Drag-and-drop Icons to GameMaker Language Reference

For users with previous experience of using GameMaker's drag-and-drop interface, making the leap to pure coding can be quite stressful. Trying to understand what each icon is doing can be confusing, as it is often more than a single function. In order to help users undergo the transition successfully, we have provided a thorough reference of the code equivalent to all the drag-and-drop icons.

Understanding the reference

The purpose of this reference is to help readers who have familiarity with drag-and-drop coding, and want to be able to convert their knowledge into using the GameMaker Language. This reference is not intended to explain how to use the functions, explain all the parameters, or demonstrate how it could be used in a game. GameMaker: Studio comes with thorough help documentation for this purpose.

The reference is set up with three columns, the first with the Drag-and-Drop **Icon**, the second contains the **Options** available, and the third has the **Code Equivalent**. When looking at the Options, we have declared the type of data it is expecting to be input, such as a number. If the Code Equivalent is a variable we set it to the data type. For example, as you can see in the following table, in the Icon column we have **Speed Horizontal**. The only Option for this Action is **Hor speed** which requires a number. The GML Code Equivalent for this is to set a number value for the variable `hspeed`:

Icon and icon name	Options	Code Equivalent
 Speed Horizontal	**Hor speed:** number	`hspeed = number;`

If the Code Equivalent is a function we use the name of the Option as the parameter. For example, as you can see in the following table, we have **Create Instance**, which has Options for the **object**, and the **X** and **Y** positions. In the Code Equivalent for this is the `instance_create` function that has three parameters, for which we use the same name as the options: the `x` and `y` positions, and the `object`.

Icon and icon name	Options	Code Equivalent
 Create Instance	**object**: Object **X**: number **Y**: number	`instance_create( x, y, object );`

If you see other parameters in the function that are not associated with the name of an Option, it means the drag-and-drop icon does not make this adjustable and the parameter is set to its default value. For example, as seen in the following table, **Replace Sprite** has three Options, while the code equivalent has five parameters. The final two parameters are actually for the origin of the sprite, but the default value for these is `0`.

Icon and icon name	Options	Code Equivalent
 Replace Sprite	**sprite**: Sprite **filename**: filename. ext **images**: number	`sprite_ replace(sprite, filename, images, 0, 0);`

Common attributes

Many, though not all, of the drag-and-drop actions have three common attributes: **Applies To**, **Relative**, and **NOT**.

Applies To allows us to assign the executed code to a specific instance. The first option is **Self**, which is the default and does not need any special code to run. The second option is for **Other**, a special built-in variable which is only meaningful for collision events. In a collision event, the **Other** variable is assigned the unique ID of the "other" instance. If not used in a collision event, other will be returned as **no one**, a built-in constant indicating no object. The final option is for **Object**, which will apply to every instance of an object type in existence. If you select a parent object, all children will be affected as well.

The Code Equivalent for all of these is to use a `with` statement as follows:

- Applies to Other:

```
with ( other ) { //GML code goes here }
```

- Applies to Object

```
with ( object name ) { //GML code goes here }
```

Relative allows us to apply a new value to an existing value as opposed to being overwritten by the new value. For example, when scoring points we would want it to be relative to the previous score. For variables, this can be denoted by using `+=` when setting the value. In a function you would want to use the existing parameter and add the new value.

- Relative variable

```
score += 10;
```

- Relative function

```
instance_create(x+8, y+8, object0);
```

NOT is used for questions that require a negative answer, such as if the score is not equal to 100. This is denoted by the use of an exclamation mark.

- NOT variable

```
if (score != 100)
```

- NOT function

```
if (!place_free(x, y))
```

The move tab

The **move** tab contains all the functionalities for moving an instance around the world. This is broken into four subsections: **Move**, **Jump**, **Paths**, and **Steps**.

The Move subsection

The **Move** subsection contains the actions for applying velocity, gravity, and friction to an instance.

Icon and icon name	Options	Code Equivalent

 Move Fixed	**Direction:** selectable arrows
	Speed: number
		Starting from the right arrow going counterclockwise:
		`arrow1 = 0;`
		`arrow2 = 45;`
		`arrow3 = 90;`
		`arrow4 = 135;`
		`arrow5 = 180;`
		`arrow6 = 225;`
		`arrow7 = 270;`
		`arrow8 = 315;`
		Use the selected arrows only:
		`direction = choose(arrow1, arrow2,...);`
		`speed = number;`

 Move Free	**Direction:** number **Speed:** number	`direction = number;` `speed = number;`
 Move Towards	**X:** number **Y:** number **Speed:** number	`move_towards_point( x, y, speed);`
 Speed Horizontal	**Hor speed:** number	`hspeed = number;`
 Speed Vertical	**Vert speed:** number	`vspeed = number;`

Icon and icon name	Options	Code Equivalent
 Reverse Horizontal	None	`hspeed = -hspeed;`
 Reverse Vertical	None	`vspeed = -vspeed;`
 Set Gravity	**Direction:** number **Gravity:** number	`gravity_direction = number;` `gravity = number;`
 Set Friction	**Friction:** number	`friction = number;`

The Jump subsection

The **Jump** subsection has the actions for relocating and redirecting the movement of an instance.

Icon and icon name	Options	Code Equivalent
 Jump to Position	**X:** number **Y:** number	`x = number;` `y = number;`
 Jump to Start	None	`x = xstart;` `y = ystart;`
 Jump to Random	**Snap hor:** number **Snap vert:** number	`move_random(snap hor, snap vert);`

Icon and icon name	Options	Code Equivalent
 Align to Grid	**Snap hor:** number **Snap vert:** number	`move_snap(snap hor, snap vert);`


Wrap Screen
	Direction:
	• **Horizontal**
	• **Vertical**
	• **In both directions**
		Horizontal:
		`move_wrap(true, false, sprite_width / 2);`
		Vertical:
		`move_wrap( false, true, sprite_height / 2);`
		In both directions:
		`move_wrap(true, false, sprite_width / 2);`
		`move_wrap( false, true, sprite_height / 2);`


Move to Contact
	Direction: number
	Maximum: number
	Against:
	• **Solid objects**
	• **All objects**
		Solid objects:
		`move_contact_solid( direction, maximum );`
		All objects:
		`move_contact_all( direction, maximum );`


Bounce
	Precise: (Boolean)
	• **Not precisely**
	• **Precisely**
	Against:
	• **Solid objects**
	• **All objects**
		against solid objects:
		`move_bounce_solid( precise );`
		against all objects:
		`move_bounce_all( precise );`

The Path subsection

The **Path** subsection contains the actions related to using the built-in GameMaker paths.

Icon and icon name	Options	Code Equivalent
 Set Path	**path:** Path **speed:** number **at end:** (0-3) • **stop** • **continue from start** • **continue from here** • **reverse** **relative:** (Boolean) • **relative** • **absolute**	`path_start( path, speed, at end, relative );`
 End Path	None	`path_end();`
 Path Position	**position:** number between 0-1	`path_position = number between 0-1;`
 Path Speed	**speed:** number	`path_speed = number;`

The Step subsection

The **Step** subsection has the actions for advanced path finding.

Icon and icon name	Options	Code Equivalent
 Step Towards	**x:** number **y:** number **speed:** number **stop at:** (Boolean) • **solid only** • **all instances**	`mp_linear_step( x, y, speed, stop at);`
 Step Avoiding	**x:** number **y:** number **speed:** number **avoid:** (Boolean) • **solid only** • **all instances**	`mp_potential_step( x, y, speed, avoid);`

The main1 tab

The **main1** tab contains the most common functionality related to the various game assets. There are four subsections: **Objects**, **Sprite**, **Sounds**, and **Rooms**.

The Objects subsection

The **Objects** subsection contains the actions for creating and destroying instances.

Icon and icon name	Options	Code Equivalent
 Create Instance	**object:** Object **x:** number **y:** number	`instance_create( x, y, object );`

 Create Moving	**object:** Object
	x: number
	y: number
	speed: number
	direction: number
		`obj = instance_create(x , y, object);`
		`obj.speed = number;`
		`obj.direction = number;`

 Create Random	**object 1:** Object **object 2:** Object **object 3:** Object **object 4:** Object **x:** number **y:** number	`obj = choose(object1, object2, object3, object4);` `instance_create(x, y, obj);`
 Change Instance	**change into:** Object **perform events:** (Boolean) **not** **yes**	`instance_change(change into, perform events);`
 Destroy Instance	None	`instance_destroy();`
 Destroy at Position	**x:** number **y:** number	`position_destroy( x, y );`

The Sprite subsection

The **Sprite** subsection has the actions for altering the sprite of an instance.

Icon and icon name	Options	Code Equivalent

 Change Sprite	**sprite:** Sprite
	subimage: number
	speed: number
		`sprite_index = Sprite;`
		`image_index = number;`
		`image_speed = number;`

 Transform Sprite	**xscale:** number
	yscale: number
	angle: number
	mirror:
	• **no mirroring**
	• **mirror horizontal**
	• **flip vertical**
	• **mirror and flip**
		`image_xscale = number;`
		`image_yscale = number;`
		`image_angle = number;`
		Mirror Horizontal:
		`image_xscale *= -1;`
		Flip Vertical:
		`image_yscale *= -1;`
		Mirror and Flip:
		`image_xscale *= -1;`
		`image_yscale *= -1;`

 Color Sprite	**color:** Color **alpha:** number	`image_blend = color;` `image_alpha = number;`

The Sounds subsection

The **Sounds** subsection has the actions related to audio.

Icon and icon name	Options	Code Equivalent

 Play Sound	**sound:** Sound
	loop: (Boolean)
	• **false**
	• **true**
		Legacy Mode:
		`sound_play( sound );`
		loop: `true`
		`sound_loop( sound );`
		New Audio Engine:
		Normal Sound:
		`audio_play_sound( sound, 0, loop );`
		Background Music:
		`audio_play_music( sound, loop );`

Icon and icon name	Options	Code Equivalent

 Stop Sound	**sound:** Sound	Legacy Mode:
		`sound_stop( sound );`
		New Audio Engine:
		Normal Sound:
		`audio_stop_sound( sound );`
		Background Music:
		`audio_stop_music( sound );`

 Check Sound	**sound:** Sound	Legacy Mode:
		`if sound_isplaying( sound );`
		New Audio Engine:
		`audio_is_playing( sound );`

The Rooms subsection

The **Rooms** subsection contains the actions for switching rooms.

Icon and icon name	Options	Code Equivalent
 Previous Room	None	`room_goto_previous();`
 Next Room	None	`room_goto_next();`
 Restart Room	None	`room_restart();`
 Different Room	**room:** Room	`room_goto(room);`

Icon and icon name	Options	Code Equivalent
 Check Previous	None	`if (room_previous(room) != -1)`
 Check Next	None	`if (room_next(room) != -1)`

The main2 tab

The **main2** tab contains the common functionality for dealing with time, showing messages, game, and resource controls. There are four subsections: **Timing**, **Info**, **Game**, and **Resources**.

The Timing subsection

The **Timing** subsection contains the actions for using alarms and timelines.

Icon and icon name	Options	Code Equivalent
 Set Alarm	**number of steps:** number **in alarm no:** 0-11	`alarm[ 0 - 11 ] = number;`

 Set Time Line	**time line:** Time Line
	position: Number
	start: (Boolean)
	• **Start Immediately**
	• **Don't Start**
	loop: (Boolean)
	• **Don't Loop**
	• **Loop**
		`timeline_index = Time Line;`
		`timeline_running = Boolean;`
		`timeline_loop = Boolean;`

 Time Line Position	**position:** number	`timeline_position = number;`
 Time Line Speed	**speed:** number	`timeline_speed = number;`
 Start Time Line	None	`timeline_running = true;`
 Pause Time Line	None	`timeline_running = false;`

Icon and icon name	Options	Code Equivalent
 Stop Time Line	None	`timeline_running = false;` `timeline_position = 0;`

The Info subsection

The **Info** subsection has the actions for displaying messages and opening websites.

Icon and icon name	Options	Code Equivalent
 Display Message	**message:** string	`show_message(string);`
 Open URL	**URL:** URL	`url_open( URL );`

The Game subsection

The **Game** subsection has the actions for restarting and ending games. It also has two obsolete functions that will be removed from the tab in a future version of GameMaker: Studio.

Icon and icon name	Options	Code Equivalent
 Restart Game	None	`game_restart();`
 End Game	None	`game_end();`

Icon and icon name	Options	Code Equivalent
 Save Game	Obsolete	Obsolete
 Load Game	Obsolete	Obsolete

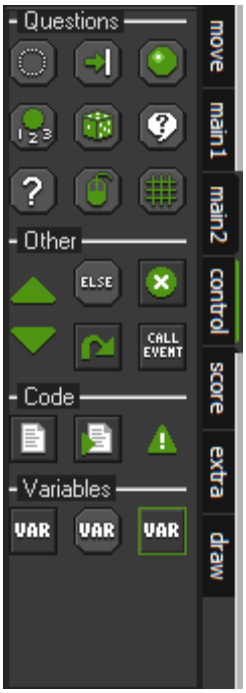
The Resources subsection

The **Resources** subsection has the actions for replacing game assets.

Icon and icon name	Options	Code Equivalent
 Replace Sprite	**sprite:** Sprite **filename:** filename.ext **images:** number	`sprite_replace(sprite, filename, images, 0, 0);`
 Replace Sound	**sound:** Sound **filename:** filename.ext	`sound_replace(sound, filename, 0, 0);`
 Replace Background	**background:** Background **filename:** filename.ext	`background_replace(background, filename);`

The control tab

The **control** tab contains the most common functions related to basic code structure. This is broken into four subsections: **Questions**, **Other**, **Code**, and **Variables**.



The Questions subsection

The **Questions** subsection has the most common conditional statements related to instances in a game.

Icon and icon name	Options	Code Equivalent

 Check Empty	**x:** number
	y: number
	objects:
	• **Only Solid**
	• **All**
		Only Solid:
		`if (place_free(x, y))`
		All:
		`if (place_empty(x, y))`

Icon and icon name	Options	Code Equivalent


Check Collision
	x: number
	y: number
	objects:
	• **Only Solid**
	• **All**
		Only Solid:
		`if (place_free(x, y))`
		All:
		`if (place_empty(x, y))`

 Check Object	**object:** Object **x:** number **y:** number	`if (place_meeting( x, y, object))`


Test Instance Count
	object: Object
	number: number
	operation:
	• **equal to**
	• **smaller than**
	• **larger than**
		equal to:
		`if (instance_number( object ) == number)`
		smaller than:
		`if (instance_number( object ) < number)`
		larger than:
		`if (instance_number( object ) > number)`

 Test Chance	**sides:** number	`if (floor( random( sides )) == 0)`
 Check Question	**question:** string	`if (show_question( question ))`
 Test Expression	**expression:** expression	`if (expression)`
 Check Mouse	**button:** • **no** • **left** • **middle** • **right**	`if (mouse_check_button( button ))`

Icon and icon name	Options	Code Equivalent
 Check Grid	**Snap Hor:** number **Snap Vert:** number	`if (place_snapped(snap hor, snap vert))`

The Other subsection

The **Other** subsection has the actions for writing blocks of code.

Icon and icon name	Options	Code Equivalent
 Start Block	None	{
 End Block	None	}
 Else	None	`else`
 Exit Event	None	`exit;`
 Repeat	**times:** number	`repeat ( times )`
 Call Parent Event	None	`event_inherited();`

The Code subsection

The **Code** subsection has the actions for executing GameMaker Language code.

Icon and icon name	Options	Code Equivalent
 Execute Code	None (opens script editor)	No code alternative as it is a script local to the object
 Execute Script	**script:** Script **argument0:** value **argument1:** value **argument2:** value **argument3:** value **argument4:** value	Any script in the Resource tree can be called by name with `()` at the end. Up to 16 arguments can be passed as parameters.


Comment
	comment: String	to remove a single line of code:
		`//`
		starts a block of comments:
		`/*`
		ends a block of comments:
		`*/`

The Variables subsection

The **Variables** subsection has actions for using variables.

Icon and icon name	Options	Code Equivalent
 Set Variable	**variable:** string **value:** value	`variable = value;`

Icon and icon name	Options	Code Equivalent


Test Variable
	variable: string
	value: value
	operation:
	• **equal to**
	• **less than**
	• **greater than**
	• **less than or equal to**
	• **greater than or equal to**
		equal to:
		`if (variable == value)`
		less than:
		`if (variable < value)`
		greater than:
		`if (variable > value)`
		less than or equal to:
		`if (variable <= value)`
		greater than or equal to:
		`if (variable >= value)`

 Draw Variable	**variable:** string **x:** number **y:** number	`draw_text(x, y, variable);`

The score tab

The **score** tab contains the functionality for setting and drawing of the global game scoring. This is broken into three subsections: **Score**, **Health**, and **Lives**.

The Score subsection

The **Score** subsection has the actions for dealing with the score of the game.

Icon and icon name	Options	Code Equivalent
 Set Score	**new score:** number	`score = number;`

 Test Score	**value:** number
	operation:
	• **equal to**
	• **smaller than**
	• **larger than**
		equal to:
		`if (score == value)`
		smaller than:
		`if (score < value)`
		larger than:
		`if (score > value)`

 Draw Score	**x:** number **y:** number **caption:** string	`draw_text(x ,y, "caption" + score);`
 Clear Highscore	None	`highscore_clear();`

The Lives subsection

The **Lives** subsection has the actions for dealing with the player's lives.

Icon and icon name	Options	Code Equivalent
 Set Lives	**new lives:** number	`lives = number;`

Icon and icon name	Options	Code Equivalent

 Test Lives	**value:** number
	operation:
	• **equal to**
	• **smaller than**
	• **larger than**
		equal to:
		`if (lives == value)`
		smaller than:
		`if (lives < value)`
		larger than:
		`if (lives > value)`

 Draw Lives	**x:** number **y:** number **caption:** string	`draw_text(x ,y, "caption" + lives);`

 Draw Life Images	**x:** number
	y: number
	image: Sprite
		`for ( i = 0; i < lives; i++)`
		`{`
		`slot = i * sprite_get_width(image);`
		`draw_sprite(image, 0, x + slot, y);`
		`}`

The Health subsection

The **Health** subsection has the actions for dealing with the health of the player.

Icon and icon name	Options	Code Equivalent
 Set Health	**value (0-100):** number	`health = value;`

 Test Health	**value:** number
	operation:
	• **equal to**
	• **smaller than**
	• **larger than**
		equal to:
		`if (health == value)`
		smaller than:
		`if (health < value)`
		larger than:
		`if (health > value)`

Icon and icon name	Options	Code Equivalent
 Draw Health	**x1:** number **y1:** number **x2:** number **y2:** number **back color:** Color **bar color:** Color	`draw_healthbar( x1, y1, x2, y2, health, back color, bar color, bar color, 0, true, true);`
 Score Caption	Obsolete	Obsolete

The extra tab

The **extra** tab contains the functionality for particles and changing the appearance of the cursor. It has only two subsections: **Particles** and **Other**.

The Particles subsection

The **Particles** subsection has all the actions for creating and using particle systems, emitters, and particles.

Icon and icon name	Options	Code Equivalent
 Create Part System	**Depth:** number	`system = part_system_create();` `part_system_depth( system, depth);`
 Destroy Part System	None	`part_system_destroy(system);`
 Clear Part System	None	`part_clear_system(system);`

 Create Particle	**type id:** Particle
	shape: Shape
	sprite: Sprite
	min size: number
	max size: number
	size increment: number
		Particle:
		`particle = part_type_create();`
		Shape:
		`part_type_shape(particle, shape);`
		Sprite:
		`part_type_sprite(particle, sprite, true, false, false);`
		Size:
		`part_type_size(particle, min size, max size, size increment, 0);`

Icon and icon name	Options	Code Equivalent

 Particle Color	**type id:** Particle
	Color Mix:
	• **mixed**
	• **changing**
	color1: Color
	color2: Color
	start alpha: number
	end alpha: number
		Mixed color:
		`part_type_color_mix(type id, color1, color2);`
		Changing color:
		`part_type_color2(type id, color1, color2);`
		Start alpha and end alpha:
		`part_type_alpha2(type id, start alpha, end alpha);`

 Particle Life	**type id:** Particle **min life:** number **max life:** number	`part_type_life(type id, min life, max life);`
 Particle Speed	**type id:** Particle **min speed:** number **max speed:** number **min dir:** number **max dir:** number **friction:** number	`part_type_speed(type id, min speed, max speed, -friction, 0);` `part_type_direction(type id, min dir, max dir, 0, 0);`
 Particle Gravity	**type id:** Particle **amount:** number **direction:** number	`part_type_gravity(type id, amount, direction);`
 Particle Secondary	**type id:** Particle **step type:** Particle **step count:** number **death type:** Particle **death count:** number	`part_type_step(type id, step count step type);` `part_type_death(type id, death count, death type);`

Icon and icon name	Options	Code Equivalent
 Create Emitter	**emitter id:** Emitter **shape:** • **rectangle** • **ellipse** • **diamond** • **line** **xmin:** number **xmax:** number **ymin:** number **ymax:** number	`emitter = part_emitter_create(system);` `part_emitter_region(system, emitter, xmin, xmax, ymin, ymax, shape, ps_distr_linear);`
 Destroy Emitter	**emitter id:** Emitter	`part_emitter_destroy(system, emitter id);`
 Burst from Emitter	**emitter id:** Emitter **particle type:** Particle **number:** number	`part_emitter_burst(system, emitter id, particle type, number);`
 Stream from Emitter	**emitter id:** Emitter **particle type:** Particle **number:** number	`part_emitter_stream(system, emitter id, particle type, number);`

The Other subsection

The **Other** subsection has only a single action that sets the cursor to a different Sprite.

Icon and icon name	Options	Code Equivalent

 Set Cursor	**sprite:** Sprite
	cursor:
	• **don't show**
	• **show**
		`cursor_sprite = Sprite;`
		don't show:
		`windows_set_cursor(cr_none);`
		show:
		`windows_set_cursor(cr_default);`

The draw tab

The **draw** tab contains the functionality for drawing graphics, sprites, and text on the screen, as well as the related settings. There are three subsections: **Drawing**, **Settings**, and **Other**.

The Drawing subsection

The Drawing subsection has the actions for drawing sprites, text, and shapes, including using gradients.

Icon and icon name	Options	Code Equivalent
 Draw Self	None	`draw_self( );`
 Draw Sprite	**sprite:** Sprite **x:** number **y:** number **subimage:** number	`draw_sprite(sprite, subimage, x, y);`

Icon and icon name	Options	Code Equivalent


Draw Background
	background: Background
	x: number
	y: number
	tiled: (Boolean)
	• **false**
	• **true**
		`tiled = false`
		`draw_background( background, x, y);`
		`tiled = true`
		`draw_background_tiled( background, x, y);`

 Draw Text	**text:** string **x:** number **y:** number	`draw_text( x, y, text);`
 Draw Scaled Text	**text:** string **x:** number **y:** number **xscale:** number **yscale:** number **angle:** number	`draw_text_transformed( x, y, text, xscale, yscale, angle);`
 Draw Rectangle	**x1:** number **y1:** number **x2:** number **y2:** number **filled:** (Boolean) • **filled** • **outline**	`draw_rectangle( x1, y1, x2, y2, filled);`
 Horizontal Gradient	**x1:** number **y1:** number **x2:** number **y2:** number **color1:** Color **color2:** Color	`draw_rectangle_color( x1, y1, x2, y2, color1, color2, color2, color1, false);`

Icon and icon name	Options	Code Equivalent
 Vertical Gradient	**x1:** number **y1:** number **x2:** number **y2:** number **color1:** Color **color2:** Color	`draw_rectangle_color( x1, y1, x2, y2, color1, color1, color2, color2, false);`
 Draw Ellipse	**x1:** number **y1:** number **x2:** number **y2:** number **filled:** (Boolean) • **filled** • **outline**	`draw_ellipse(x1, y1, x2, y2, filled);`
 Gradient Ellipse	**x1:** number **y1:** number **x2:** number **y2:** number **color1:** Color **color2:** Color	`draw_ellipse_color( x1, y1, x2, y2, color1, color2, false);`
 Draw Line	**x1:** number **y1:** number **x2:** number **y2:** number	`draw_line( x1, y1, x2, y2);`
 Draw Arrow	**x1:** number **y1:** number **x2:** number **y2:** number **tip size:** number	`draw_arrow( x1, y1, x2, y2, tip size);`

The Settings subsection

The **Settings** subsection has the actions for setting the color and fonts to be used, and for switching to full screen.

Icon and icon name	Options	Code Equivalent
 Set Color	**color:** Color	`draw_set_color( color );`
 Set Font	**font:** Font **align:** • **left** • **center** • **right**	`draw_set_font( font );` `draw_set_halign( align );`

 Set Full Screen	**action:**
	• **switch**
	• **window**
	• **fullscreen**
		switch:
		`if (window_get_fullscreen())`
		`{`
		`window_set_fullscreen(false);`
		`}else{`
		`window_set_fullscreen(true);`
		`}`
		window:
		`window_set_fullscreen(false);`
		fullscreen:
		`window_set_fullscreen(true);`

The Other subsection

The **Other** subsection has the actions for taking a screen grab and creating a pre-built particle effect.

Icon and icon name	Options	Code Equivalent
 Take Snapshot	**filename:** filename.ext	`screen_save( filename );`


Create Effect
	type: Effect
	x: number
	y: number
	size: (0, 1, 2)
	• **small**
	• **medium**
	• **large**
	color: Color
	where:
	• **below objects**
	• **above objects**
		below objects:
		`effect_create_below(type, x, y, size, color);`
		above objects:
		`effect_create_above(type, x, y, size, color);`