

Readme – Pro TexturePlayer 1.2.2

Pro TexturePlayer provides a set of powerful image playback, display, and loading APIs for your professional apps/games. Easy load and display image sequences in your Unity app and game.

The best solution for displaying image banners, posters that load with URL and local file path, or load images from the local folder and play like a slideshow or simply display them like a gallery app.

This package is integrated with several important features that tackle many issues for loading, displaying and managing images for your app. Optimized for better performance, ease of use, and help prevent app crashes when a large number of images are loaded.

Highlights

- **Image Cache Management**

Cache the downloaded images on local storage (Application Paths), so your app doesn't need to re-download the images every time between runs.

- **Memory Usage Optimization**

Significantly less memory usage when multiple images loaded in one player. Ensure better stability and performance.

- **Texture Resize**

Resize the loaded textures without changing the origin files, by inputting the target width and height in the Play method.

- **Texture Player Controls**

Play, Pause, Resume, Stop, PlayForward, PlayBackward, Display particular image by texture index, examples of Callbacks for handling events such as image click event.

One of our popular assets - [Image Loader](#) is also included in this package~

** Note that this asset supports load images from public accessible URL or local path, and cache on local storage with Read/Write permission. It is not a native plugin for Load/Save files from the Android Gallery or iOS Photos app.*

Features

Pro Texture Player

- Play images with list/array of URLs or local paths.
- Play textures list/array.
- Play images from the local folder with the folder path (the application paths, not the native gallery),
- Play images like a slideshow, or just simply display them.
- Playback Controls: Play, Pause, Resume, Stop, PlayForward, PlayBackward, etc.
- Customizable playback framerate.
- Trigger callbacks/events with the image ID of the particular image (e.g. click on the image to trigger an event by its ID).
- Build-in image resize feature, easily constraint the size of all the images by specifying the target width and height. Unifying your image size on loaded and saving memory usage.
- Optimize Memory Usage option, significantly reducing memory usage for loaded textures.
- Supports display on UGUI Image & RawImage, Renderers(Cube, Plane, Sphere etc.);
- Displays on materials/objects that supports Texture2D or Sprite. Easy to extend with the PlayerComponent.

Image Loader

- ImageLoader: load single image, support multiple instances for loading images.
- ImageQueuedLoader: load multiple images with a single API, supports limiting the number of loaders, split loading process to avoid blocking the main thread when loading a large number of images.
- ImageBatchLoader: wrapped the ImageLoader & ImageQueuedLoader for easily loading multiple images. Load multiple images with a single API.
- Load image(s) from local(application paths), and load image(s) with Url(s) from web using the same API.
- Detect actual image Extension Name and MIME Type.
- Flexible sequential file naming system (for batch download)
- Easy, powerful images cache management, with rich settings:
 - Retry option
 - Timeout option
 - Cache mode: NoCache, UseCached, Replace
 - Cache file limit per folder
 - Cache as per URL, or Cache using specific Filename/Sequence
 - Particular cache(Save/Load) directory and folder
 - Sequential file naming formats
 - Min. keep file time & file expire(max.) time

Bonus Features

EasyIO Plugin: supports saving & loading image, text, class object, and file byte array on different platforms. Provides the ability for your app to write files in IndexedDB, so the cache management system can work correctly on WebGL.

(1) Pro TexturePlayer Components

1. Base: IMBX_PlayerComponent

The base class of the Pro TexturePlayer. Includes the Play methods, player control methods, callback methods, and clear method. Also uses to hold the textures and related variables. Extend with this component to support displaying textures on UI components or other materials.

2. IMBX_PlayerImage

Extends the PlayerComponent, supports display textures on UGUI Image.

3. IMBX_PlayerRawImage

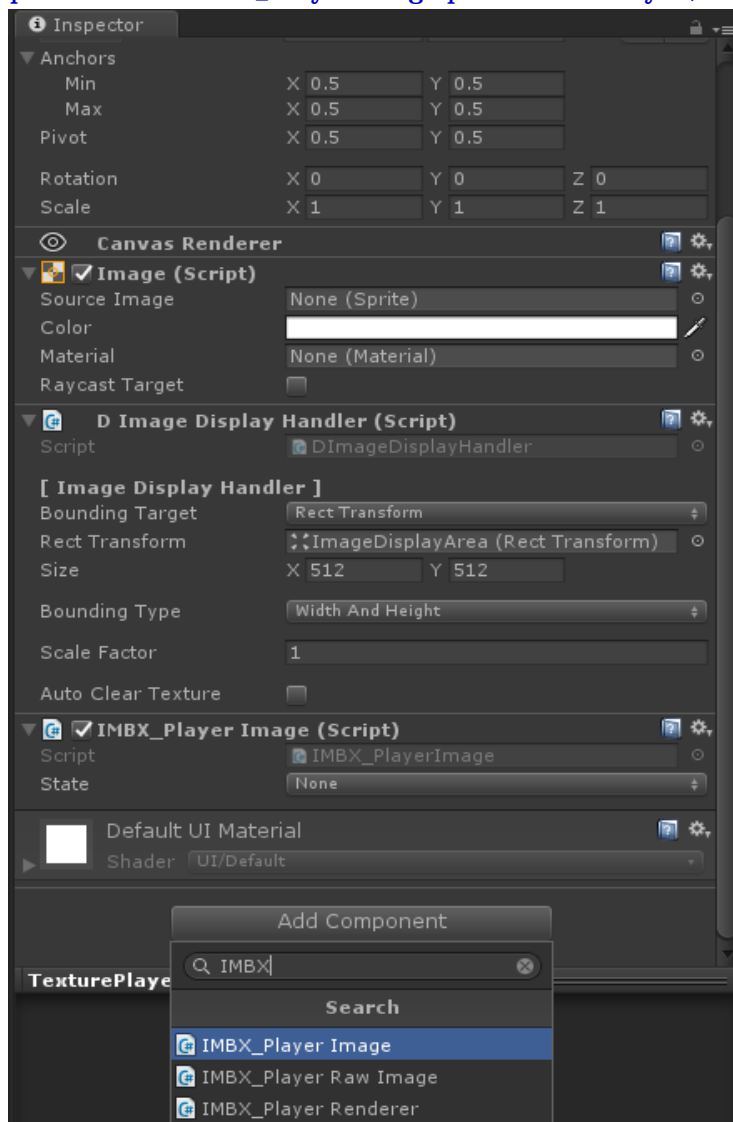
Extends the PlayerComponent, supports display textures on UGUI RawImage.

4. IMBX_PlayerRenderer

Extends the PlayerComponent, supports display textures on Mesh Renderers such as Plane, Cube, Sphere etc.

Add a component (e.g. [IMBX_PlayerImage](#)) to an object in the scene or prefab, add a reference in your script and link the object:

```
public IMBX.IMBX_PlayerImage proTexturePlayer;
```



(2) Texture Player API

After adding a player component to the target object in your scene or prefab, reference it to your script(see the provided example scenes). Then simply call the methods of the component to play and control the player.

e.g. `proTexturePlayer.Play(...);`

Play Methods:

`Play(List<string>:imagePaths, float:fps, bool:optimizeMemoryUsage, bool:ensureInSequence);`

`Play(List<string>:imagePaths, float:fps, bool:optimizeMemoryUsage, int:maxUnifyWidth, int:maxUnifyHeight, bool:ensureInSequence);`

`Play(Texture2D[]:textures, float:fps, bool:optimizeMemoryUsage, int:maxUnifyWidth, int:maxUnifyHeight);`

`Play(Texture2D[]:textures, float:fps, bool:optimizeMemoryUsage);`

`Play(string:fullFolderPath, float:fps, bool:optimizeMemoryUsage, int:maxUnifyWidth, int:maxUnifyHeight, Enum.ImageFormatFiltering:imageFormatFilter, int:maxNoOfImage, int:startIndex);`

`Play(string:fullFolderPath, float:fps, bool:optimizeMemoryUsage, Enum.ImageFormatFiltering:imageFormatFilter, int:maxNoOfImage, int:startIndex);`

Callbacks:

`SetOnPlayStartedCallback(Action<int, int>:onPlayStartedCallback);`

This callback will be fired when the texture player started (once only). Returns the texture width and height with the callback.

`SetOnPlayingCallback(Action<TextureIMBX, int>:onPlayingCallback);`

This callback will be fired per frame when the player is playing. Returns the currently played texture and frame index with the callback.

`SetOnLoadingCompleteCallback(Action:onLoadingCompleteCallback);`

This callback will be fired when all the loading process finished.

Player Control Methods:

`float PlaybackSpeed {get; set;}`

`bool IgnoreTimeScale {get; set;}`

`Pause();`

`Resume();`

`Stop();`

`PlayForward();`

`PlayBackward();`

`Reverse();`

`SetDisplayFrame(int:frameIndex);`

`Clear();`

(3) Storage Cache & Loader Management

The ImageLoader functions are integrated into Pro TexturePlayer, it provides highly customizable settings for loading images and handles cache management per folder to cache images in the device storage. This can be done by setting appropriate parameters in the **LMGT** object (**LoaderManagement**) before loading images, like the below example.

Example:

```
proTexturePlayer.LMGT.CacheMode = ImageLoader.CacheMode.UseCached;
```

- **Cache Mode**

[ImageLoader.CacheMode](#) : [CacheMode](#)

(The behavior for handling Load and Cache files. **NoCache**: do not auto save the image; **UseCached**: use the locally cached file if exist; **Replace**: download and replace the locally cached file is exist.)

- **Cache Directory**

[string](#) : [CacheDirectory](#) (Getter)

(The root directory for loading and storing/caching image files. Supports the application paths only, e.g. Application.persistentDataPath.)

[FilePathName.AppPath](#) : [CacheDirectoryEnum](#)

(The enum that determines which application path to load and save(cache) file to.)

- **Cache Folder Path**

[string](#) : [CacheFolderPath](#) (Getter)

(The cache folder path that combined by the root CacheDirectory and FolderName.)

- **Sub-Folder**

[string](#) : [FolderName](#)

(The sub-folder under cache directory for loading and storing(caching) files to.)

- **Cache as per URL**

[bool](#) : [CacheAsPerUrl](#)

(If 'true', for URL start with 'http', the loader will cache the image as per URL address.)

- **Batch FileName Prefix (for batch loader)**

[string](#) : [FileNamePrefix](#)

(The filename prefix for storing(caching) the image files. The final filename will be combined by this prefix, separator, and the index together.)

- **File Extension**

[string](#) : [FileExtension](#)

(Suggest use .jpg or .png only)

- **Number of Digits for the Index follow the Filename Prefix (for batch loader)**

[uint](#) : [FileIndexFormatDigitsCount](#)

(e.g. For digits count = 5, and index = 12, the result index string becomes 00012)

- **File Name Starting Index (for batch loader)**

[uint](#) : [FileNameStartingIndex](#)

(Set this value to set an offset for the filename index. The default starting index is 0.)

- **Separator text between the File Name and Index (for batch loader)**
[string : FileNameAndIndexSeparator](#)
(Please use filename friendly characters only)
- **Max. Files Per Folder**
[uint : MaxCacheFilePerFolder](#)
(The maximum number of files to cache per folder. Auto delete the older files if exceed the set limit. Zero means no limit.)
- **Min. Keep File Time**
[uint : MinTimeForKeepingFiles](#)
(e.g. m_MinTimeForKeepingFiles = 86400, 86400 seconds = 1 day
For images in the target folder that modified within 1 day will not be auto deleted.)
- **File Expire Time**
[uint : MaxTimeForKeepingFiles](#)
(e.g. m_MaxTimeForKeepingFiles = 864000, 864000 seconds = 10 days
Delete images modified more than 10 days ago.)
- **Loading Retry**
[uint : LoadingRetry](#)
(Number of times to retry when a loading failed. Retry per second until the retry value is Zero.)
- **Loading Timeout**
[float : LoadingTimeOut](#)
(The time duration for waiting for a loading/downloading, stop and kill the loader if time exceeds.)
- **Allow/Avoid Loading the same URL (at the same time)**
[bool : AllowDuplicateDownload](#)
(If true, allows downloading the same URL using multiple loaders, else it will not start a new download if that URL is being downloaded.)
- **Keep/Drop the loaded Image Data**
[bool : KeepImageData](#)
(If 'true', reserve the loaded image byte array data in the Result/CacheItem objects.)
- **Check if an image exists in the cache folder of this LMGT object:**
[bool: HasStorageCache\(string:filename\);](#)

(4) Extra

Set waiting time for a particular frame:

```
proTexturePlayer.SetFrameDelay(int:frameIndex, float:delayTimeInSeconds);
```

Set to display a particular frame:

```
proTexturePlayer.SetDisplayFrame(int:frameIndex);
```

Get the current frame index:

```
proTexturePlayer.m_CurrentFrameIndex;
```

Get total frame count:

```
proTexturePlayer.m_TotalFrame;
```

Add a texture to the player:

```
proTexturePlayer.RemoveTexture(int:index);
```

Remove a texture from the player:

```
proTexturePlayer.AddTexture(Texture2D:texture, float:interval);
```

(5) Image Loader Tools (Included)

Load images from local storage(in-app, Application paths), or download images from the web with image full URL. Details please refer to the Readme document in the ImageLoader folder, or visit our [website](#).

THANK YOU

Thank you for using this package!

For any question and bug report please contact us at swan.ob2@gmail.com.

Remember to rate this asset on the Asset Store. Your review is always appreciated, and very important to the development of this asset!

[Review And Rating](#)

Visit our asset page to find out more!

<https://www.swanob2.com/assets>

SWAN DEV