random seed, meaning results will always randomize the same way every time the game is restarted. To avoid this behavior, use the built-in `randomize` function to create a new seed.

Example


```
array_shuffle(my_array);  
array_shuffle(my_array[0][0]);
```



The "array_read" Function

Syntax

```
array_read(str);
```

Argument	Type	Description
<code>str</code>	string	A previously-encoded string to decode as an array

Description

An alias of `json_parse`. Converts a string previously created by `array_write` into an array and returns the result.

Note that all array contents will be treated as strings. If the array contains pointers to other types of data, the pointer will be read literally rather than reading the contents of the data itself.

Example

```
var file = file_text_open_read("save.dat");  
my_array = array_read(file_text_read_string(file));  
file_text_close(file);
```



The "array_write" Function

Syntax

```
array_write(id, [pretty]);
```

Argument	Type	Description
<code>id</code>	array	The index of a previously-created array to encode as a string
<code>[pretty]</code>	boolean	<i>Optional:</i> Enables or disables formatting the string with linebreaks and indentation

Description

Converts an array of any dimensions to a string, optionally with "pretty-print" to separate values by line with proper indenting.

Note that all array contents will be treated as strings. If the array contains pointers to other types of data, the pointer will be written literally rather than writing the contents of the data itself. The exception to this rule is sub-arrays and structs, which will be preserved.

Example

```
var file = file_text_open_write("save.dat");  
file_text_write_string(file, array_write(my_array, true));  
file_text_close(file);
```



The "is_empty" Function

Syntax

```
is_empty(val);
```

Argument	Type	Description
<code>val</code>	any	A variable/value to check

Description

Checks if a given value is "empty", which can be `true` or `false` depending on the type of data contained in the input value.

Some examples of "empty" data include:

- `undefined`
- `NaN`
- `false`
- `0`
- `"0"`
- `""`

- []
- {}
- etc.

If the input value points to a data structure, buffer, etc., the structure will be considered empty if no values exist inside the structure itself. Transparent surfaces are also considered empty.

Some types of data cannot be evaluated and will return `false` by default. Also note that different data types incur different performance costs to evaluate.

CAUTION

Note that in some cases this function may not return the expected result due to the way GameMaker handles pointers. This means some types of data can share the same value, and whichever one happens to be first will take priority.

Example

```
var surf = surface_create(1280, 720);
var ds = ds_list_create();

if (is_empty(surf)) {
    surface_copy(surf, 0, 0, application_surface);
}

if (is_empty(ds)) {
    ds_list_add(ds, surf);
}
```



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.