Greetings!

Thank you for your interest in **Advanced FPS Counter**, tiny counters-on-steroids set for Unity3D!

Please read carefully through this document to know more about each counter and all plugin features in general.

ADVANCED FPS COUNTER

v. 1.4.4
[version history]

Full API documentation can be found here:
codestage.net/uas_files/afps/api/

Contents

Installation and setup

IMPORTANT:

To avoid different compatibility issues and errors, always remove previous version from project Assets before updating.
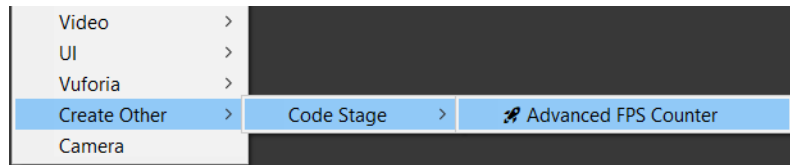
As you import plugin, you have such options for adding Advanced FPS Counter to your scenes:

- Drag CodeStage/AdvancedFPSCounter/Prefabs/Advanced FPS Counter prefab to the Scene Hierarchy.

    Hint:
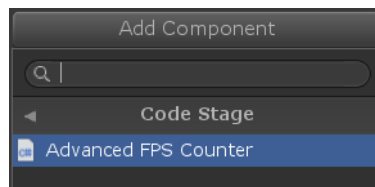
    You also may find prefabs from the Examples/Prefabs folder useful for the VR or World Space setup.

- Add from the Scene Hierarchy context menu (or Game Object menu):

Create Other > Code Stage > Advanced FPS Counter

- Add AFPSCounter component to any existing Game Object using Add Component button or just drag dropping it from Project Browser:

"Add Component" button > Code Stage > Advanced FPS Counter

- Add to scene from code using AFPSCounter.AddToScene() API

    Hint:

    You need to add namespace CodeStage.AdvancedFPSCounter to usings in order to work with plugin from code.

    Here is a simple example:

```csharp
// place this line right at the beginning of your .cs file!
using CodeStage.AdvancedFPSCounter;

// ...
private void Start()
{
    // instantiates AFPSCounter in scene if it not exists
    // with disabled keepAlive option
    AFPSCounter.AddToScene(false);

    // changes spacing between counters
    AFPSCounter.Instance.CountersSpacing = 1;
}
```

AFPSCounter draws all its data using Labels, which are the uGUI Texts placed within the automatically generated Canvas with Screen Space - Overlay mode.
Thus, in order to see the counters, you do not need to perform any additional actions in most cases.

IMPORTANT:

If you will place AFPSCounter somewhere inside the existing Canvas, auto-generated one will inherit options from the parent. This is useful when you wish to use plugin in some special environment, such as a **World Canvas** in the **VR** project.

Please refer to the *Troubleshooting* section below if you do not see the counters after performing initial setup.

Plugin features in-depth

Common Settings
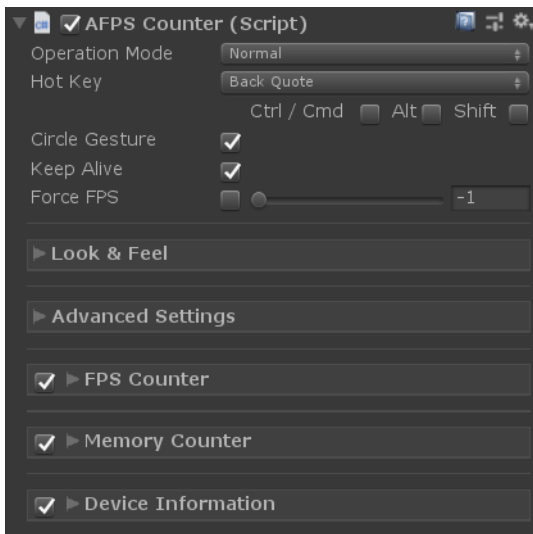
Operation Mode: controls how AFPSCounters runs.
- Disabled mode is used to remove all counters and stop all internal processes except the global hotkey / gesture listener.
- Normal mode should be used in most cases: counters are visible and operate as intended.
- Background mode allows reading all enabled counters data using Scripting API without uGUI output thus avoiding any additional resources usage. Useful for performance / hardware stats, for hidden performance monitoring and quality settings suggestion, etc.

Hot Key: customizable global hotkey to show / hide plugin using Disabled / Normal Operation Mode switch. Does not affect Background Operation Mode.

Circle Gesture: complements Hot Key and does same job.
Touch screen / touchpad or hold left mouse button and make ~2 circles in any direction to switch Operation Mode.

Keep Alive: allows keeping Advanced FPS Counter's Game Object on new scene load (using DontDestroyOnLoad).

> **IMPORTANT**:
> The topmost root Game Object will be kept alive if AFPSCounter is placed on the nested Game Object.

Force FPS: allows trying your game at specified frame rate, may help to debug your game behavior / physics on slow devices; specified frame rate is not guaranteed though (and may not work at all).

Look & Feel settings

Auto Scale: controls Canvas's CanvasScaler uiScaleMode.
When disabled, uses ConstantPixelSize with specified **Scale Factor**.
When enabled, uses ScaleWithScreen.

Labels Font: font used to render counters in Labels (uGUI Texts).
Monospace font with Bold style support usually looks great.

Font Size: controls size of the used Font.

Line Spacing: controls space between lines in single Label using uGUI Text LineSpacing.

**Counters Spacing:** empty lines count between different counters in same Label.

Padding Offset: offset for the Labels placement. Relative to the Anchor position set at the *per-counter settings* and automatically changes sign for the right and bottom Labels.

Pixel Perfect: controls own Canvas PixelPerfect property in overlay modes.

Background: colored background effect with customizable **Color** and **Padding**. Costs only 1 Draw Call. Fair deal for better visibility.

Shadow & **Outline:** text rendering effects with customizable **Color** and **Distance**. These are resources-heavy (especially outline) so use with caution on mobile devices.
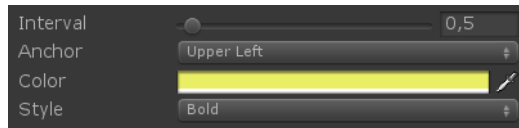
Advanced settings
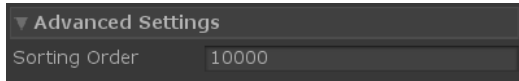
Sorting Order: controls Canvas's sortingOrder property.
With higher value Canvas gets closer to the user while sorting with other canvases.

Counters settings

Common per-counter settings

All counters have same per-counter settings:

Interval: delay between counter updates, in seconds. Recommended value lays in 0.5 - 1 secs range. Only updatable counters have this setting.

Anchor: specifies counter screen placement within one of these Labels:
Upper Left, Upper Center, Upper Right
Lower Left, Lower Center, Lower Right
It allows covering all corners, top and bottom of the screen.
All counters with same anchor are drawn within one Label, thus, one Label may contain several Counters. Labels content is refreshed (and reassembled) only if any containing Counter is marked as dirty (has changed value since last update). It allows avoiding unnecessary waste of resources.
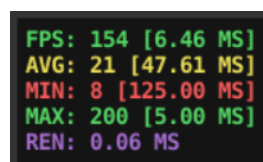
Color: counter text color. FPS Counter has more complex text coloration with three coloration ranges - for normal, warning and critical value.

Style: counter text style with these values:
Normal, Bold, Italic, Bold and Italic

Note: Not all fonts support these styles.

FPS (Frames Per Second) Counter

Game View

This Counter shows current **FPS** and many additional optional metrics.

Here are all Counter settings in same order as you see them in Inspector:

Common settings section, with more flexible **Color Range** selection for three FPS value ranges: normal, warning and critical.

Realtime FPS: shows actual FPS. In some cases, you might want to see only Average FPS for example, so you can disable realtime FPS here.

- **Milliseconds:** shows an approximate time spent to process one frame. Available for all FPS metrics.

Average FPS: shows an averaged FPS using accumulates samples. Notable sub-settings are:

- **Samples:** controls how much of last FPS readouts to use for average calculation. More samples - more slow and smooth average value change. Use 0 value to collect all FPS samples since last **Average FPS** reset (optimized calculation will be used in such case).
- **Auto Reset:** resets this FPS metric and cleans accumulated samples on scene load.

**MinMax FPS:** shows minimum and maximum FPS.
Notable sub-settings are:

- **Delay:** "pre-warm" amount of **Interval** updates to wait before starting Min /Max registration. Allows skipping initialization FPS spikes.
- **Auto Reset:** resets this FPS metric on scene load.

Render Time: shows approximate time spent by Camera(s) to render last frame. Requires usage of the helper AFPSRenderRecorder component. Add it to Game Objects with Camera(s) you wish to measure render time for, or use the **Auto add** option to let AFPSCounter automatically add AFPSRenderRecorder component to the Game Object with "Main Camera" tagged Camera if your scene has it.
Helper component will be re-added to the Main Camera's Game Object after scenes switch if **Keep Alive** option is used (see below).

Note: does not take into account Image Effects and IMGUI.

Memory Counter
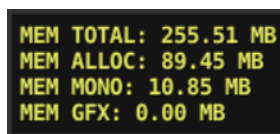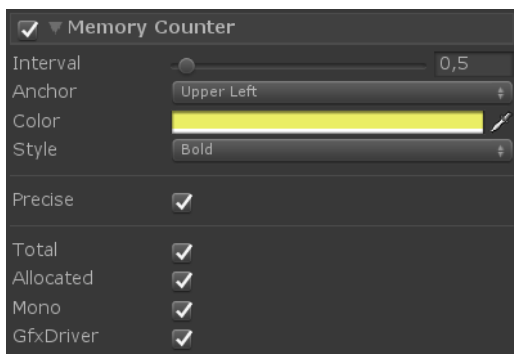
Game View

This Counter shows different memory usage information.

Here are all Counter settings in same order as you see them in Inspector:

Common settings section.

Precise: allows to output memory values with additional accuracy using float values instead of int ones. Requires some extra resources though, thus try to avoid using it in conjunction with low update interval.

Total: shows total **private** memory amount, **reserved** by OS for the application. Other applications cannot use this memory. Applications may ask OS to reserve some amount of private memory, which is usually a bit more than they really use at that moment. This allows making new allocations faster if reserved memory chunk has enough space for them.

Allocated: shows amount of private memory currently actually **used** by application. In other words - how much memory all your textures, sounds, etc. use.

Mono: shows amount of memory, allocated by the managed objects, such as UnityEngine.Object (and everything derived from it), your classes, etc. This memory is called managed because the lifecycle of objects is managed by the runtime - it automatically allocates and frees memory for the objects. Garbage Collector frees memory allocated by unreferenced objects automatically so you do not need to manage them yourself. IL2CPP builds have managed objects and Garbage Collector as well.

GfxDriver: shows amount of the allocated memory for the graphics driver. This feature requires Unity 2018 or newer and works in Development builds or Editor only.

Device Information Counter

Game View (PC)
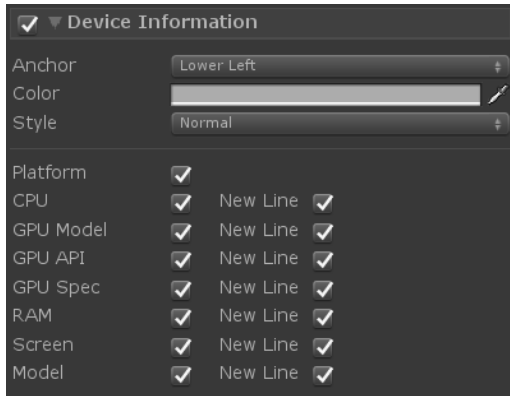
```
OS: Windows 10  (10.0.0) 64bit [WindowsPlayer]
CPU: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz [12 cores]
GPU: NVIDIA GeForce GTX 1080
GPU: Direct3D 11.0 [level 11.1] [Direct3D11]
GPU: SM: 5.0, VRAM: 8079 MB
RAM: 32696 MB
SCR: 2560x1440@70Hz [window size: 1024x768, DPI: 120]
Model: System Product Name (System manufacturer)
```
Game View (PC)

```
OS: Android OS 8.1.0 / API-27 (OPM4.171019.021.P1/4820305) [Android]
CPU: ARMv7 VFPv3 NEON [4 cores]
GPU: Adreno (TM) 530
GPU: OpenGL ES 3.2 V@258.0 (GIT@2941438, I916dfac403) (Date:10/03/17) [OpenGLES3]
GPU: SM: 5.0, VRAM: 1024 MB
RAM: 3754 MB
SCR: 1080x1920@60Hz [window size: 1080x1920, DPI: 420]
Model: Google Pixel
```
Game View (Mobile)

This Counter shows different information about current device and environment where your app or game is running.

Here are all Counter settings in same order as you see them in Inspector:

Common settings section without interval setting since this counter updates only once on start.

Platform: outputs Operating System name with version (if possible) and runtime platform type.

CPU: outputs CPU model and cores count (including virtual cores from Intel's Hyper Threading).

GPU Model: outputs GPU model name.

GPU API: outputs graphics API name, version and type (if possible).

GPU Spec: outputs graphics shader model (if possible) and total VRAM (if possible).

RAM: outputs total RAM on current device in Megs.

Screen: outputs resolution with refresh rate, current window size and screen DPI (if possible).

Model: outputs device model (if possible). Mostly handy on the mobile devices.

Tips

- All plugin features and counters values are available from code through the public APIs. It allows to make a totally custom UI for the counters - you may use any non-standard UI framework, output to charts and graphs, etc.

- Please, take a look at the ExampleScene (at the Examples folder) to see how to work with plugin from code and check how you can alter AFPSCounter's settings and setup at runtime.

- While Unity didn't introduce better way to save values set at runtime, you may easily tune counters (colors, intervals, etc.) in Play mode and save adjusted values using these steps:
  - enter Play mode
  - tune AFPSCounter component settings in inspector
  - right-click on the AFPSCounter component's header and select "Copy Component"
  - exit Play mode
  - right-click on the AFPSCounter component's header and select "Paste Component Values"
  This technique works for any other components as well.

- To enable and disable whole AFPSCounter from code just use **AFPSCounter.Instance.OperationMode** property (switch it between AFPSCounterOperationMode.Disabled and AFPSCounterOperationMode.Normal).

- You may add any text to any counter using **SetAppendAndUpdate()** API. For example:

  `AFPSCounter.Instance.fpsCounter.SetAppendAndUpdate("<b>Temp:</b> <color=#A76ED1>40</color>");`

  Counter will be immediately updated after calling this API. To remove text, just pass null. Rich Text is supported.

Compatibility

Plugin should work fine on any platform, including WebGL and VR and it was tested on these platforms:

PC (Win, Mac, Linux, WebGL), **iOS** (incl. Google Cardboard), **Android** (incl. Gear VR, Google Cardboard, Vive Focus), **Windows Universal Platform**.

In addition, customers reported it as working on these:

Wii U devkit (thx Black Lodge Games), **Xbox One** (thx Yaroslav Bakhvalov).

Please report if plugin does not work for you on some specific platform.

All features should work fine with **any** stripping level, **IL2CPP** runtime, **.NET 4.6** compiler and Assembly Definitions.

Troubleshooting

- **I cannot see AFPSCounter on the screen**
  - Make sure you have added it to the current scene (either in Editor or from code).
  - Make sure it is enabled and active.
  - Check console for any error messages or warnings.
  - Make sure your Canvas Sorting Order has value lower than AFPSCounter's Sorting Order advanced setting value.
  - If you are working with VR, place AFPSCounter inside an existing Canvas with World Space Render Mode to make it visible in your VR device (see Examples/Prefabs/VR Example).

- **Something is wrong with the font I use for the counters**
  - Unity had a bug, which led to the exclusion of the font styles in case you are using different font files for the different styles. Please try updating your Unity version to check if it fixes this issue for you.

- **I see GC allocations from AFPSCounter**
  - Relatively low amount of GC allocations is expected in the Normal operation mode as it uses UGUI Text which do not allow to use char[] instead of string to prevent GC allocations. You may use Background operation mode (which does not produce GC allocations while running) to make your own GC-free visualization.

Final words from author

I hope you will find Advanced FPS Counter helpful and it will save some of your priceless time!
Please, leave your reviews at the plugin's Asset Store page and feel free to drop me bug reports, feature suggestions and other thoughts on the forum or via support contacts!

AFPSCounter links:

Asset Store | Web Site | Forum | API | YouTube

Support contacts:
E-mail: support@codestage.net
Other: codestage.net/contacts

Follow for updates and news:
@codestage_net
@codestage

Best wishes,
Dmitriy Yukhanov
Asset Store publisher
codestage.net

P.S. #0 I wish to thank my family for supporting me in my Unity Asset Store efforts and making me happy every day!
*P.S. #1 I wish to say huge thanks to **Daniele Giardini** (DOTween, HOTools, Goscurry and many other happiness generating things creator) for priceless feedback on this plugin!*