ULTIMATE Screenshot Creator

v1.11.0

for Unity 2018.4 to 2022.2

January 18, 2023

Contents

Thank you for purchasing *Ultimate Screenshot Creator*! I wish this package will meet your needs and expectations. Please do not hesitate to contact me if you have a feature request, or any question, issue or suggestion.

Arnaud,
support@wildmagegames.com

1 OVERVIEW

1.1 Introduction

Ultimate Screenshot Creator is the most complete and customizable screenshot creator available on the asset store. The asset is powerful, flexible, easy to use, and works in Editor and on all platforms.

For developers, it is the ideal tool to create professional marketing pictures, wallpapers and mobile store screenshots, using custom cameras, resolutions, backgrounds, watermarks, and more. Use the composition and batches process to generate hundred of multilingual promotional pictures in one click.

For your users, it is the ideal tool to take in-game screenshots with a lot of custom settings, and to display and manage the screenshots from the game. Perfect for AR apps.

There are several ways to use the asset, such as the Editor Window, the Screenshot Manager component, and even direct calls from scripts to capture Screenshots or Textures.

1.2 Requirements

- The solution works with Unity 5.0 and later free or pro edition.
- On iOS, adding the picture to the phone gallery requires the Photo Gallery access rights. By disabling some of the asset features it is possible to remove that requirement, see Section 4.13 for more details.
- We advise to use Unity 5.4 and later for a better GameView management using *GAMEVIEW_RESIZING* in editor (see FAQ 7.2.7).

1.3 Limitations

- Unity WebPlayer platform is not supported.
- SpeedTree leaves can not be rendered with a transparent background.

1.4 Known Issues

- Possible incompatibility with obfuscators. Whitelist the namespace AlmostEngine.Screeenshot.
- Due to a Unity bug, some screenshots may appear black in iOS devices depending on the Unity version and the MultiSampling quality settings.

2 QUICK START GUIDE

2.1 Screenshot Capture Workflows

Ultimate Screenshot Creator is a very flexible and easy to use asset, and there are several ways to use it. In the following section, we describe some of the most common way to use it.

2.1.1 Take screenshots in editor: ScreenshotWindow

The ScreenshotWindow is an editor window providing a quick way to take screenshots in editor. It can be fully configured and its settings are saved automatically.

1. Open the screenshot window in the menu *Window/AlmostEngine/Screenshot Window*.
2. Edit the window settings (see below).
3. The settings are automatically saved, except scene object references. To keep persistent references to scene objects, use a ScreenshotManager.

You will find more details about the ScreenshotWindow on Section 3.

2.1.2 Easy to use in-game screenshots: ScreenshotManager

The ScreenshotManager is a gameobject to be added on the scene to take screenshots in editor or in play mode. It is an all-in-one and easy to configure screenshot solution. It handles everything, like filenames, export directory, camera, canvas, etc.

1. Put the *ScreenshotManager* prefab into your scene.
2. Select the *ScreenshotManager* to configure it (see below).
3. Do not forget to Apply your modifications to the prefab to save them, or to create a new prefab.

You will find more details about the ScreenshotManager on Section 4.

Available Example: In the DefaultExample scene, the "ScreenshotManager" game object has a ScreenshotManager component.

2.1.3 Capture and export screenshots from script: SimpleScreenshotCapture

There are several way to take screenshot from scripts. For instance you can configure a ScreenshotManager game object and call the Capture() method from a custom script.

You can also take screenshots without any ScreenshotManager, by using the SimpleScreenshotCapture static methods. You can chose to capture the current screen, a set of cameras, or to specify all capture settings. It will take the screenshot and export it using the given fullname. See Section 8.1 for more details.

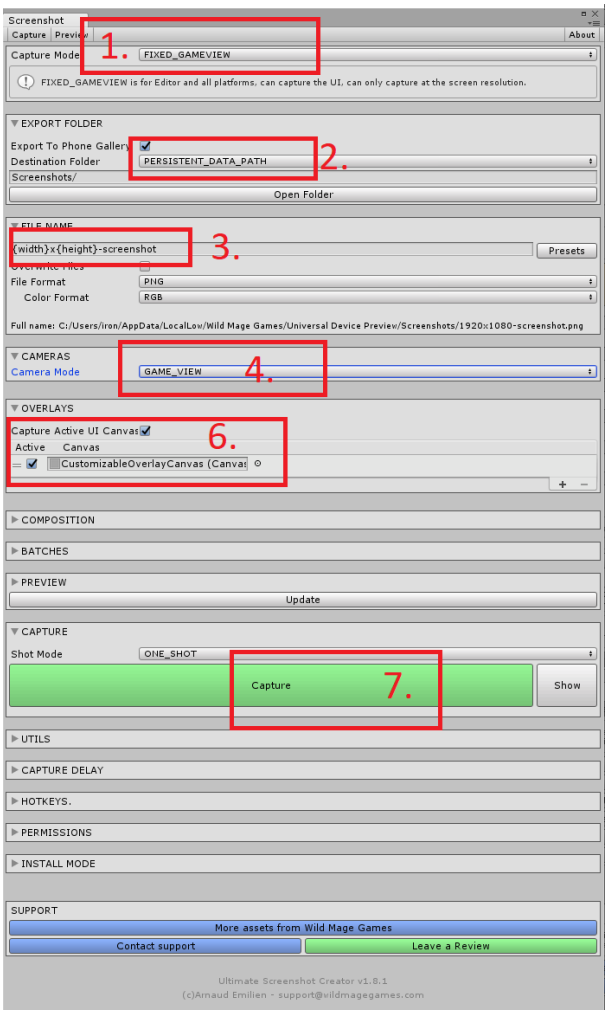
Available Example: In the DefaultExample scene, the "Simple Capture Script Example" game object has a CaptureScreenshotExample component.

2.1.4 Capture textures from script: ScreenshotTaker

The ScreenshotTaker component is the core of all screenshot taking methods of the asset. It provides several capture coroutines, for instance to capture the current screen, a set of cameras, etc. At the end of the coroutine process, the provided texture is updated and can be used directly. See Section 8.1 for more details.

Available Example: In the DefaultExample scene, the "CaptureTextureCanvas Example" game object has a CaptureCameraToTextureExample component.

2.2 Quick Screenshot Configuration



1. Select the capture mode. *GAMEVIEW_RESIZING* is the most robust capture mode, and captures the UI perfectly. It works in Editor and Windows Standalone only. *RENDER_TO_TEXTURE* works in editor and on all platforms, but Screenspace Overlay UI are not rendered. *FIXED_GAMEVIEW* works in editor and on all platforms, can only capture at the current screen resolution. It is the recommended settings for taking screenshots in-game. See Section 4.1 for more details.

2. Set the destination folder...

3. ... and the filename. See Section 4.2 for more details.

4. Select the camera mode: *GAME_VIEW* mode copies what you see in game view, *CUSTOM_CAMERAS* mode allows you to use a set of cameras. In custom mode, select the cameras to be used for the capture.

   Note that you can also customize their rendering settings. See Section 4.6 for more details.

5. If you capture *GAMEVIEW_RESIZING* or *RENDER_TO_TEXTURE*, select the resolutions mode: *GAME_VIEW* mode uses the game view resolution, *CUSTOM_RESOLUTIONS* mode allows you to customize the resolutions to be used. In custom mode, select the resolutions to be used for the capture, or use the list + to create a custom resolution. See Section 4.4 for more details.

6. The overlay system enables the easy integration of your company or game logo into each screenshot. Edit the example canvas or create your own to display your game logo. You can reference scene objects or prefabs. Select the overlays to be used for the capture. You can customize the default overlay or create a new one using a *Canvas*.

   Note that you can also disable the overlay system if you do not want any canvas, by simply disabling or removing all canvas in the overlays list. See Section 4.7 for more details.

7. Capture !

2.3 Platform Specific Configuration

This section details the platform specific configuration required to be able to take screenshots in-game.

2.3.1 Android

Capture Mode. The capture mode *FIXED_GAMEVIEW* is recommended to capture exactly what is on screen. *RENDER_TO_TEXTURE* is possible too.

Export Folder. On Android, *CUSTOM_FOLDER*, *PERSISTENT_DATA_PATH* and *DATA_PATH* all export to the external media directory associated with your app. This is the recommended behavior and it guarantees the access to the pictures on all Android devices without any permission requirement.

Note that with this settings, your saved screenshots will be removed when your app is uninstalled.

Picture Folder. You can also chose to export to the *PICTURE_FOLDER*. Your pictures will not be removed when your app is uninstalled, however it requires the *access to external storage* permission (see below).

Export to gallery. To export to the gallery, check the *Export to phone Gallery* element on the inspector.

Permission. No permissions are required to export screenshots and add them to the phone gallery. When exporting to *PICTURE_FOLDER*, you need the *access to external storage* permission.

In the Android Player Settings, set *Write Access* to *External*. In that case, the permission should be asked at the first app launch.

If your app does not have access to the external storage (for instance if the user rejects the permission), the export will fall back to the external media directory associated with your app.

2.3.2 iOS

IMPORTANT. On Unity 5.6 and later, you need to add the dependency to the **Photos** framework to the iOSUtils.m plugin. This should be done automatically, but if you have compilation issues on iOS, you can do it manually. In Unity, select the iOSUtils.m file, look in the Platform settings, and add the Photos dependency in the *rarely used frameworks* list.

Capture Mode. The capture mode *FIXED_GAMEVIEW* is recommended to capture exactly what is on screen. *RENDER_TO_TEXTURE* is possible too.

Export Folder. On iOS, the export folder settings are ignored and the screenshots are exported to the media directory associated with your app.

Export to gallery. To export to the gallery, check the *Export to phone Gallery* element on the inspector, and enable the iOS permissions in the ScreenshotManager inspector.

Permission. No permissions are required to only export screenshots.

Adding the pictures to the phone gallery requires the Photos access rights. This is done automatically by the script *iOsPostProcessBuild.cs*. Note that is it possible to remove the permission needs by excluding some features of the asset. See Section 4.13 for more details.

The permission to access the gallery should be asked at the first screenshot capture. To ask the permission at startup, add the script *RequestAuthAtStartup* to your project, or call *iOsUtils.RequestGalleryAuthorization()* when you want to.

Photo Usage Description. You can personalize the Photo usage description by editing the associated field in the ScreenshotManager inspector, or by editing the *UltimateScreenshotCreator/Assets/PhotoUsageDescription.asset* object.

2.3.3 WebGL

1. Move the *Plugins* folder at the root of the *Assets* folder. The plugin should be located exactly here: *Assets/Plugins/WebGL/WebGLUtils.jslib*.
2. The capture mode the capture mode $FIXED\_GAMEVIEW$ is recommended to capture exactly what is on screen. $RENDER\_TO\_TEXTURE$ is possible too.
3. Note that with WebGL, the export mode is ignored since the image will be downloaded using the web browser.

3 SCREENSHOT WINDOW

The screenshot window allows to easily take screenshots in the editor, without the need to add anything on your scenes. Open the settings window in *Window/AlmostEngine/Screenshot Window*.

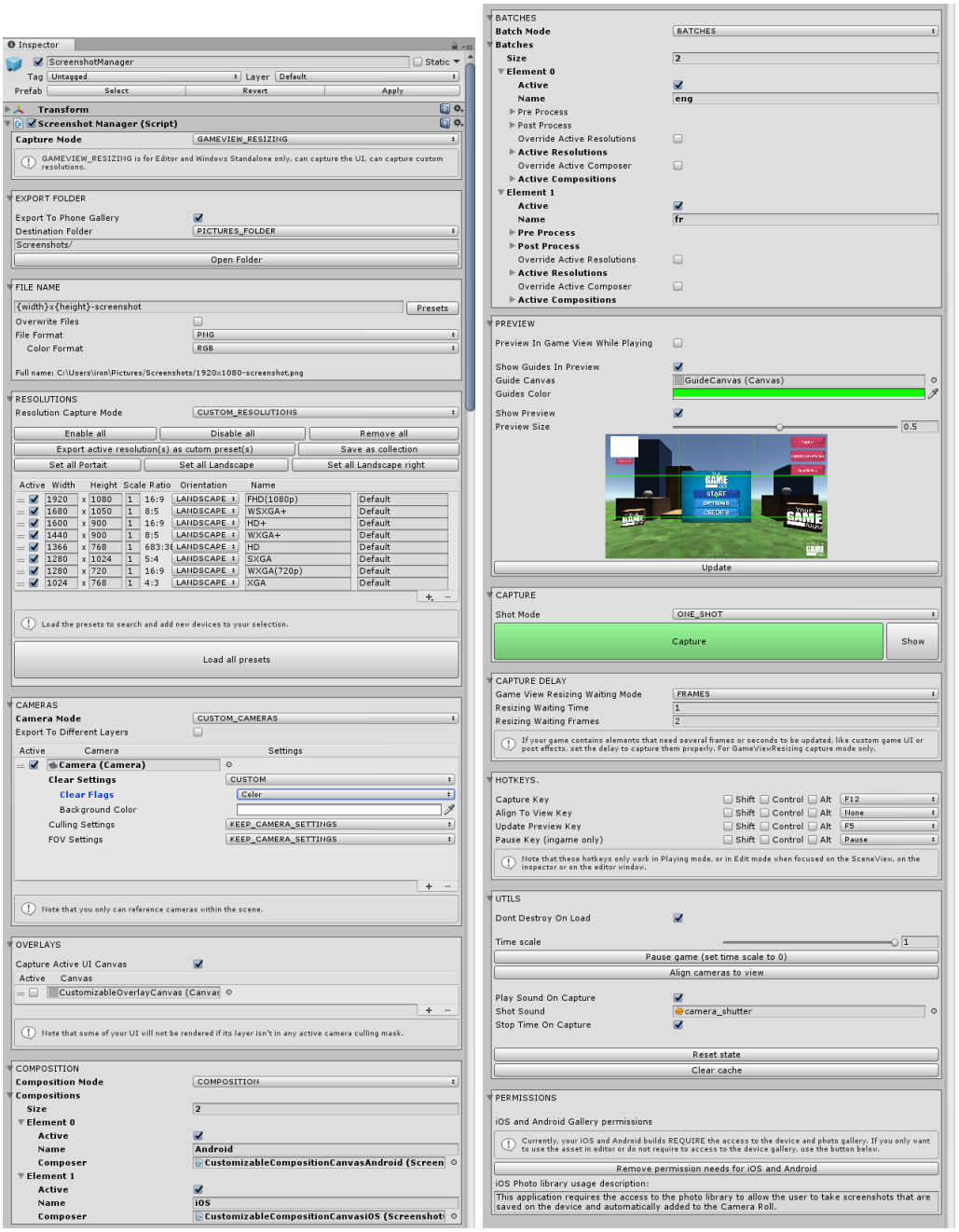


Figure 1: The screenshot window.

The Screenshot Window settings are saved in the file *ScreenshotWindowConfig.asset*. Note that the Screenshot Window can not save references to scene objects. Reference prefabs objects or use a Screenshot Manager game object.

See Section 4 for more details.

4 CONFIGURATION GUIDE

4.1 Capture Mode

Figure 2: Capture mode settings.

There are three capture modes:

- *GAMEVIEW_RESIZING* is the most robust capture mode, and captures the UI perfectly. It works in Editor and Windows Standalone only. Note that with this mode the editor gameview or game windows needs to be resized for a few milliseconds.
- *RENDER_TO_TEXTURE* works in editor and on all platforms, but Screenspace Overlay UI are not rendered.
- *FIXED_GAMEVIEW* works in editor and on all platforms, can only capture at the current screen resolution. It is the recommended mode for taking screenshot in-game on all platforms.

	GAMEVIEW_RESIZING	*RENDER_TO_TEXTURE*	*FIXED_GAMEVIEW*
Custom Cameras	√	√	√
Custom Resolutions	√	√	Screen resolution
UI	√	No UI	√
Overlays	√	No overlays	√
In Editor	√	√	√
In Game	Windows Standalone only	All platforms	All platforms

Table 1: Capture modes comparison.

4.2 Destination

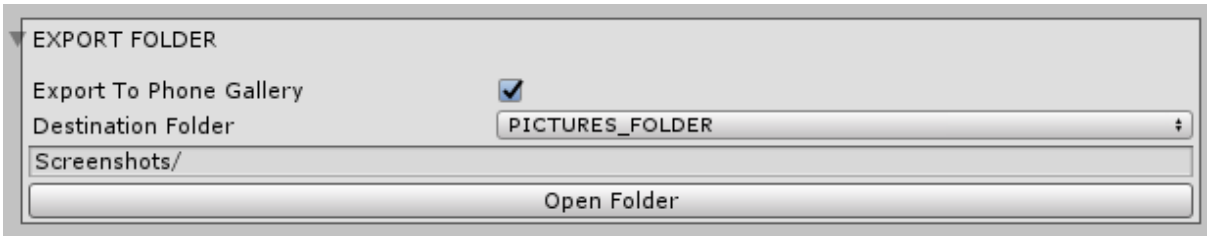


Figure 3: Destination settings.

There are three export modes:

- *CUSTOM_FOLDER* allows you to select the export folder.
- *DATA_PATH* exports the screenshots relatively to the project data path.

- *PERSISTENT_DATA_PATH* exports the screenshots relatively to the persistent data path.
- *PICTURE_FOLDER* exports the screenshots relatively to the platform picture folder. It is the recommended mode for taking screenshot in-game on all platforms.

Export to Phone Gallery. Set if the screenshot should be automatically added to the phone gallery on iOS and Android.

4.3 File Name

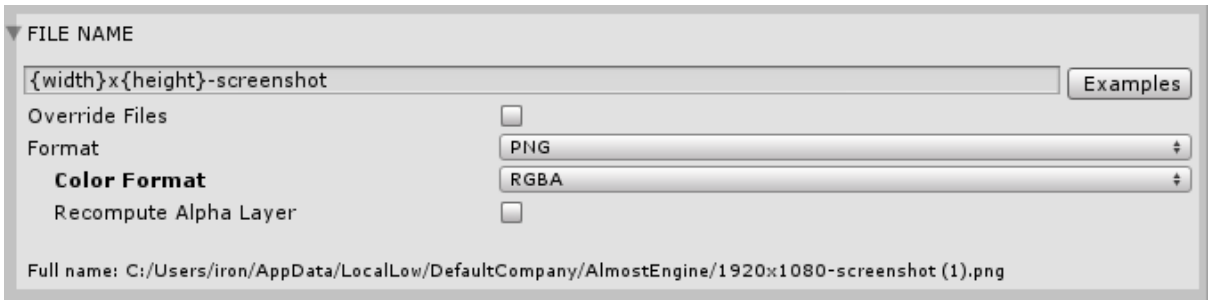


Figure 4: File name settings.

FileName. Defines the file name to be used for the screenshots. Use the *Examples* button to select one of the name presets.

You can use and combine the following symbols to better match your needs:

- {year}, {month}, {day}, {hour}, {minute}, {second} for the current time information,
- {width}, {height}, {scale}, {ratio}, {orientation}, {ppi}, {percent}, {name}, {category} for the resolution information.
- {layer} for the camera name in separated layer mode.
- {composer} for the current composer name in composition mode.
- {batch} for the current batch process name in batches mode.

Note that you can use the file name to group screenshots by resolution folders, etc. For instance {*width*}-{*height*}*/screenshot.png* will create one folder by resolution and create one *screenshot.png* in each of them.

Overwrite Existing Files. By default, if you try to create a screenshot file that already exists, its name will be incremented. For instance: screenshot.png, screenshot(1).png, ... Set *overwrite* to *true* if you want to overwrite the existing files.

Format. You can export to *PNG* or to *JPG* with a custom quality.

Color Format. In *PNG*, you can export to *RGB* or to *RGBA*. Use *RGBA* to create screenshots with an alpha layer, enabling transparent backgrounds.

Recompute Alpha Layer. Force alpha layer to be recomputed. This is a costly process. Use only if you have alpha problems in *RGBA* mode.

Full Name Preview. You can look at the full name preview to check if everything is correct. Note that the resolution information used for the full name preview is the one of the first resolution in the list.

4.4 Resolutions

Resolution Capture Mode. There are two resolution capture modes :

- *GAME_VIEW* just capture the game view.
- *CUSTOM_RESOLUTIONS* allows you to capture multiple resolutions in one click.

Managing resolutions. Use the resolution list to specify which resolutions are going to be captured. Use the list + button to add one of the resolution presets. Select *ALL* to include the whole group in one click.

Select the resolution to be removed and press the list − button, or right click on the resolution and select *remove item element*.

Scale. This setting allows you to easily take ultra HD screenshots. For instance, a resolution of 1600×1200 with a scale of 2 will take a screenshot with a resolution of 3200×2400.

Note that Unity limits the maximum resolution size to $\sim 8000 \times 4000$

Orientation. Switch between *LANDSCAPE* and *PORTRAIT* mode.

Resolution Name and Category. The resolution name can be customized, and used in the file name field with the symbol {*name*}. You can also specify a category to be used with the file name symbol {*category*}.

4.5 Managing Resolution Presets

Resolution presets are stored as asset files in the project folder. It is possible to easily create your own presets.

Creating Custom Presets. Click on "Save active resolution(s) as custom preset(s)" to automatically create new preset files using the resolutions in your custom resolution list.

Also, you can right click on your project folder and select *Asset/Almost Engine/Ultimate Screenshot Creator/Custom preset*.

Creating Custom Collections. Collections are set of presets that can be grouped and loaded simultaneously.

To create a new collection, you can export your active resolution list to a new collection by clicking on "Save as collection".

Also, you can right click on your project folder and select *Asset/Almost Engine/Ultimate Screenshot Creator/Custom collection*. Then, add all desired resolution preset to the list.

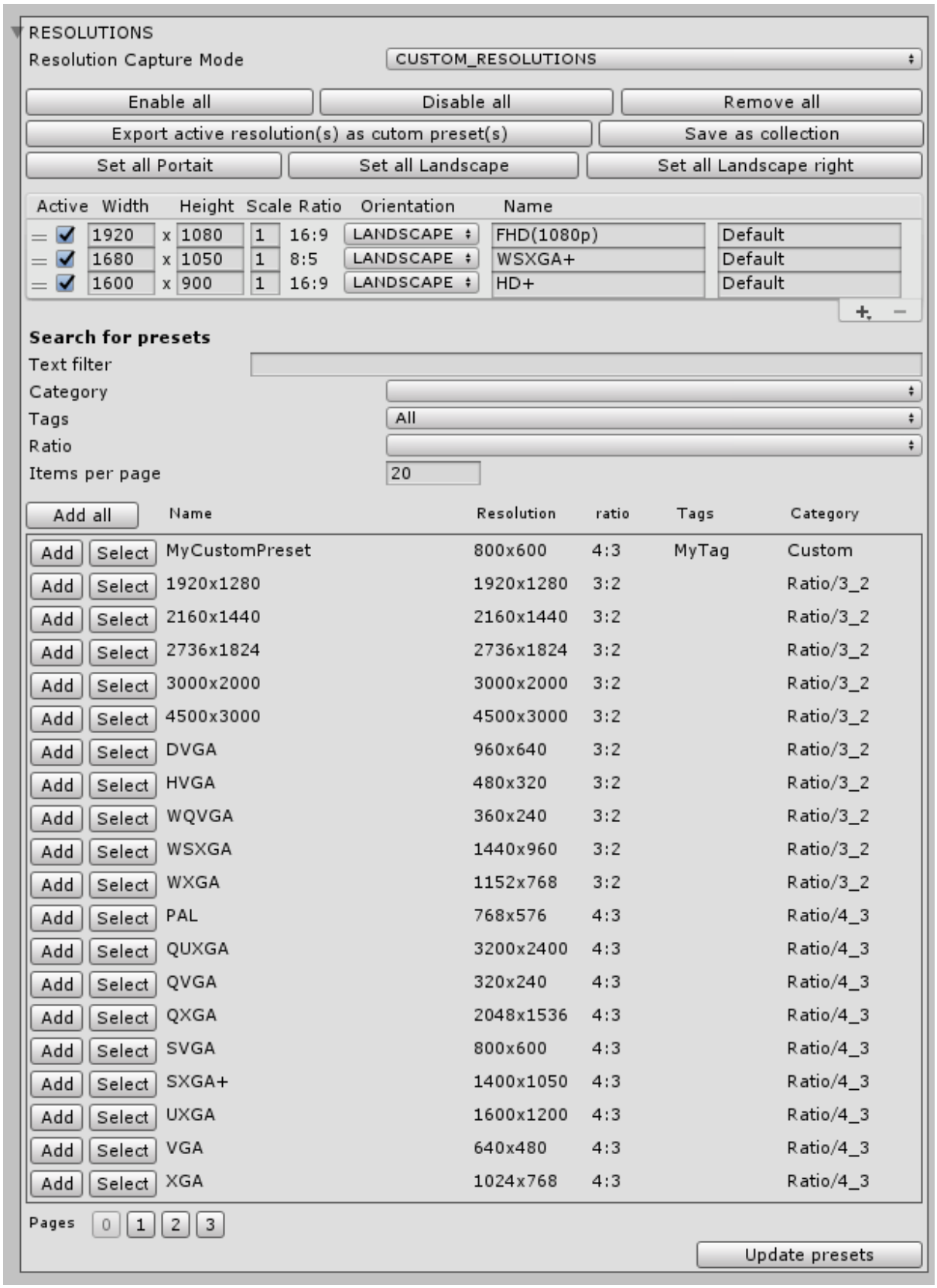


Figure 5: Resolutions settings.

Copying Presets and Settings to an New Project. If you have some custom presets you want to copy to a new project, simply copy and paste your preset .asset and .meta files.

To copy your Screenshot Manager settings, copy ScreenshotWindowConfig.asset to your new project.

4.6 Cameras

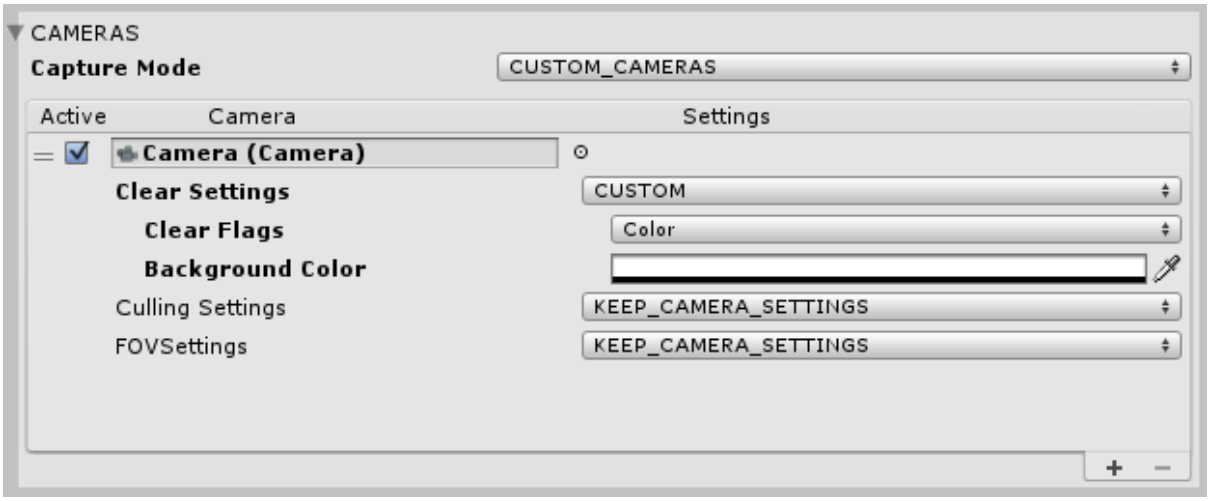


Figure 6: Camera settings.

Camera Capture Mode. There are two camera capture modes :

- *GAME_VIEW* just capture the game view.
- *CUSTOM_CAMERAS* allows you the customize the cameras to be used for the capture.

Export to separate layers. By enabling this option, a screenshot will be created for each camera. See Section 7.1.6 for more details.

Managing Cameras. Use the camera list to specify which cameras are going to be used for the capture. You can select multiple cameras at a time if your game requires a layered camera rendering (for instance for UI elements or overlays, or if you have a camera for the scene rendering and an other camera for rendering the player gun).

Note that the other cameras of the scene will be disabled during the capture.

Custom Camera Settings. Camera settings can be overwritten during the capture:

- *KEEP_CAMERA_SETTINGS* keeps the camera settings unchanged,
- *CUSTOM* allows you to change the camera clear mode and background color, the camera culling mask, and the camera Field Of View.

Note that the custom settings overwite the current camera settings during the capture process, and should correctly restore them at the end of it.

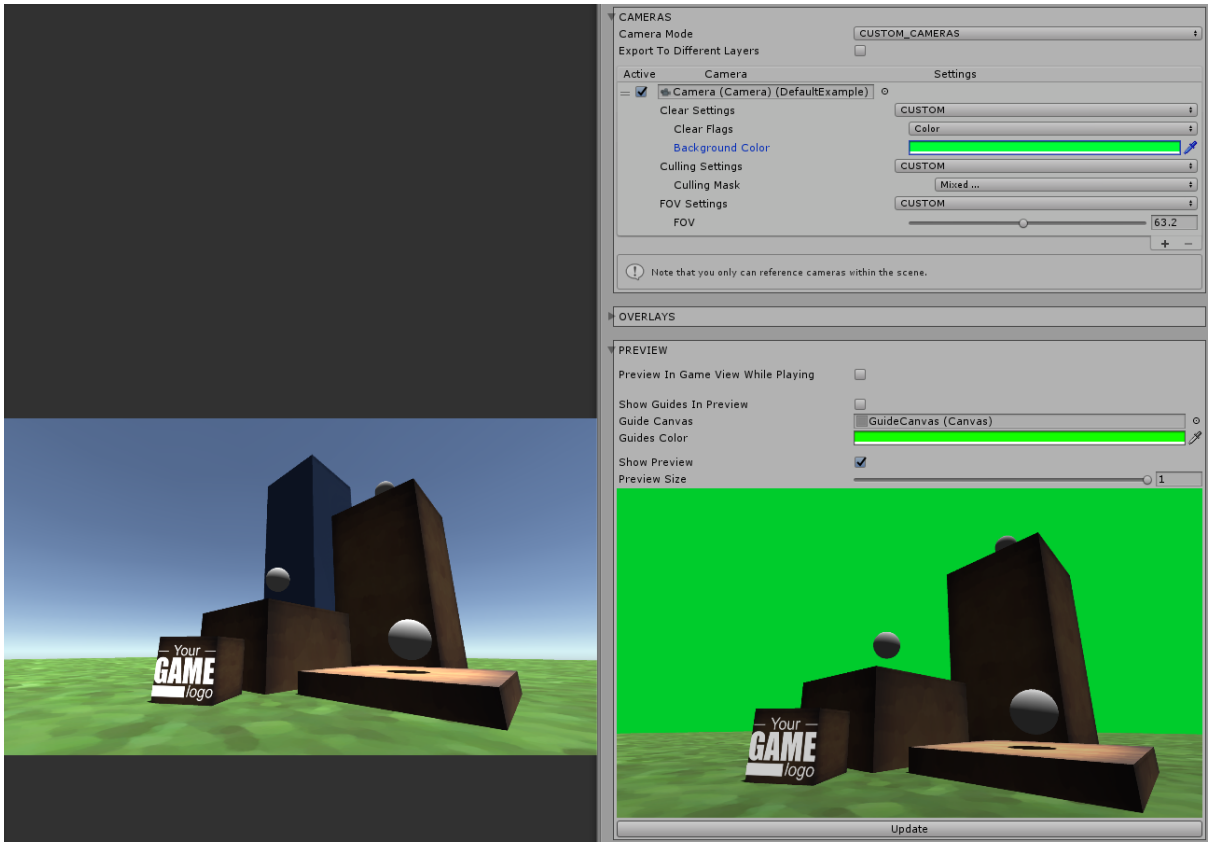


Figure 7: Example of custom camera settings.

4.7 Overlay Canvas

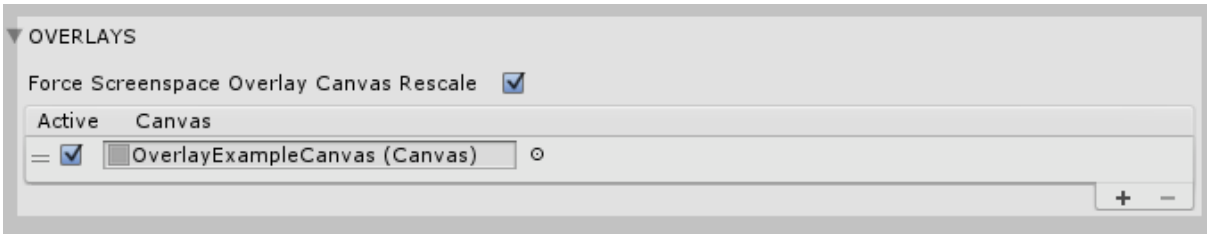


Figure 8: Overlay settings.

The overlay system allows you to easily integrate your game or studio logo into each screenshot, and much more. Use the overlay list to specify which overlays are going to be used for the capture.

Force Screenspace Overlay Canvas Rescale. Force Screenspace Overlay Canvas to be rescaled in $RENDER\_TO\_TEXTURE$ mode.

Render Active UI Canvas. You can choose to hide or not all your active game UI during the capture.

Default Overlay Customization. The *OverlayExampleCanvas* is a *Canvas* prefab. You can customize it or create your own overlay canvas.

Creating Your Own Overlays. The overlay system is based on the Unity 4.6 Canvas system, and allows unlimited customization of the screenshot overlays. To create an overlay:

1. Create a new Unity *Canvas*.
2. Edit your new *Canvas* as you want.
3. Save the Canvas as a prefab and remove it from the scene.
4. Create a new overlay item in the screenshot manager list, and select your new *Canvas* prefab.
5. Disable the *Canvas* game object. This keeps the *Canvas* hidden during the game and only displays it during the capture.

Figure 9: Example of screenshot captured using the default overlay (Viking scene, ©Unity).

4.8 Composition

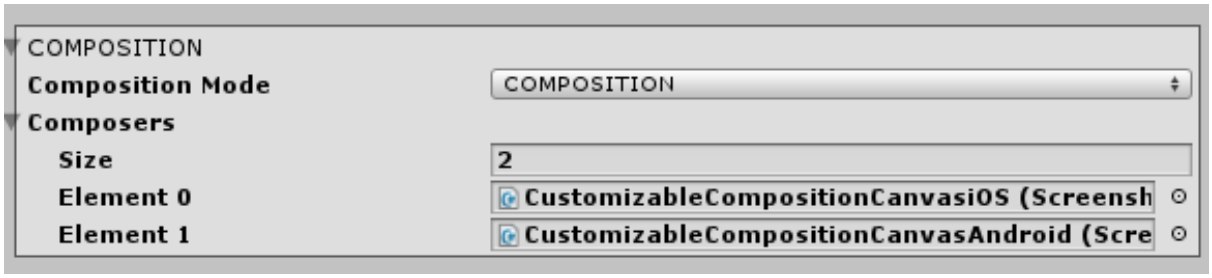


Figure 10: Composition settings.

The composition system allows to automatically capture your game and embed it in a promotional picture. Set the composition mode to *SIMPLE_CAPTURE* to disable it, or set it to *COMPOSITION*.

For each composer added to the composition list, the whole screenshot process is run to capture and integrate the screenshots within the composer canvas.

Creating Your Own Composition. Create a new Canvas game object and add a ScreenshotComposer component. Set the default UI Camera prefab to the camera field. The canvas field should reference the current composition canvas. In the textures list, add all RawImage game objects where your game will be embedded. For instance, it can be the UI image positioned in front of the device screen.

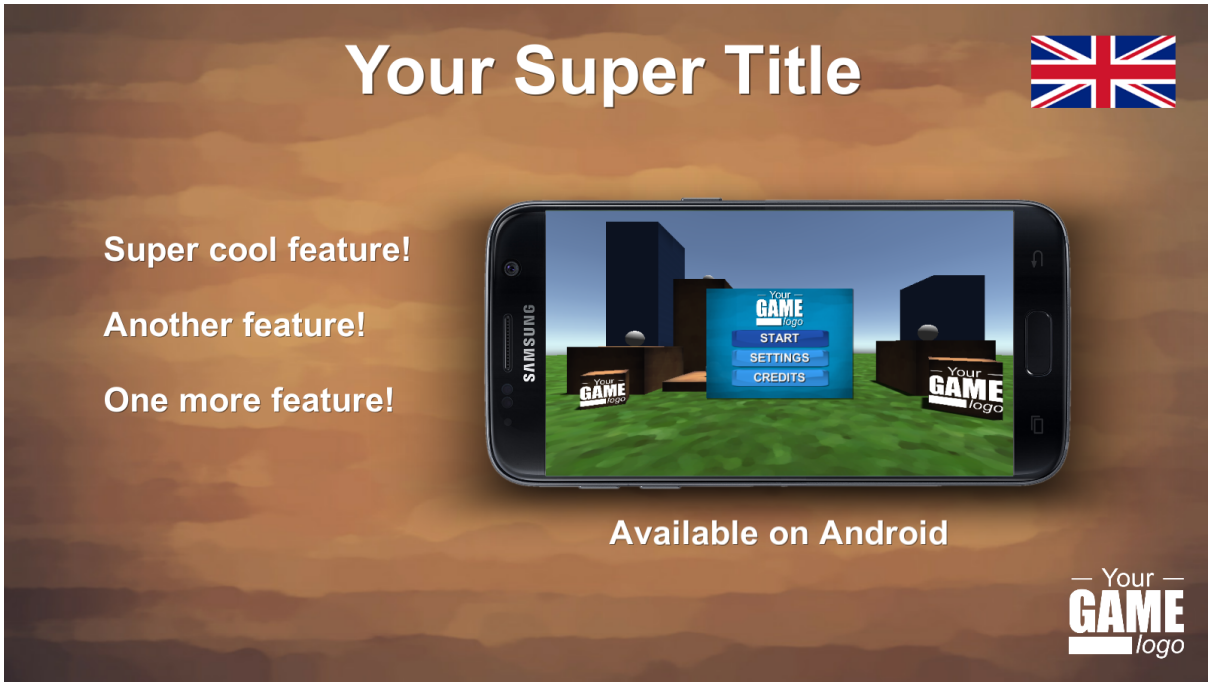


Figure 11: Example of composition using the Android template and the simple localization.

Figure 12: Example of composition using the iOS template and the simple localization.

4.9 Batches and Process

The batches system allows to automatically capture series of screenshots while applying some scripts before and after the capture. Set the batches mode to $SIMPLE\_CAPTURE$ to disable it, or set it to $BATCHES$.

For each batch added to the batches list, the whole screenshot process is run to capture with all pre process and post process.

For instance, to capture your game in multiple language, you can create two batches, one that does nothing, and one that changes the language in pre process and restore it in post process.

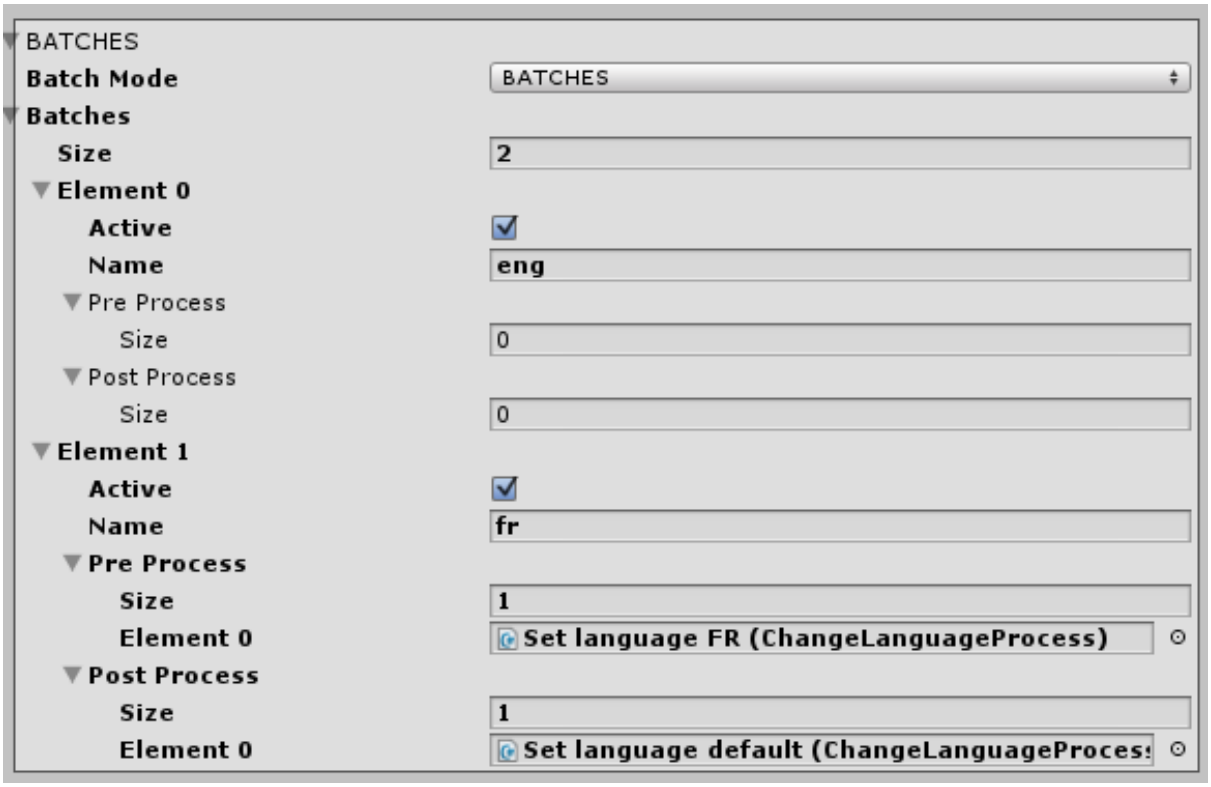


Figure 13: Composition settings.

Creating Your Own Process. Create a new script that inherits from ScreenshotProcess. Overwrite the Process method or coroutine depending on your needs. Add your script to a new game object of prefab, and add the game object reference to a pre process or post process list.

4.10 Preview

Guides. You can display a photography guide to better frame your screenshots. The default guide is a *Canvas* prefab. This guide is visible in the preview picture only, and will not be rendered in the final image.

Preview in GameView while Playing. If you want to visualize how the screenshot will look while playing, you can set this option to true. When the game will start playing, the preview settings will be applied to the GameView, and restored when the game is stopped.

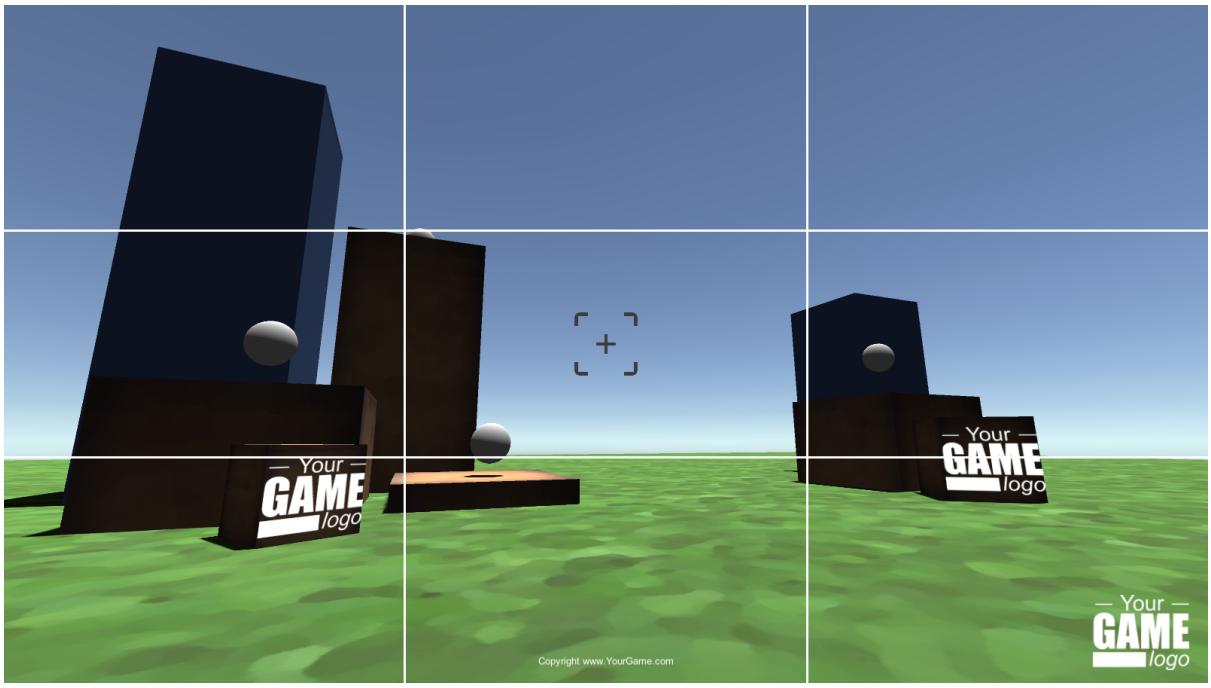


Figure 14: Use the framing guide in GameView while playing to better frame your screenshot.

4.11 Capture

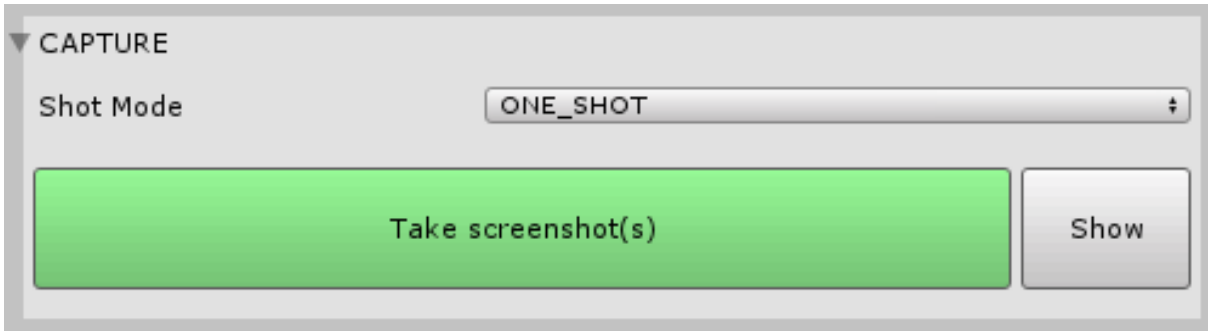


Figure 15: Capture settings.

Capture Mode. There are two capture modes:

- *ONE_SHOT* captures one frame,
- *BURST* allows to take a series of screenshots with a fixed and customizable timestep. Note than you can use the screenshots as inputs of a GIF creator software.

4.12 Hotkeys.

Figure 16: Hotkeys.

In this panel you can set the various hotkey values, and specify the sound to be used for the capture. Note that for the hotkeys only works when focused on the SceneView, ScreenshotManager Inspector or ScreenshotWindow, or on the GameView in play mode.

4.13 Permissions

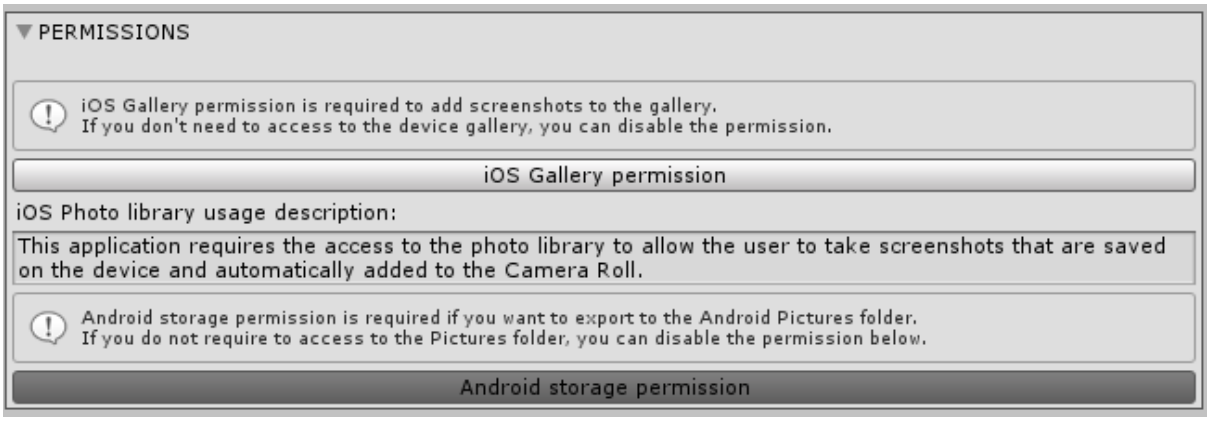


Figure 17: Permission settings.

iOS Gallery Permission To display your screenshot on the iOS Gallery, you need to request the Gallery permission. Toggle the iOS Gallery permission to do it automatically. You will be able to customize the usage description that is adde to your iOS app.

Android Storage Permission To export to the Pictures folder, you need to request the storage permission. Toggle the Android Storage Permission button to automatically do it.

4.14 Exclude Features from Build

You can choose to exclude some features from the build depending on your needs. If you want only an in-editor use, you can exclude everything and nothing will be added to your project. You can also exclude the Gallery and Share features if you do not use them to prevent the need for any iOS permission request.

Figure 18: Exclude form build settings.

4.15 Utils

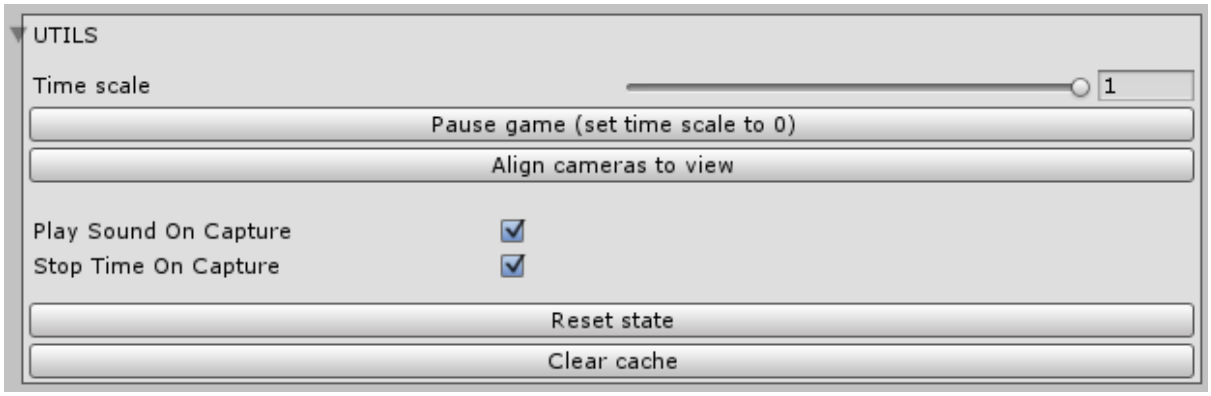


Figure 19: Utils.

Align to View. You can align all the cameras in the camera list with the current scene view. Note that this is a reversible operation using Undo/Redo.

Time Scale. You can use these tools to pause or slow down the time while playing to edit the scene and/or better frame your screenshot.

Reset State. See FAQ 7.2.8.

4.16 Screenshot Taker Config

The ScreenshotTaker component is automatically added to each ScreenshotManager.

GameView Resizing Waiting Mode. In *GAMEVIEW_RESIZING* mode, you can specify the number of frame or time to wait between the screen resizing and the screenshot capture. It is particularly useful if you use some script or special effects that need some time to adapt to the screen size.

Force Layout Preservation. See Faq 7.2.7.

5 EXTRA FEATURES

5.1 Sharing on iOS, Android, WebGL

It is possible to directly share a screenshot on WebGL, iOS and Android. When the share action is triggered, the share popup of the device will be shown to the user, who will then be able to share the screenshot using all possible apps of its device.

You can share using the ScreenshotManager ShareAll() method, or by directly calling the ShareUtils.ShareImage() method.

Can share. You can check if the share feature is supported by your platform and browser by calling the ShareUtils.CanShare() method.

WebGL compatibility. On WebGL, the share button is based on Navigator.share API. Compatible browsers include Microsoft Edge, Safari, Chrome for Android, Opera for Android, Safari for iOS.
More info on https://developer.mozilla.org/en-US/docs/Web/API/Navigator/share

ShareOnlyActivate. If you want to add a share button that is only enabled when the platform and browser support sharing, add a ShareOnlyActivate component on the button. It will be automatically disabled if sharing is not possible to prevent user confusion.

Example. The customizable ValidationCanvas provides an example of share button.

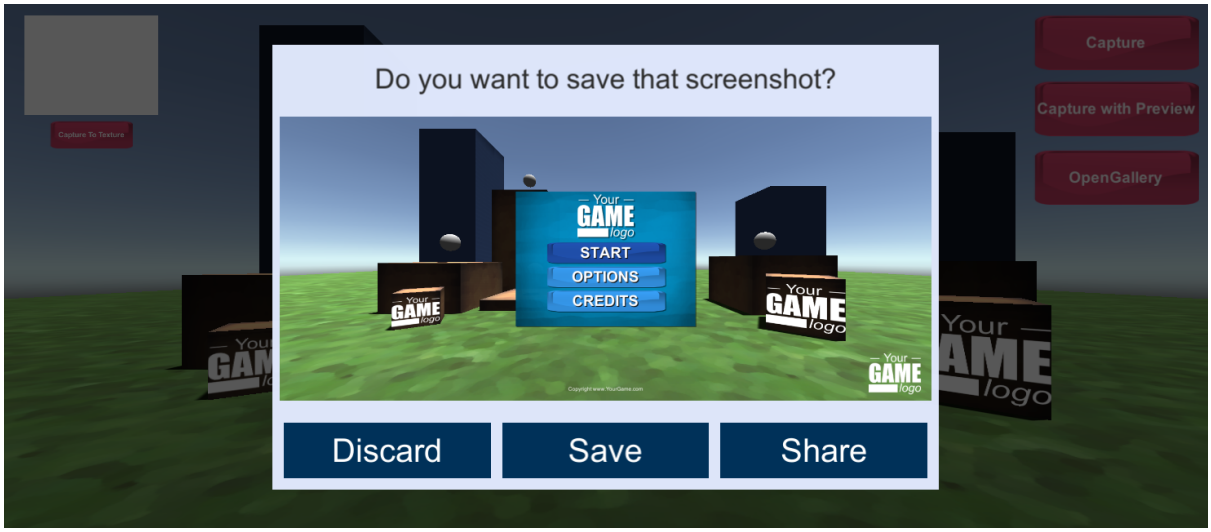


Figure 20: The customizable ValidationCanvas share button.

5.2 In-game Gallery and Screenshots Management

The asset contains an in-game gallery system to display and manage the taken screenshots. It is based on the Grid Gallery to display the images, and the Greybox preview to perform actions on the screenshots.

5.2.1 Customizable Grid Gallery

The GridGalleryCanvas component is an example of component displaying a gallery based on the grid layout component. It inherits from the ScreenshotGallery, a more generic class which contains the basic in-game gallery methods.

Figure 21: Example canvas for the GridGalleryCanvas component.

You can customize the canvas by creating a new canvas prefab and editing it as you want. You can also create a new component inheriting from GridGalleryCanvas or from ScreenshotGallery, for instance to create a gallery script using an different layout than the grid layout.

Reference to the Greybox Canvas. By default, when an image is selected, a Greybox canvas is opened with that image to be displayed.

Automatic folder path. By default, you have to manually specify the folder path to be used for loading the screenshots from the device. If you are already using a ScreenshotManager, you can add a *SetScreenshotGalleryFolderPath* component to the *GridGalleryCanvas* gameobject to automatically set the path as the ScreenshotManager export directory.

Available Example: In the DefaultExample scene, you will find a functional example of the gallery grid, the greybox and the confirmation window.

5.2.2 Customizable Greybox

The GreyboxCanvas component is an example of component to handle the gallery image selection, for instance to zoom perform several actions.

Figure 22: Example canvas for the GreyboxCanvas component.

One of the example action is to delete the selected screenshot. When clicked, the ui button shows the confirmation canvas, and the yes button calls the GreyboxCanvas delete method.

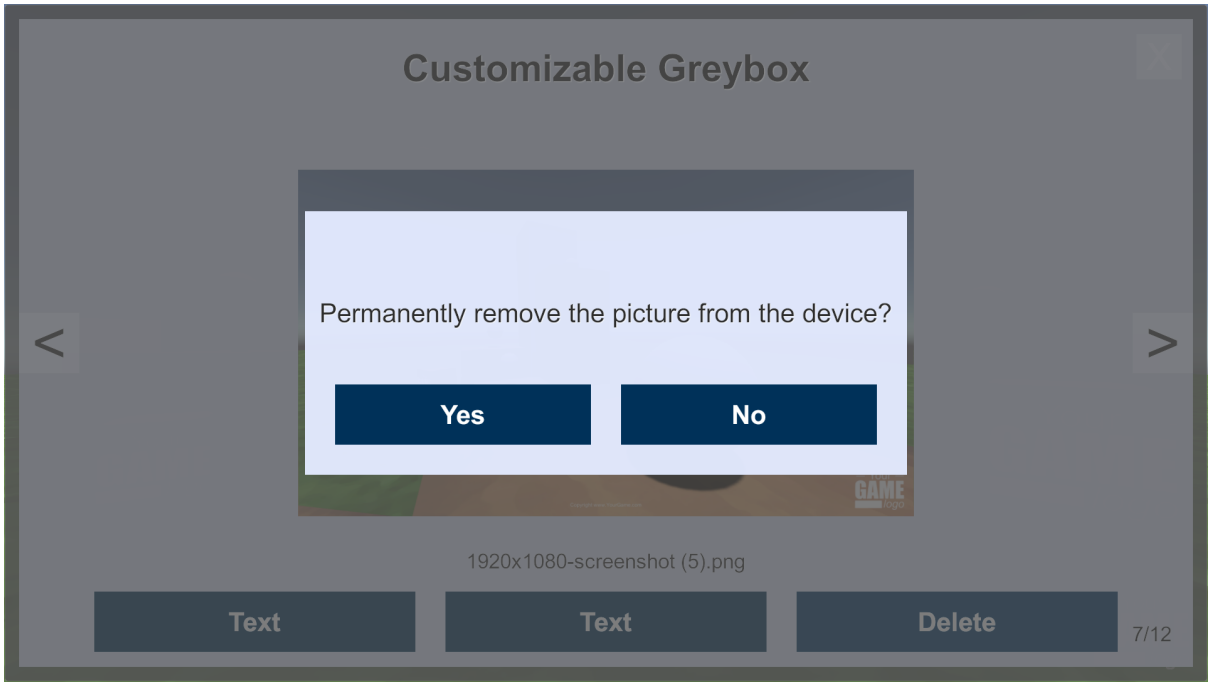


Figure 23: Example of confirmation window when removing a screenshot.

5.3 Display a Validation UI

The *ValidationCanvas* component shows how to call a capture event, and then display a validation ui to save or discard the screenshot. The idea is as follow:

1. Update the texture without exporting them using *UpdateAll()*.
2. Wait for the capture end event.
3. Display a UI and update its texture by accessing the screenshot texture.
4. Call *ExportAllToFiles()* if the user validates the screenshot.

Figure 24: Example of validation canvas to preview the screenshot before saving it.

Available Example: In the DefaultExample scene, you will find a functional example of the ValidationGUIExample.

5.4 Thumbnail

The ShowScreenshotThumbnail script allows to temporarily show the screenshot thumbnail after each capture process. To show a thumbnail, just add the CustomizableThumbnailCanvas prefab to your scene. Note that you can customize it as you want, for instance by adding animations, etc.

5.5 Screenshot Cutter

If you want to capture only a sub-part of the screen, add the *ScreenshotCutter* component to one of your scene object. Set the SelectionArea property to the part of the screen you want to capture. It can be any RectTransform, such as an image or a UI panel. The *CutterExample* scene and *ScreenshotCutterExample* prefab illustrate a possible use of the screenshot cutter.

Note that the screenshot cutter only works on *GAMEVIEW_RESIZING* or *FIXED_GAMEVIEW* modes.

5.6 Message Canvas

The message canvas component is a simple script to display some text when the screenshot manager export succeeds or fails. By adding the MessageCanvas prefab to your scene, the message will be displayed automatically. If you do not want any messages, simply remove the MessageCanvas game object.

6 SIMPLE LOCALIZATION

The Simple Localization asset allows to easily localize UI texts and images. It has been added freely to the Ultimate Screenshot Creator asset to enable the easy localization of your screenshots, but can be used independently to localize your whole game.

Language Settings. The LocalizationLanguages.asset file contains the list of the language IDs to be used by the localization components. It can be edited directly, or edited using the inspector of any SimpleTextLocalizer or SimpleImageLocalizer.

Text Localization. Add a SimpleTextLocalizer component to the UI text you want to localize. For each language id, write the localized text.

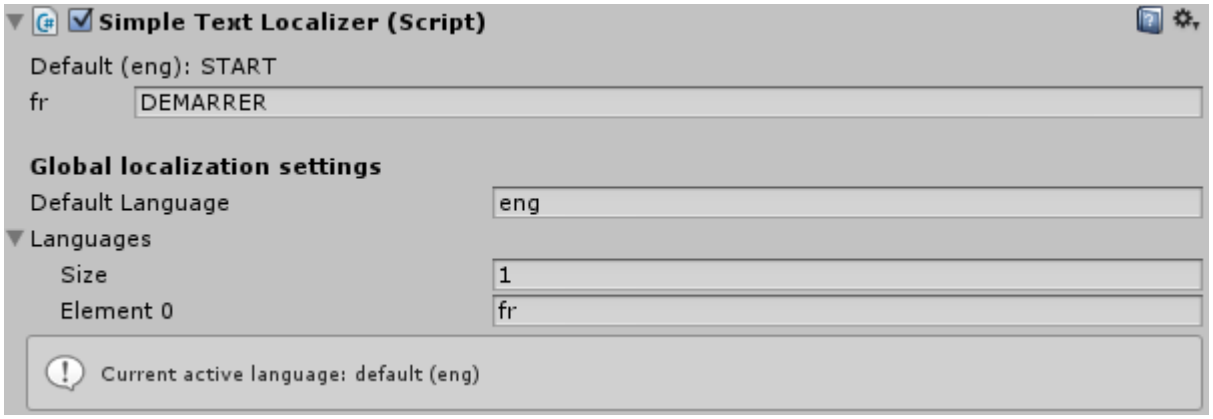


Figure 25: Text localization example.

Image Localization. Add a SimpleImageLocalizer component to the UI image you want to localize. For each language id, set the texture to be used.

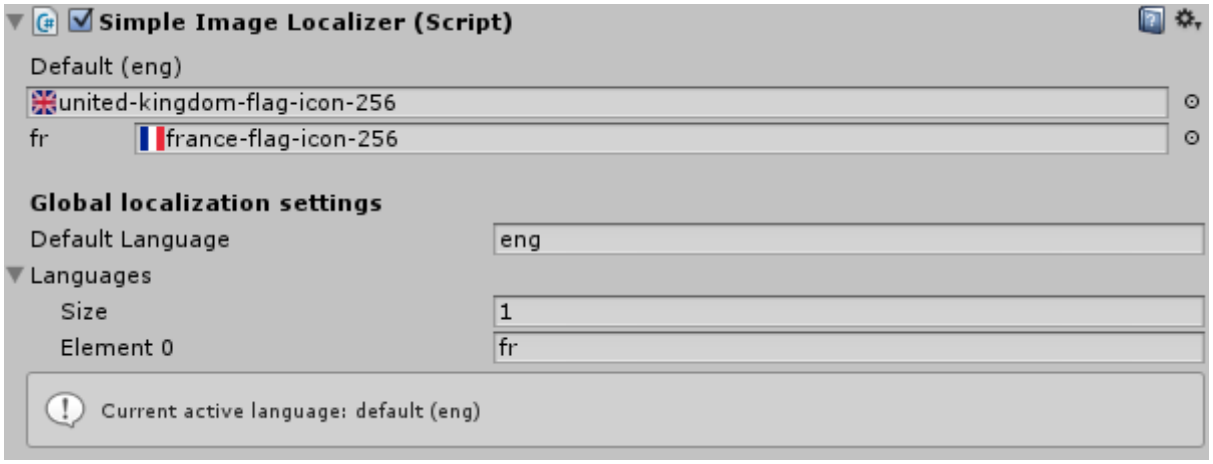


Figure 26: Image localization example.

Changing the Language. In play mode, call the SimpleLocalizationLanguagesAsset.SetLanguage method to change the language of all text and images using the simple localization scripts.

7 FAQ

7.1 How to ...

7.1.1 Do a minimal install of the plugin

You can safely remove the following folders:

1. AlmostEngine/UltimateScreenshotCreator/Documentation
2. AlmostEngine/UltimateScreenshotCreator/Examples
3. AlmostEngine/SimpleLocalization/Examples
4. AlmostEngine/Shared/Examples

7.1.2 Hide Some Scene Objects

Using a component You can hide a specific objet by adding the component *HideOnCapture* to the game object.

Using a culling layer You can hide all objects of a specific layer in *CUSTOM_CAMERAS* mode:

1. For each camera, select the camera and *CUSTOM_SETTINGS* culling mode.
2. For each camera, deselect the layers to be hidden in the culling list.

7.1.3 Access the screenshots taken by the ScreenshotManager

After the capture process, the textures can be accessed using the *ScreenshotManager GetActiveResolutions()* method, which returns a list of all active resolutions that were captured. It contains only one element in *FIXED_GAMEVIEW* mode. The texture used is stored in the *m_Texture* field, and its full file name in the *m_Filename* field.

7.1.4 Create a Screenshot with a Transparent Background

1. Set your camera clear mode to color, or use the *CUSTOM_CAMERAS* and *CUSTOM_SETTINGS* to temporary overwrite the camera clear settings.
2. Select the *color* clear mode, and set the color to 255 255 255 0 (white transparent).
3. Set the export format to *PNG* and choose the *RGBA* color format.
4. If you have an issue with the alpha layer, try the *RecomputeAlphaLayer* settings (Not in HDRP).
5. In HDRP follow the instruction below.

7.1.5 Create a Screenshot with a Transparent Background using HDRP

Follow the guide above, and then do the following.

1. Check that the screenshot manager settings are PNG, RGBA, recompute alpha to false, camera clear color white transparent.
2. In your HDRP sky and fog profile, set fog to false. If the fog is enabled, its color overwrites the camera background color and this is not transparent anymore.
3. In your HDRenderPipelineAsset, set color buffer format to R16G16B16A16, to enable transparency.

7.1.6 Export using separated layers

Exporting to separated layers means that each camera will be rendered separately.

1. Set your export name. The {layer} symbol in the filename will be replaced by the name of the camera.
2. Set the capture settings to PNG and $RGBA$.
3. Set the camera mode to $CUSTOM\_CAMERAS$, and toggle *Export to different layers.*
4. For each camera, expect the first one, set the custom color to black and set its alpha to zero, in order to have a transparent background.
5. Set your cameras culling masks and game object layers like you want.

Available Example: The SeparatedLayersExample scene contains a complete example.

7.1.7 Capture an off-screen scene

You can use the ScreenshotTaker to capture custom cameras to a texture, with your custom cameras being disabled in the scene. If your camera game objects are disabled, the asset will perform an off-screen rendering without enabling them, allowing you to capture something that is not displayed on the gameview.

To prevent any blinking, be sure to set the off-screen camera depth to -1.

Available Example: The OffscreenRenderingExample scene contains a "CaptureCamerasToTextureCanvas Example" gameobject with a CaptureCameraToTextureExample component.

7.1.8 Capture using an UI button

If you are using the ScreenshotManager, you need to call its Capture() method. Select the button, add an onClick event, set the manager as the target and select the Capture() method.

Available Example: The DefaultExample and OffscreenRenderingExample scenes contain several examples to call a capture method, using the ScreenshotManager, or direct script calls to SimpleScreenshotCapture or ScreenshotTaker.

7.2 Common issues

7.2.1 I have a compilation error on iOS

Check that you added the **Photos** framework dependency to the iOS plugin. Please refer to Section 2.3.2 for more details.

7.2.2 Transparency issues with HDRP or URP

Follow the guides aboves 7.1.4, and then do the following.

1. If the transparency is wrong, try disabling some post processing effects, like bloom, tonemapping, or virtual environment.
2. You can recompute the transparency using the "recompute alpha" option, this will try to re-create the alpha channel after the capture process.

7.2.3 Issues with multi display settings

In build, the asset will detect if the application is multi display, and try to capture the good screen automatically. In editor and build, you can add a MultiDisplayCameraCapture component to the camera you want to capture, the asset will capture the screen associated to that camera.

7.2.4 I have some artifacts with temporal anti-aliasing, or other special effects

In editor, using the *GAMEVIEW_RESIZING* mode, if you have any artifact when taking a screenshot at a custom resolution, or a UI element at the wrong size, you can increase the *ScreenshotTaker m_GameViewResizingWaitingFrames* value. The *ScreenshotTaker* component is located on the same game object that the *ScreenshotManager*.

The *m_GameViewResizingWaitingFrames* value specifies how many frame the screenshot taker waits before to take the screenshot after the gameview has been resized. The default value of 2 should be enough for most settings. Increase this number when some elements are not well updated, like GUI, or when you see some post effects artifacts. Post effects like temporal anti aliasing require a value of at least 10.

7.2.5 I can not create screenshots with the ScreenshotTaker

Check that you provide a valid full path as the filename. Please refer to Section 8.1 for more details.

7.2.6 I can not take screenshots on Android

If you get a message saying the folder or file can not be created, check the storage permissions. Please refer to Section 2.3.1 for a detailed configuration guide.

7.2.7 My GameView and layout are doing strange things when I capture a screenshot

Having the GameView deformed or blinking is expected when you use the *GAMEVIEW_RESIZING* capture mode.

For Unity 4.6 to 5.3, the algorithm rescales the GameView window to match the screenshot dimensions, and this undocks the GameView window. To prevent it, the editor layout is saved before the capture and restored after it, creating a sort of "flashing" effect. If this annoys you, you can set *Force Layout Preservation* to false.

With Unity 5.4 and later, the GameView has an inner "scale" which allows the modification of its dimensions without modifying the editor layout. During the capture, its resolution is changed several times before being restored, creating a sort of "blinking".

7.2.8 Nothing Happens when I try to Update the Preview(s)

Sometimes, the renderer may be locked in Editor and refuse to update the preview. This happens rarely, but if you do not have any response from the *ScreenshotManager*, stop and play the game.

Do not hesitate to contact me if you can reproduce this bug, so I can correct it.

8 PROGRAMMING

8.1 API

8.1.1 ScreenshotTaker

The ScreenshotTaker is the component used to capture the screenshots to textures. You can use it directly to capture a texture by calling one of its capture coroutine.

```
/// <summary>
/// Captures the current screen at its current resolution.
/// The texture will be resized if needed to match the capture settings.
/// </summary>
public IEnumerator CaptureScreenToTextureCoroutine(Texture2D texture,
                        bool captureGameUI = true,
                        ScreenshotTaker.ColorFormat colorFormat =
                            ScreenshotTaker.ColorFormat.RGB,
                        bool recomputeAlphaMask = false)



/// <summary>
/// Captures the scene with the specified width, height, using the mode
    RENDER_TO_TEXTURE.
/// Screenspace Overlay Canvas can not be captured with this mode.
/// The texture will be resized if needed to match the capture settings.
/// </summary>
public IEnumerator CaptureCamerasToTextureCoroutine(Texture2D texture, int width
    , int height,
                         List<Camera> cameras,
                         int antiAliasing = 8,
                         ScreenshotTaker.ColorFormat colorFormat =
                             ScreenshotTaker.ColorFormat.RGB,
                         bool recomputeAlphaMask = false)



/// <summary>
/// Captures the game with the specified width, height.
/// The texture will be resized if needed to match the capture settings.
/// </summary>
public IEnumerator CaptureToTextureCoroutine(Texture2D texture, int width, int
    height,
                       List<Camera> cameras = null,
                       List<Canvas> canvas = null,
                       ScreenshotTaker.CaptureMode captureMode = ScreenshotTaker
                           .CaptureMode.RENDER_TO_TEXTURE,
                       int antiAliasing = 8,
                       bool captureGameUI = true,
                       ScreenshotTaker.ColorFormat colorFormat = ScreenshotTaker
                           .ColorFormat.RGB,
                       bool recomputeAlphaMask = false)


public IEnumerator CaptureAllCoroutine(List<ScreenshotResolution> resolutions,
                     List<ScreenshotCamera> cameras,
                     List<ScreenshotOverlay> overlays,
                     CaptureMode captureMode,
                     int antiAliasing = 8,
                     bool captureGameUI = true,
```

```
                ColorFormat colorFormat = ColorFormat.RGB,
                bool recomputeAlphaMask = false,
                bool stopTime = false,
                bool restore = true)
```

8.1.2 SimpleScreenshotCapture

You can capture a screenshot using only the SimpleScreenshotCapture. That static class provides several capture methods, that you can choose depending on your needs.

Note that you need to provide a **valid full path** *fileName* to the Screenshot Taker, for instance *C:/Screenshots/myScreenshot.png*. You can use the ScreenshotNameParser.ParsePath() or ScreenshotNameParser.ParseFileName() to get one.

```
        /// <summary>
        /// Captures the current screen at its current resolution, including UI,
            to a texture.
        /// </summary>
        public static void CaptureScreenToTexture(UnityAction<Texture2D>
            onTextureCaptured, ScreenshotTaker.ColorFormat colorFormat =
            ScreenshotTaker.ColorFormat.RGB)


        /// <summary>
        /// Captures the camera to a texture.
        /// </summary>
        public static Texture2D CaptureCameraToTexture(int width, int height,
            Camera camera, ScreenshotTaker.ColorFormat colorFormat =
            ScreenshotTaker.ColorFormat.RGB)

        /// <summary>
        /// Captures cameras to a texture.
        /// </summary>
        public static Texture2D CaptureCamerasToTexture(int width, int height,
            List<Camera> cameras, ScreenshotTaker.ColorFormat colorFormat =
            ScreenshotTaker.ColorFormat.RGB)

        /// <summary>
        /// Captures the current screen at its current resolution, including UI.
        /// The filename must be a valid full name.
        /// </summary>
        public static void CaptureScreenToFile(string fileName,
                                               TextureExporter.ImageFileFormat
                                                   fileFormat = TextureExporter.
                                                   ImageFileFormat.PNG,
                                               int JPGQuality = 100,
                                               bool captureGameUI = true,
                                               ScreenshotTaker.ColorFormat colorFormat
                                                    = ScreenshotTaker.ColorFormat.RGB,
                                               bool recomputeAlphaMask = false)

        /// <summary>
        /// Captures the scene with the specified width, height, using the mode
            RENDER_TO_TEXTURE.
        /// Screenspace Overlay Canvas can not be captured with this mode.
        /// The filename must be a valid full name.
        /// </summary>
```

```
        public static void CaptureCamerasToFile(string fileName,
                                        int width, int height,
                                        List<Camera> cameras,
                                        TextureExporter.ImageFileFormat
                                            fileFormat = TextureExporter.
                                            ImageFileFormat.PNG,
                                        int JPGQuality = 100,
                                        int antiAliasing = 8,
                                        ScreenshotTaker.ColorFormat
                                            colorFormat = ScreenshotTaker.
                                            ColorFormat.RGB,
                                        bool recomputeAlphaMask = false)

        /// <summary>
        /// Captures the game.
        /// The filename must be a valid full name.
        /// </summary>
        public static void CaptureToFile(string fileName,
                                    int width, int height,
                                    TextureExporter.ImageFileFormat fileFormat =
                                        TextureExporter.ImageFileFormat.PNG,
                                    int JPGQuality = 100,
                                    List<Camera> cameras = null,
                                    List<Canvas> canvas = null,
                                    ScreenshotTaker.CaptureMode captureMode =
                                        ScreenshotTaker.CaptureMode.
                                        RENDER_TO_TEXTURE,
                                    int antiAliasing = 8,
                                    bool captureGameUI = true,
                                    ScreenshotTaker.ColorFormat colorFormat =
                                        ScreenshotTaker.ColorFormat.RGB,
                                    bool recomputeAlphaMask = false)
```

8.2 Delegates

If you want to run some custom code before and after the capture processes, you can use the following delegates:

8.2.1 Screenshot Manager

```
public static void onCaptureStartDelegate  ()
```

Is called when the capture starts.

```
public static void onCaptureEndDelegate  ()
```

Is called when the capture ends.

```
public static void onResolutionExportSucessDelegate (ScreenshotResolution res)
```

Is called just after the screenshot is exported.

```
public static void onResolutionExportFailedDelegate (ScreenshotResolution res)
```

Is called just after the screenshot export fails.

8.2.2 Screenshot Taker

```
public static void onResolutionUpdateStartDelegate (ScreenshotResolution res)
```

Is called just before capturing the given resolution.

```
public static void onResolutionScreenResizedDelegate (ScreenshotResolution res)
```

Is called just after resizing the gameview in *GAMEVIEW_RESIZING* mode.

```
public static void onResolutionUpdateEndDelegate (ScreenshotResolution res)
```

Is called just after capturing the given resolution.

9 SUPPORT

Please do not hesitate to contact me if you have a feature request, or any question, issue or suggestion : support@wildmagegames.com.