Welcome to Xtend - The Simplest Display Scaler

Display scalers: everyone needs one, and Xtend is the last one you'll ever need!

Xtend automatically and intelligently resizes your games and applications to fill any window or display, desktop or mobile, with pixel precision and no black bars. It's easy to use and comes pre-configured to suit most projects, while also offering deep customization and powerful camera functions for even the most challenging scaling needs.

Features

Setupless

How to use Xtend? Add it to your project. Done. Seriously. (But you can always customize it if you want to!)

Universal

Xtend supports multiple scaling modes for different needs. No matter your project, there's a mode for you! Such as:

- **Linear** - For desktop-style applications that need 1:1 screen real estate
- **Aspect** - For games and interactive applications that need to fill the screen while preserving a base play area
- **Axis** - For horizontal or vertical splitscreen
- **Pixel** - For retro art styles that demand pixel perfection at any shape and size

All modes also support **DPI scaling** for windowed applications, ensuring consistent physical size across any screen resolution and pixel density.

Optimal

Xtend only runs when there's a change in viewport resolution. Your app's performance is uncompromised!

Powerful

Still not enough? Xtend includes additional functions for managing multiple view cameras, all with their own scaling modes, and drawing regular content relative to display scale. It's the complete package!

Helpful

Take a peek under the hood at any time with built-in debug statistics and hint boxes. You'll see exactly what's being scaled and how! Take it for a spin in the included demo!

In this reference guide, you'll learn...

- Xtend basics, from ~~setup~~ (woops, there is none!) to advanced config parameters
- Display scaling best practices, and how Xtend macros can help
- Individual functions, arguments, and what they mean

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

Version History

1.0.5

- Added auto base DPI
  - Setting a manual DPI base value in `xtend_config` is now optional
- Added mobile DPI scaling support
- Added cutout ("notch") awareness for Android 9+ and iOS 11+ devices
  - Also includes macros for Android 8 and below for use with `display_set_ui_visibility`
- Modifying base dimensions from `xtend_config` in-game now has the same effect as modifying view camera dimensions in-game
- Reduced frequency of DPI warnings and edited debug message for clarity
- Improved automatic view camera setup at runtime
- Improved demo touch support
- Fixed bug causing base DPI to always report as changed, preventing window resizing in some configurations

1.0.4

- HTML5 now auto-disables DPI scaling with a debug warning (DPI scaling is handled by the browser instead)

1.0.3

- Added support for real-time scale enable/disable
- Added `preserve` setting to scale config
  - Optionally keeps the scaled resolution or resets to default when scaling is disabled in real-time
- Added support for real-time DPI scale modifications
  - Incompatible DPI settings for the current display will now show a warning in the debug console
- Added support for real-time min/max aspect ratio modifications
- Performance optimizations

1.0.2

- Added `view_x` and `view_y` macros to track master scale view position in room
- Added `xtend.scale.sample` option as final resolution scale multiplier
- Added fallback in case view camera is inaccessible while still being assigned to a view
  - Solves rare `I32 argument is undefined` errors
- Fixed room viewport autoconfiguration failing in some cases when rooms are not listed in order of index
- Updated GML+ dependencies to latest version (supports GameMaker Studio 2.3.1+)

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.

1.0.5

- **Added cutout ("notch") awareness for Android 9+ and iOS 11+ devices**
  - Android devices will now include cutout areas as part of the overall display resolution, which may result in slight layout changes with the new version of Xtend.
  - iOS devices are unaffected, as the cutout area is always rendered by default.
  - Mobile developers should now use `display_get_bbox_*` functions when drawing UI elements to avoid rendering inside cutouts on relevant Android and iOS devices.
- **Added mobile DPI scaling support**
  - Mobile devices may exhibit different scaling when using default settings. Customizing settings (or disabling DPI scaling) to restore previous behavior may be required.

Xtend Reference Guide

GameMaker Studio renders content at multiple resolutions at different stages in the rendering process. Sometimes they'll all match, but not always. To make matters worse, out of the box, developers only have two choices for scaling their content to different sizes and shapes of display: Aspect Ratio (letterboxing) and Full Scale (stretching). This often creates confusion as developers are disappointed to find their applications distorted and blurred.

In modern applications, it's simply unacceptable to support only one resolution or aspect ratio. But how can an application possibly support every display, from desktop to mobile and beyond?

The answer is: **Xtend**!

Xtend is *setupless*, meaning all you have to do is import it to your project, and Xtend itself will handle the rest. It is self-instantiating and ships with a default config that will be suitable for the vast majority of applications out of the box. Base resolution will be detected based on current project settings, and with the default `aspect` scaling mode, you'll never see black bars or cropping again. The viewport will be automatically extended either horizontally or vertically to fill the available space *by aspect ratio only*, minimizing differences in viewport resolution to preserve the look

and feel of your content. Only the final application surface will be rendered 1:1 with the display to provide the sharpest possible output image.

But that's only scratching the surface. For best results, you should always follow certain best practices when **designing** content for scaling. And for projects with unique scaling needs, Xtend offers deep config customization and a variety of useful functions to make your application look perfect.

In this reference guide, we'll examine all this and more in detail.

Display Scaling Best Practices

Peeling the Onion

GameMaker Studio renders content at multiple resolutions at different stages in the rendering process. Sometimes they'll all match, but not always. To understand display scaling, it's important to understand the various resolutions at which your project operates.

It's helpful to think of each resolution as a layer, all stacked together to form the final image:

- At the lowest level is the **room**, where all your objects and sprites are drawn. The room is projected onto a **view** (or **view camera**) which represents the currently visible area (or, in keeping with the analogy, a small 'slice' of the bigger room).
- Next, the view camera is projected onto a **port** (or **viewport**) which determines where the camera appears on the screen. Typically, the viewport will fill the entire window or display, but it is not required to. In fact, it's not required to match the view camera OR the screen in any way, and can be as big or small or wide or tall as it pleases, potentially resulting in stretching or squishing the view camera's pixels beyond their original resolution.
- Next, the viewport is projected onto the **application surface**, which is the final step in the primary rendering process. All viewports are calculated and displayed on a single flat texture which always fills the output window

(according to the scaling mode set in the project's Game Settings). However, like the viewport before it, the application surface has its own independent resolution and is not required to match the viewport OR the window. It is simply a way to collect all previous layers into one.

- Next, there is the **GUI layer**. GameMaker provides this separate layer for drawing UI and HUD elements that don't belong in the room itself. And--you guessed it--it also has its own independent resolution. Like the application surface, it is nearly always forced to always fill the output window (according to the scaling mode set in the project's Game Settings), potentially resulting in stretching or squishing. While it technically *can* be moved and scaled (similar to a view) this is generally not advisable. By default, the GUI layer will match resolution with either the application surface or the game window, whichever GameMaker calculates is more preferable.
- Finally, we have the output **window** (or **display**--but because an application is still technically in a window even when running in fullscreen, 'window' should always be assumed). The window will frame content in one of two ways, determined by the scaling mode set in the project's Game Settings: **Aspect Ratio**, or **Full Scale**. Aspect Ratio scaling will resolve any differences between the window itself and the application surface/GUI layer by adding black bars on either side of the image, while Full Scale will simply stretch the application surface/GUI layer to fill the window.

So, then, a GameMaker Studio application does not run at one resolution, but potentially five different resolutions all at once!

Changing resolutions causes any surfaces (including the application surface) to break and therefore require re-rendering. For custom surfaces containing dynamically-generated content, this can result in lost visual information. To avoid this problem while scaling, see `draw_get_surface` from GML+!

A Display Scaler's Role

It's important to note that Xtend does not replace this rendering process, but rather *manages* it intelligently. In addition, Xtend cannot override the built-in scaling method set in the project's Game Settings. While both available methods will produce the same results a majority of the time with a good display scaler, there are some cases where Aspect Ratio is preferred, so this setting should always be enabled when using Xtend.

> ⓘ NOTE
>
> This shouldn't be confused with Xtend's own `aspect` scaling mode, as all Xtend scaling modes are subject to GameMaker's built-in scaling.

While one might assume that the goal of display scaling is to match application resolution with window resolution, this is no longer true in modern applications. In fact, with Xtend, only the application surface and GUI layer will match window resolution 1:1. This ensures that UI and HUD elements can appear at full resolution regardless of internal resolution, while internal contents are not skewed by mismatches between the application surface and window.

For everything else, the goal is twofold:

1. Preserve square pixels
2. Match view *shape* proportionately to eliminate black bars

Unfortunately, it is not always possible to achieve the first goal perfectly, but it *is* possible to get close enough that any discrepencies are invisible on modern high-resolution displays.

TIP

For low-resolution art styles, see Xtend's `pixel` scaling mode, which simulates integer scaling for true pixel perfection

A Designer's Role

Content that scales elegantly to different shapes of display is called **Responsive Design**. First popularized by websites that must work equally well on both desktop and mobile devices, the same principles now apply to a broad variety of applications. While no specific design changes are required to use Xtend, for best results, it's important to foster a few particular design habits in your programming:

Percentages, Percentages, Percentages

First, the #1 rule of responsive design is to ditch the concept of pixels as a unit of measurement. Instead, all coordinates and dimensions must be thought of as percentages of the output viewport ***or*** other key visual elements (which are in turn scaled relative to the viewport).

This can be expressed in different ways, but a common approach is to calculate a pixel value by simply multiplying the primary viewport dimensions by a fraction. Xtend includes built-in macros for this, aptly named `view_width` and `view_height`, where, for example, `view_width*0.5` (or 50%) would be center. GameMaker Studio itself also includes functions for getting the dimensions of other viewports (`camera_get_view_width(view_camera[1])`), layers (`display_get_gui_width()`), and elements (`sprite_get_width(my_sprite)`).

Calculating pixel values as percentages of other pixel values ensures elements retain the same visual placement at any resolution.

Two Layouts are Better than One

What makes these values especially powerful is that they apply to both **position** and **scale**. Designing relative to percentages will keep your layout consistent at different sizes, and also resize individual elements within that layout appropriately for the available space.

By checking for different resolution or aspect ratio thresholds, you can even trigger entirely different layouts suitable for different viewport shapes. For example, `if (view_aspect > 1)` will return `true` in landscape orientation and `false` in portrait. `if (view_width < 1280)` will determine whether the current resolution is sub-HD, at which point a larger font may be required for text elements.

Scale can also play a useful role as a multiplier of base resolution. Xtend includes built-in macros for `view_xscale` and `view_yscale` which can be applied to all sorts of elements to ensure they occupy a suitable area of the display at any resolution. This includes both functions which explicitly define scale (e.g. `draw_sprite_ext`) and those which define dimensions as a value of pixels (e.g. `draw_sprite_transformed`). Either supply the scale value itself, or multiply the base dimensions by the scale value to produce flexible content.

TIP

To scale groups of elements as a whole, see `instance_link` from GML+!

Keep it Centered

A rule of thumb: shift your design paradigm southeast by 50%. Computers in general like to draw things from the top left corner to the bottom right, but that doesn't mean your design should work the same way. In many cases, you'll want to base your content around the *center* of the viewport rather than a corner. Xtend also includes built-in macros for `view_xcenter` and `view_ycenter` to provide an easy origin point. Of course, what matters is that these values will *always* be center regardless of window shape. You can then add or subtract to position content from there.

This rule is not universal, but critical where it matters. Nothing will break faster when changing display scale than manually centered content designed for a fixed resolution.

TIP

You can also center viewports with Xtend! See `camera_set_view_halign` and `camera_set_view_valign`!

... Or Don't

With these concepts in mind, it's also important to know where responsive design *doesn't* apply. Remember, all five layers in the rendering 'onion' exist to provide a window into your game world. That world is still ultimately comprised of pixels, and may or may not benefit from changing itself relative to the viewer.

While puzzle games and visual novels are integrated directly into the user interface, platformers and action games primarily exist in a separate space. In the end, only you, the designer, can decide which elements require responsive design *and which don't* to provide the best experience.

Of course, even the best design has its limits. To protect against the extremes, Xtend's config allows setting a **minimum** and **maximum aspect ratio**, beyond which the application will letterbox or pillarbox. The result still takes advantage of the available space as much as is reasonable, but without breaking design. After all, there's no shame in creating a good design that works in *some* scenarios over winding up with a design too broad to feel great in any scenario.

Target whatever range of design you can and let Xtend take care of the rest! To learn how, continue on to the rest of this reference guide.

Configuring Xtend

Xtend is **setupless**, meaning all you have to do is add it to your project, and you're good to go. But of course, you can always customize it too!

Upon importing Xtend, you'll see an `XGASOFT > Xtend` folder added to your asset browser. At the root of this folder is a file called `xtend_config`. You may freely edit this file to change the default configuration or even add new entries for your own reference.

Xtend's config file uses a `struct` format similar to JSON. Settings are grouped into categories under the root, which is simply called `xtend`. Any values changed in this file will be applied at runtime, but they can also be accessed or changed later by chaining categories and properties together:

```
xtend.scale.enabled = true;
```

Some properties are optional, and are commented out by default. Simply uncomment any properties you wish to modify, and your custom values will be applied.

```
debug: {
    ...
    stats_enabled: false,   <-- This value is active
    //stats_color: c_lime,  <-- Remove "//" to activate this value
    ...
}
```

> ⚠ CAUTION
>
> Any properties not commented out by default are required and cannot be removed.

For details on each property, see the sections covering each category below. Categories are ordered logically rather than alphabetically.

`win`

The **window** category sets the foundation for scaling behaviors. Of primary importance is the **base resolution**, which is the unscaled resolution your game design should *assume*. Scaling will be applied from the base resolution to whatever window it may be displayed on, so choose this value carefully. There is no right or wrong resolution; only the appropriate resolution for your game design and art style.

Many properties here are optional, and will be detected automatically based on room settings. However, **getting these right is important**, so it is recommended to set them manually rather than rely on Xtend's best guesses. Only you can decide what suits your project!

```
win: {
    ...
    width_base: 1920,
    height_base: 1080,
    ...
},
```

Base resolution determines the initial resolution of your game, after which scaling will be applied. If at first scaling isn't working as expected, start here.

> TIP
>
> Viewports managed by Xtend can still be moved and resized freely. If you resize an active view camera later, the resized dimensions will become the new base resolution.

Of close secondary importance is **DPI scaling**. For high-resolution displays, DPI scaling will magnify content to present the application at the same physical size as on lower-resolution displays. It is generally recommended to enable DPI scaling, especially on desktop platforms.

How this magnification is applied depends on the **base DPI**. This value is technically arbitrary, but in general, a base DPI of 96 or 160 is the accepted standard by desktop and mobile operating systems, respectively. (By default, Xtend will use a custom DPI setting for each major platform.)

```
win: {
    ...
    dpi_enabled: true,
    dpi_base: 96,
    ...
},
```

DPI scaling will be calculated as a multiplier of base DPI vs actual DPI. Decreasing the base DPI will result in *higher* magnification. On desktop platforms, this applies to window size only (scaling the window *up*), while on mobile platforms, this applies to the main view camera (scaling the viewport *down*).

> ⓘ NOTE
>
> If the magnified window is larger than the physical display, DPI scaling will be disabled automatically. It can be enabled again by running `window_set_dpi` or setting `xtend.win.dpi_enabled` to `true`.

Finally, Xtend allows setting **minimum** and **maximum aspect ratios** to prevent extreme window shapes from breaking your design. This is most easily expressed as `width/height`. If you are familiar with aspect ratios such as 4:3, 16:9, or 21:9, these can be written as `4/3`, `16/9`, and `21/9`, respectively. For portrait orientation, simply reverse the order: `3/4`, `9/16`, or `9/21`, for example. If either aspect ratio is set to `0`, no limit will be applied (this is the default behavior).

```
win: {
    ...
    aspect_min: 4/3,
    aspect_max: 21/9,
    ...
},
```

> ⓘ NOTE
>
> These values are just examples, and may represent too narrow a range for many users. You should always try to accommodate the broadest range of ratios possible!

`scale`

With your window behaviors defined, you can also customize how content is scaled to fit within. Xtend provides 5 unique **scaling modes** to suit different types of content:

- `linear` : Crops or extends both dimensions to match available screen area on a 1:1 basis. Ideal for desktop-style applications.
- `aspect` : Preserves the base area of the view (width AND height) and extends the longer axis to fill the screen. Never crops. (Recommended, enabled by default.)
- `axis_x` : Preserves height at the cost of either cropping or extending width to fill the screen. Ideal for horizontal splitscreen.
- `axis_y` : Preserves width at the cost of either cropping or extending height to fill the screen. Ideal for vertical splitscreen.
- `pixel` : For retro art styles with very low resolution. Finds the nearest integer scale on larger resolutions and crops or extends width and height only as necessary to fill aspect ratio. Square pixels are preserved at all costs.

```
scale: {
    ...
    mode: aspect,
    ...
},
```

The chosen scaling mode will be applied to a **view camera**. GameMaker Studio 2 features a camera system that allows for any number of cameras to exist at the same time, but only 8 can be applied to active views at once (for a maximum of 8-way splitscreen). These views are numbered 0-7, where one must be dedicated exclusively to Xtend as the 'master scaling view'. The chosen view will be configured

automatically, but be warned that it will also override any manual settings, so it's best to choose a view not currently occupied for other purposes.

```
scale: {
    ...
    view: 0,
    ...
},
```

TIP

Additional views can also be configured in `xtend_config`. See the `view#` category for details.

GameMaker Studio also features a separate **GUI layer** independent of the view camera system. In most cases, it is preferable to allow Xtend to automatically scale this layer as well, but this is not required. Xtend will scale the GUI to match window resolution 1:1 rather than view camera resolution. For other behaviors, you may disable Xtend GUI scaling to take control of the GUI layer yourself.

```
scale: {
    ...
    gui: true,
    ...
},
```

In some cases, scaling may cause visible pixellation which is undesirable for certain art styles. This can be mitigated with **texture filtering** to smooth out scaling results with linear interpolation. However, it is not universally suitable and is disabled by default.

```
    scale: {
        ...
        filter: true,
        ...
    },
```

In other cases, certain art styles may actually *benefit* from additional pixellation, or low-end devices that can't render a project at native resolution. By default, the application surface will render 1:1 with the scaled resolution, but this can be adjusted with the **sample** multiplier. This will increase or decrease the rendered resolution of the application surface without affecting viewports or the GUI layer. Multiples of 2 will produce best results.

```
    scale: {
        ...
        sample: 0.5,
        ...
    }
```

Sample size is capped to acceptable minimum and maximum texture resolutions as defined by GameMaker Studio export modules. Multiplier range may vary in real-time depending on scaling. Any invalid inputs will be automatically adjusted.

Finally, Xtend provides a way to **disable scaling** without having to remove it from your project. This can be useful for testing purposes or even for providing users the option of a fixed aspect ratio if they prefer. However, it's important to note that this setting disables scaling, but *it does not disable Xtend*! Other features, like Xtend's built-in debug mode, will remain accessible regardless.

```
scale: {
    ...
    enabled: false,
    ...
},
```

By default, disabling scaling in real-time will cause the viewport to reset to its original size, potentially introducing letterboxing/pillarboxing. However, if you'd prefer to simply "freeze" the current scale, it is possible to **preserve** it instead.

```
scale: {
    ...
    preserve: true,
    ...
}
```

debug

Xtend features a built-in debug mode for displaying scaling **statistics** and **hint boxes**. Hint boxes highlight the area where content has been extended or cropped to provide context for base vs scaled dimensions. Stats and hints can be enabled or disabled independently to avoid cluttering the display with too much information.

```
debug: {
    ...
    hints_enabled: true,
    stats_enabled: true,
    ...
},
```

CAUTION

Enabling debug features will incur a significant performance impact and should be disabled when profiling your application.

Because debug information can be difficult to see depending on what's in the background, stats and hints can also be given custom **colors**, and hints **alpha** can be customized for more or less contrast with background content. Hints alpha has a range from 0-1, while colors can be provided in any GameMaker format. Note that the chosen hints color will be inverted to show negative space from cropped areas.

```
debug: {
    ...
    hints_color: c_aqua,
    hints_alpha: 0.5,
    stats_color: c_lime,
    ...
},
```

TIP

For hex notation support, see `make_color_hex` from GML+!

NOTE

Debug stats use a range of font sizes to accommodate different resolutions. Each size is stored as a separate bitmap font asset, named `fnt_debug_###`, where `###` is a multiple of 50 (relating to DPI scale). Included sizes cover the spectrum from 360p to 4K, but additional fonts can be supplied for other resolutions by adding a new font asset with the desired DPI scale value in the name.

If an exact match can't be found for the current resolution, the nearest applicable font size will be used instead.

`view#`

GameMaker Studio supports up to 8 active views, numbered 0-7, where one must be dedicated exclusively to Xtend as the 'master scaling view'. However, additional views can also be defined in `xtend_config` for preconfigured splitscreen. This is not required, as additional views must always be scaled later with `camera_set_view_scale` to reflect any changes in real-time, but some may find `xtend_config` a more convenient interface for the initial setup.

As soon as any view is scaled by Xtend, configuration entries will be added for it in the running application. (This includes the master scaling view.) Each view has its own category, named `view#`, where `#` is the view number (e.g. `view1`), and requires four properties to be valid:

```
view1: {
    width_base: 1920,
    height_base: 1080,
    halign: va_left,
    valign: va_top,
},
```

In the case of the master view, `width_base` and `height_base` will always equal the values defined in the `win` category. Any manual changes will be overwritten.

For other views, these values determine the initial size of the view camera--or put differently, the area of the room visible to the camera. Where the camera appears *on the screen* is determined by the viewport settings defined in `camera_set_view_scale`. If no configuration exists for the view when it is first scaled, size will be determined automatically. You can also use the GameMaker Studio room editor, or run `camera_set_view_size` before `camera_set_view_scale` to have the same effect as defining a size in `xtend_config`.

Also note that unlike the master view, additional views are not forced visible by default, and must be enabled manually either in the GameMaker Studio room editor or by setting `view_visible[#] = true;` in code executed within the relevant room itself.

Xtend is compatible with most built-in GameMaker view camera functions. You can freely resize, move, zoom, and even rotate view cameras, and Xtend will automatically update itself to reflect the manual changes.

Additionally, each view managed by Xtend supports **horizontal** and **vertical alignment**. These values determine the 'anchor point' or 'attachment' of the view camera when it is resized. When this occurs, the camera position will be offset to whichever side is indicated by `halign` and `valign`.

Possible values include:

- `halign`
  - `va_left`
  - `va_center`
  - `va_right`
- `valign`
  - `va_top`
  - `va_middle`
  - `va_bottom`

While the default `va_left`, `va_top` alignment is preferable in the vast majority of cases, some applications may benefit from preserving a particular focal point during rescale operations. This requires thinking about your design from a particular origin point, however, and may not produce the results you expect without the proper design approach.

View alignment can always be determined later with `camera_get_view_halign` / `camera_get_view_valign` and `camera_set_view_halign` / `camera_set_view_valign`.

Macros & Keywords

For the sake of memorability and forwards-compatibility, Xtend provides built-in keywords and custom macros to aid in developing projects with responsive design. It is strongly recommended to always use keywords and macros when available, as their literal values may change in future updates.

Note that most macros are **read-only**, and should be used for reference, not for modification. For example, to change the position of the master scaling view within the room, `view_x = 50;` is **invalid**. Instead, use regular built-in GML functions for modifying view camera properties, e.g. `camera_set_view_pos(view_camera[0], 50, 0);`.

TIP

Many macros are prefixed, meaning you don't have to memorize each one to use them. Simply begin typing the first few letters of a macro or keyword in your code editor and you'll be shown auto-complete options to choose the desired item from a list.

Viewport Properties

Keyword/Macro	Value	Description
`view_x`	`camera_get_view_x(view_camera[xtend.scale.view])`	Returns the room X coordinate of the master scaling view, in pixels
`view_y`	`camera_get_view_y(view_camera[xtend.scale.view])`	Returns the room Y coordinate of the master scaling view, in pixels
`view_width`	`camera_get_view_width(view_camera[xtend.scale.view])`	Returns the width of the master scaling view, in pixels
`view_height`	`camera_get_view_height(view_camera[xtend.scale.view])`	Returns the height of the master scaling view, in pixels

Keyword/Macro	Value	Description
`view_xcenter`	`(view_width*0.5)`	Returns half the width of the master scaling view, in pixels (does not include room X coord)
`view_ycenter`	`(view_height*0.5)`	Returns half the height of the master scaling view, in pixels (does not include room Y coord)
`view_xscale`	`(view_width/xtend.win.width_base)`	Returns the relative horizontal scale of the master scaling view as compared to base width, as a multplier

Keyword/Macro	Value	Description
`view_yscale`	`(view_height/xtend.win.height_base)`	Returns the relative vertical scale of the master scaling view as compared to base height, as a multplier
`view_aspect`	`(view_width/view_height)`	Returns the aspect ratio of the master scaling view, as a fraction

Android System Flags

As of version 1.0.5, Xtend supports rendering cutout areas in Android 9+. While Android 8 and below do not support cutouts, they do handle the system UI in different ways depending on manufacturer. This can include showing or hiding different system UI elements, which cut into an app's rendering space. To explicitly set system UI behavior, GameMaker Studio includes the function `display_set_ui_visibility`, but does not provide the flags needed to actually use it. Xtend solves this problem by including predefined flags consistent with the Android SDK.

Keyword/Macro	Value	Description
`SYSTEM_UI_FLAG_VISIBLE`	0	View has requested the system UI (status bar) to be visible (the default).
`SYSTEM_UI_FLAG_LOW_PROFILE`	1	View has requested the system UI to enter an unobtrusive "low profile" mode. Status bar and/or navigation icons may dim.
`SYSTEM_UI_FLAG_HIDE_NAVIGATION`	2	View has requested that the system navigation (Home, Back, and the like) be temporarily hidden.
`SYSTEM_UI_FLAG_FULLSCREEN`	4	View has requested to go into the normal fullscreen mode so that its content can take over the screen while still allowing the user to interact with the application.
`SYSTEM_UI_FLAG_LIGHT_NAVIGATION_BAR`	16	Requests the navigation bar to draw in a mode that is compatible with light navigation bar backgrounds. For this to take effect, the window must request `FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS` but not `FLAG_TRANSLUCENT_NAVIGATION`.
`SYSTEM_UI_FLAG_LAYOUT_STABLE`	256	Requests content insets to always represent the worst case that the application can expect as a continuous state (depending on other active flags).
`SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION`	512	View would like its window to be laid out as if it has requested `SYSTEM_UI_FLAG_HIDE_NAVIGATION`, even if it currently hasn't.

Keyword/Macro	Value	Description
`SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN`	1024	View would like its window to be laid out as if it has requested SYSTEM_UI_FLAG_FULLSCREEN, even if it currently hasn't.
`SYSTEM_UI_FLAG_IMMERSIVE`	2048	View would like to remain interactive when hiding the navigation bar with `SYSTEM_UI_FLAG_HIDE_NAVIGATION`
`SYSTEM_UI_FLAG_IMMERSIVE_STICKY`	4096	View would like to remain interactive when hiding the status bar with `SYSTEM_UI_FLAG_FULLSCREEN` and/or hiding the navigation bar with `SYSTEM_UI_FLAG_HIDE_NAVIGATION`. Use this flag to create an immersive experience while also hiding the system bars.
`SYSTEM_UI_FLAG_LIGHT_STATUS_BAR`	8192	Requests the status bar to draw in a mode that is compatible with light status bar backgrounds. For this to take effect, the window must request `FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS` but not `FLAG_TRANSLUCENT_STATUS`.

TIP

Multiple flags can be assigned in `display_set_ui_visibility` at once by separating values with bitwise OR, aka the pipe symbol (`|`)

The "window_set_dpi" Function

Syntax

```
window_set_dpi([dpi]);
```

Argument	Type	Description
`[dpi]`	real	*Optional:* New override (not base) DPI value to set

Description

Applies the current DPI scale to the window, if DPI scaling is enabled. Can also be used to override the base DPI setting if a new DPI value is provided. If DPI scaling is disabled, base DPI will be applied instead regardless of override.

Use with caution! DPI scaling is managed by Xtend, so this function should generally not be run manually. Consider assigning a new base DPI value in `xtend_config` instead.

Example

```
window_set_dpi();

window_set_dpi(128);
```

The "window_get_dpi" Function

Syntax

```
window_get_dpi();
```

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

Description

Returns the current window DPI as a **scale multiplier** of the base DPI setting. Result depends on the running platform. Desktop and mobile DPI multipliers are inverted, such that desktop platforms will generally scale up (values > 1.0), while mobile platforms will generally scale down (values < 1.0).

Example

```
if (os_type == os_windows) {
    if (window_get_dpi() < 1) {
        draw_set_font(font_small);
    } else {
        draw_set_font(font_large);
    }
}
```

The "camera_set_view_scale" Function

Syntax

```
camera_set_view_scale(camera, mode, xport, yport, wport, hport,
[force]);
```

Argument	Type	Description
`camera`	camera	The camera index to modify (typically `view_camera[#]`)
`mode`	constant	The scaling mode to use (same as config)
`xport`	real	The horizontal position to display the camera within the master view
`yport`	real	The vertical position to display the camera within the master view
`wport`	real	The horizontal area to display the camera within the master view
`hport`	real	The vertical area to display the camera within the master view
`[force]`	boolean	*Optional:* Perform scaling even if a resize was not detected

Description

While the master view defined in config will always fill the entire window, additional views can be created for splitscreen display within the master view. This script applies the same scaling behaviors to splitscreen views as are applied to the master view.

GameMaker currently provides 8 built-in view cameras, numbered 0-7, although custom cameras can be assigned to the view as well. To use a built-in camera, input

`camera` as `view_camera[#]`. `view_visible[#]` must be `true` for the viewport to be displayed.

Note that if the master view is input in this script, it will be ignored.

It is recommended to use the built-in `view_width` and `view_height` macros to position and size the viewport relative to the master view. If this script is run in a Step event, inner views will respond to changes in window size with correct scaling. For other events, enabling the optional `[force]` argument will trigger scaling immediately.

By default, base width/height will be determined by the size of the camera when this script is first run. This can be set with `camera_set_view_size` or by defining `width_base` and `height_base` within a `view#` section in config.

Example

```
camera_set_view_scale(view_camera[1], axis_x, 0, 0, view_width*0.5,
view_height);
```

The "camera_get_view_xscale" Function

Syntax

```
camera_get_view_xscale(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the horizontal scale multipler of a view camera scaled with `camera_set_view_scale`.

If the input view has not been scaled, it will return a value of 1. If the input view does not exist, it will return 0 instead.

Example

```
image_xscale = camera_get_view_xscale(view_camera[1]);
```

The "camera_get_view_yscale" Function

Syntax

```
camera_get_view_yscale(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the vertical scale multipler of a view camera scaled with `camera_set_view_scale`.

If the input view has not been scaled, it will return a value of 1. If the input view does not exist, it will return 0 instead.

Example

```
image_yscale = camera_get_view_yscale(view_camera[1]);
```

The "camera_set_view_halign" Function

Syntax

```
camera_set_view_halign(camera, halign);
```

Argument	Type	Description
`camera`	camera	The camera index to modify (typically `view_camera[#]`)
`halign`	constant	The alignment to set (`va_left` , `va_center` , or `va_right`)

Description

Sets the horizontal alignment of a camera, if it is currently assigned to a view. Alignment determines which side(s) of the view are expanded or cropped following a resize operation. Possible values are `va_left` , `va_center` , or `va_top` (`va_left` by default).

Example

```
camera_set_view_halign(view_camera[1], va_center);
```

The "camera_get_view_halign" Function

Syntax

```
camera_get_view_halign(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the horizontal alignment of the input camera, if it is currently assigned to a view. Possible values are `va_left`, `va_center`, `va_top`. If the camera is not assigned to a view, `va_left` will be returned.

Example

```
var halign = camera_get_view_halign(view_camera[1]);

draw_set_halign(halign);
```

The "camera_set_view_valign" Function

Syntax

```
camera_set_view_valign(camera, valign);
```

Argument	Type	Description
`camera`	camera	The camera index to modify (typically `view_camera[#]`)
`valign`	constant	The alignment to set (`va_top` , `va_middle` , or `va_bottom`)

Description

Sets the vertical alignment of a camera, if it is currently assigned to a view. Alignment determines which side(s) of the view are expanded or cropped following a resize operation. Possible values are `va_top` , `va_middle` , or `va_bottom` (`va_top` by default).

Example

```
camera_set_view_valign(view_camera[1], va_middle);
```

The "camera_get_view_valign" Function

Syntax

```
camera_get_view_valign(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the vertical alignment of the input camera, if it is currently assigned to a view. Possible values are `va_top`, `va_middle`, `va_bottom`. If the camera is not assigned to a view, `va_top` will be returned.

Example

```
var valign = camera_get_view_valign(view_camera[1]);

draw_set_valign(valign);
```

The "camera_get_view_bbox_left" Function

Syntax

```
camera_get_view_bbox_left(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the width in pixels of the left bounding box (or pillarbox) of a camera, if it is currently assigned to a view. This value is relative to the base width defined in config, and is useful for managing non-scaled, centered content within a scaled viewport. If the camera is not assigned to a view, 0 will be returned instead.

Note that this does not measure space outside the view when min/max aspect ratios are employed, but rather the space inside a scaled view relative to an unscaled view.

Example

```
draw_rectangle(0, 0, camera_get_view_bbox_left(), view_height, false);
```

The "camera_get_view_bbox_right" Function

Syntax

```
camera_get_view_bbox_right(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the width in pixels of the right bounding box (or pillarbox) of a camera, if it is currently assigned to a view. This value is relative to the base width defined in config, and is useful for managing non-scaled, centered content within a scaled viewport. If the camera is not assigned to a view, 0 will be returned instead.

Note that this does not measure space outside the view when min/max aspect ratios are employed, but rather the space inside a scaled view relative to an unscaled view.

Example

```
draw_rectangle(camera_get_view_bbox_right(), 0, view_width,
view_height, false);
```

The "camera_get_view_bbox_top" Function

Syntax

```
camera_get_view_bbox_top(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the height in pixels of the top bounding box (or letterbox) of a camera, if it is currently assigned to a view. This value is relative to the base height defined in config, and is useful for managing non-scaled, centered content within a scaled viewport. If the camera is not assigned to a view, 0 will be returned instead.

Note that this does not measure space outside the view when min/max aspect ratios are employed, but rather the space inside a scaled view relative to an unscaled view.

Example

```
draw_rectangle(0, 0, view_width, camera_get_view_bbox_top(), false);
```

The "camera_get_view_bbox_bottom" Function

Syntax

```
camera_get_view_bbox_bottom(camera);
```

Argument	Type	Description
`camera`	camera	The camera index to check (typically `view_camera[#]`)

Description

Returns the height in pixels of the bottom bounding box (or letterbox) of a camera, if it is currently assigned to a view. This value is relative to the base height defined in config, and is useful for managing non-scaled, centered content within a scaled viewport. If the camera is not assigned to a view, 0 will be returned instead.

Note that this does not measure space outside the view when min/max aspect ratios are employed, but rather the space inside a scaled view relative to an unscaled view.

Example

```
draw_rectangle(0, camera_get_view_bbox_bottom(), view_width,
view_height, false);
```

The "display_get_bbox_left" Function

Syntax

```
display_get_bbox_left();
```

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

Description

Android/iOS only. Returns the width in pixels of the left bounding box (or pillarbox) of a display with one or more cutouts (aka "notch" or "hole-punch"). Will return 0 on other displays.

Note that bounding box will change depending on device orientation. This function will **always** return the value for the left side of the display relative to the user's perspective.

Example

```
var str = "Hello, world!";
var sx = display_get_bbox_left();
var sy = 50;

draw_text(sx, sy, str);
```

The "display_get_bbox_right" Function

Syntax

```
display_get_bbox_right();
```

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

Description

Android/iOS only. Returns the width in pixels of the right bounding box (or pillarbox) of a display with one or more cutouts (aka "notch" or "hole-punch"). Will return 0 on other displays.

Note that bounding box will change depending on device orientation. This function will **always** return the value for the right side of the display relative to the user's perspective.

Example

```
var str = "Hello, world!";
var sx = display_get_width() - display_get_bbox_right() -
string_width(str);
var sy = 50;

draw_text(sx, sy, str);
```

The "display_get_bbox_top" Function

Syntax

```
display_get_bbox_top();
```

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

Description

Android/iOS only. Returns the height in pixels of the top bounding box (or letterbox) of a display with one or more cutouts (aka "notch" or "hole-punch"). Will return 0 on other displays.

Note that bounding box will change depending on device orientation. This function will **always** return the value for the top side of the display relative to the user's perspective.

Example

```
var str = "Hello, world!";
var sx = 50;
var sy = display_get_bbox_top();

draw_text(sx, sy, str);
```

The "display_get_bbox_bottom" Function

Syntax

```
display_get_bbox_bottom();
```

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

Description

Android/iOS only. Returns the height in pixels of the bottom bounding box (or letterbox) of a display with one or more cutouts (aka "notch" or "hole-punch"). Will return 0 on other displays.

Note that bounding box will change depending on device orientation. This function will **always** return the value for the bottom side of the display relative to the user's perspective.

Example

```
var str = "Hello, world!";
var sx = 50;
var sy = display_get_height() - display_get_bbox_bottom() -
string_height(str);

draw_text(sx, sy, str);
```

The "draw_self_scaled" Function

Syntax

```
draw_self_scaled(ratio, min, max);
```

Argument	Type	Description
`ratio`	`real`	Percentage of the view to occupy, as a value ranging from 0-1
`min`	`real`	Minimum allowable ratio, or percentage of the view
`max`	`real`	Maximum allowable ratio, or percentage of the view

Description

Draws the sprite assigned to the running object scaled relative to the size of the active view camera. Ratio is calculated as a percentage of view width OR height (whichever is greater), with a value of 1 fully covering the view at any aspect (e.g. for backgrounds).

Because this ratio alone may result in sprites smaller or larger than desirable on one axis, a min and max ratio can also be supplied to limit size on the other axis than is calculated for the base ratio (e.g. for HUD elements). If max is set to 0, clamping will be disabled.

Also returns the scale multiplier, which can be used to position or scale other elements relative to the drawn sprite. Scale multiplier will also be reflected in built-in `image_xscale` and `image_yscale` instance variables.

Example

```
draw_self_scaled(0.25, 0.15, 0.5);
```

The "draw_sprite_scaled" Function

Syntax

```
draw_sprite_scaled(sprite, subimg, x, y, ratio, min, max);
```

Argument	Type	Description
`sprite`	`sprite`	The sprite to draw scaled
`subimg`	`integer`	The sprite frame index to draw
`x`	`real`	The horizontal room coordinate at which to draw the sprite
`y`	`real`	The vertical room coordinate at which to draw the sprite
`ratio`	`real`	Percentage of the view to occupy, as a value ranging from 0-1
`min`	`real`	Minimum allowable ratio, or percentage of the view
`max`	`real`	Maximum allowable ratio, or percentage of the view

Description

Draws a sprite scaled relative to the size of the active view camera. Ratio is calculated as a percentage of view width OR height (whichever is greater), with a value of 1 fully covering the view at any aspect (e.g. for backgrounds).

Because this ratio alone may result in sprites smaller or larger than desirable on one axis, a min and max ratio can also be supplied to limit size on the other axis than is calculated for the base ratio (e.g. for HUD elements). If max is set to 0, clamping will be disabled.

Also returns the scale multiplier, which can be used to position or scale other elements relative to the drawn sprite.

Example

```
var scale = draw_sprite_scaled(my_sprite, image_index, x, y, 0.25,
0.15, 0.5);

draw_sprite_ext(other_sprite, image_index, x, y + 128, scale, scale,
0, c_white, 1);
```

The "draw_sprite_scaled_ext" Function

Syntax

```
draw_sprite_scaled_ext(sprite, subimg, x, y, ratio, min, max, rot,
col, alpha);
```

Argument	Type	Description
`sprite`	`sprite`	The sprite to draw scaled
`subimg`	`integer`	The sprite frame index to draw
`x`	`real`	The horizontal room coordinate at which to draw the sprite
`y`	`real`	The vertical room coordinate at which to draw the sprite
`ratio`	`real`	Percentage of the view to occupy, as a value ranging from 0-1
`min`	`real`	Minimum allowable ratio, or percentage of the view
`max`	`real`	Maximum allowable ratio, or percentage of the view
`rot`	`real`	The sprite rotation angle, in degrees
`col`	`color`	The sprite blending color to draw, where `c_white` is default
`alpha`	`real`	The sprite transparency, ranging from 0-1

Description

Draws a sprite scaled relative to the size of the active view camera. Ratio is calculated as a percentage of view width OR height (whichever is greater), with a value of 1 fully covering the view at any aspect (e.g. for backgrounds).

Because this ratio alone may result in sprites smaller or larger than desirable on one axis, a min and max ratio can also be supplied to limit size on the other axis than is calculated for the base ratio (e.g. for HUD elements). If max is set to 0, clamping will be disabled.

Also returns the scale multiplier, which can be used to position or scale other elements relative to the drawn sprite.

Example

```
draw_sprite_scaled_ext(my_sprite, image_index, x, y, 0.25, 0.15, 0.5,
0, c_white, 1);
```

The "draw_surface_scaled" Function

Syntax

```
draw_surface_scaled(surf, x, y, ratio, min, max);
```

Argument	Type	Description
`surf`	`surface`	The surface to draw scaled
`x`	`real`	The horizontal room coordinate at which to draw the surface
`y`	`real`	The vertical room coordinate at which to draw the surface
`ratio`	`real`	Percentage of the view to occupy, as a value ranging from 0-1
`min`	`real`	Minimum allowable ratio, or percentage of the view
`max`	`real`	Maximum allowable ratio, or percentage of the view

Description

Draws a surface scaled relative to the size of the active view camera. Ratio is calculated as a percentage of view width OR height (whichever is greater), with a value of 1 fully covering the view at any aspect (e.g. for backgrounds).

Because this ratio alone may result in surfaces smaller or larger than desirable on one axis, a min and max ratio can also be supplied to limit size on the other axis than is calculated for the base ratio (e.g. for HUD elements). If max is set to 0, clamping will be disabled.

Also returns the scale multiplier, which can be used to position or scale other elements relative to the drawn surface.

> **CAUTION**
>
> Because surface data is *volatile*, surfaces are likely to be destroyed if the window is resized. Always check if a surface exists before drawing, and include regeneration code if it does not.

Example

```
draw_surface_scaled(my_surf, x, y, 0.25, 0.15, 0.5);
```

The "draw_surface_scaled_ext" Function

Syntax

```
draw_surface_scaled_ext(surf, x, y, ratio, min, max, rot, col, alpha);
```

Argument	Type	Description
`surf`	`surface`	The surface to draw scaled
`x`	`real`	The horizontal room coordinate at which to draw the surface
`y`	`real`	The vertical room coordinate at which to draw the surface
`ratio`	`real`	Percentage of the view to occupy, as a value ranging from 0-1
`min`	`real`	Minimum allowable ratio, or percentage of the view
`max`	`real`	Maximum allowable ratio, or percentage of the view
`rot`	`real`	The surface rotation angle, in degrees
`col`	`color`	The surface blending color to draw, where `c_white` is default
`alpha`	`real`	The surface transparency, ranging from 0-1

Description

Draws a surface scaled relative to the size of the active view camera. Ratio is calculated as a percentage of view width OR height (whichever is greater), with a value of 1 fully covering the view at any aspect (e.g. for backgrounds).

Because this ratio alone may result in surfaces smaller or larger than desirable on one axis, a min and max ratio can also be supplied to limit size on the other axis

than is calculated for the base ratio (e.g. for HUD elements). If max is set to 0, clamping will be disabled.

Also returns the scale multiplier, which can be used to position or scale other elements relative to the drawn surface.

> ⚠ CAUTION
>
> Because surface data is *volatile*, surfaces are likely to be destroyed if the window is resized. Always check if a surface exists before drawing, and include regeneration code if it does not.

Example

```
draw_surface_scaled_ext(my_surf, x, y, 0.25, 0.15, 0.5, 0, c_white,
1);
```

Special Thanks

Patreon credits

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

Very special thanks goes out to:

Patreon 'Enthusiasts'

Marvin Mrzyglod

Patreon 'Developers'

AshleeVocals

AutumnInAprilArt

Daniel Sato

Darktoz

Dirty Sock Games

Josef Scott

Meyaoi Games

Patreon 'Gamers'

Kampmichi (Forgers of Novelty)

All Other Patreon Supporters

Adam Miller (Actawesome)

Cosmopath

D Luecke

Alex Lepinay

Tarquinn J Goodwin

Creative Credits

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

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

VNgen Demo Voiceover

Kanen *(as Miki and Mei)*

End-User License Agreement ("EULA")

NOTE

Last updated: 12/16/2019

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

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

License Agreement

This License Agreement (the "Agreement") is entered into by and between XGASOFT (the "Licensor"), and you (the "Licensee"). This agreement is legally binding, and becomes effective when you purchase and/or download a free product from

XGASOFT or authorized third-party distributors. If you do not agree to the terms of this Agreement, do not purchase, download, or otherwise use XGASOFT products.

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

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

Standard Lifetime License

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

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

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

purposes.

Patreon Limited License

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

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

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

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

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

commercial activities. In order to continue using XGASOFT Property publicly, you must purchase a standard lifetime license.

This limitation shall not be applied retroactively, so that any existing, complete, and publicly available commercial and non-commercial properties using XGASOFT Property will not be considered in violation of this agreement. Furthermore, this limitation shall not apply in the event that XGASOFT suspends, revokes, or disables the contribution of financial support to XGASOFT through Patreon. In such case as contributions are limited or prohibited by XGASOFT (and not the Licensee), the terms of the Standard Lifetime License shall apply to any and all XGASOFT Property granted as rewards for recurring financial support prior to the date of suspension.

Single-User

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

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

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

Modifications

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

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

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

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

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

Mutability

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

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

Liability

By using XGASOFT Property, you agree to indemnify and hold harmless XGASOFT, its employees, and agents from and against any and all claims (including third party claims), demands, actions, lawsuits, expenses (including attorney's fees) and

damages (including indirect or consequential loss) resulting in any way from your use or reliance on XGASOFT Property, any breach of terms of this Agreement, or any other act of your own.

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

Governing Law

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

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

Conclusion

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