Readme – ImageLoader 1.1.6

Load/Download image(s) with URL and local path using the same API.
Supports batch loading images, queued batch loading images, and the ability to specify the number of loaders to use.
Provides powerful cache management feature for caching your images, and handles file naming, folders, time to keep and delete images, etc.

**** Note that this asset supports load images from public accessible URL, and local storage with Read/Write permission only. It does not support Android Gallery or iOS Photos.***

Features

- 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.
- Easy, powerful images cache management.
- Flexible sequential file naming system (for batch download)
- Flexible 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

- Example scenes included.

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) Image Loader

Load/Download single image from Web or local storage(in-app, Application paths).

- **Create a new ImageLoader:**
  ImageLoader loader = ImageLoader.Create(uint:filesPerFolder,
  FilePathName.AppPath:cacheDirectory, string:gameObjectName);

- **Start to load/download the image:**
  API 1: loader.Load(uint:index, string:url, LoaderManagement:lmgt,
  Action<Texture2D, uint>:onComplet);
  or
  API 2: loader.Load(uint:index, string:url, Action<Texture2D, uint>:onComplete,
  uint:retry, float:timeOut);
  or
  API 3: loader.Load(uint:index, string:url, string:fileName, string:folderName,
  CacheMode:cacheMode, Action<Texture2D, uint>:onComplete,
  uint:retry, float:timeOut);

- **File Extension & MIME Type**: Image Loader supports getting actual file extension name and MIME type, even the file extension name is unknown or incorrectly set. You can check the following variables when loading completed in the OnComplete callback.
  string: DetecedFileExtension
  string: DetectedFileMime

(2) Image Batch Loader

Load/Download single or multiple images from Web or local (in-app, Application paths) storage.

- **Create a new ImageBatchLoader:**
  ImageBatchLoader loader = new ImageBatchLoader(uint:filesPerFolder,
  FilePathName.AppPath: cacheDirectory);

- **Setting filename formats (Optional):**
  loader.SetFileNameFormat(uint:fileIndexFormatDigitsCount,
  uint:fileNameStartingIndex, string:fileNameAndIndexSeparator);

- **Start to load/download the images:**

  Non Queued Loader methods (For FPS don't care, static loading screen, etc):
  API 1: loader.Load(List<string>:imageUrls, LoaderManagement:lmgt,
  Action<Results>:onComplete, Action<Result>:onProgress);
  or
  API 2: loader.Load(List<string>:imageUrls, Action<Results>:onComplete,
  uint:retry, float:timeOut, Action<Result>:onProgress);
  or
  API 3: loader.Load(List<string>:imageUrls, string:filenamePrefix, string:folderName,
  ImageLoader.CacheMode:cacheMode, Action<Results>:onComplete,
  Action<Result>:onProgress, uint:retry, float:timeOut);

  Queued Loader methods (Supports limit the number of loaders.):
  API 1: loader.Load_Queue(uint:maxLoaderNum, List<string>:imageUrls,
  LoaderManagement:lmgt, Action<Results>:onComplete,
  Action<Result>:onProgress);
  or
  API 2: loader.Load_Queue(uint:maxLoaderNum, List<string>:imageUrls,
  Action<Results>:onComplete, uint:retry, float:timeOut,
  Action<Result>:onProgress);
  or
  API 3: loader.Load_Queue(uint:maxLoaderNum, List<string>:imageUrls,
  string:filenamePrefix, string:folderName,
  ImageLoader.CacheMode:cacheMode, Action<Results>:onComplete,
  Action<Result>:onProgress, uint:retry, float:timeOut);

- **File Extension & MIME Type** are included in the Result/Results object that returns with the OnProgress and OnComplete callbacks.
  e.g.: result.m_DetectedFileMime;
  result.m_DetectedFileExtension;
  results.GetExtensionName(int:index);
  results.GetMimeType(int:index);

(3) Cache Management

The ImageLoader asset provides highly customizable settings for image cache management per folder. To modify the cache settings, just input appropriate values when calling the Create(or Constructor) & Load methods. Or modify the **LMGT**(**LoaderManagement** object) just after the ImageLoader or ImageBatchLoader is created.

Example:

```
loader.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 if exist.)

- **Cache Directory**
  string : CacheDirectory (Getter)
  (The path for saving and loading images if the cache feature is enabled. Supports the application paths only, eg. Application.persistentDataPath.)

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

- **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
  (eg. 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 deletes the older files if exceed this limit. Zero means no limit.)

- **Min. Keep File Time**
  uint : MinTimeForKeepingFiles
  (eg. 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
  (eg. m_MaxTimeForKeepingFiles = 864000, 864000 seconds = 10 days
  Delete images modified more than 10 days ago.)

(4) Manage Cache Files (manually)

ImageLoader only automatically performs cache management(save & delete files) in the specified folder during a Load method is executed with cache mode **UseCached** or **Replace**. It is done by executing the global method **ManageCacheFiles** in the ImageLoader class, inside the Load method. You can also call this method manually as needed (e.g. at the app start/quit).

Example:

```
ImageLoader.ManageCachedFiles(fullFolderPath, maxFilePerFolder,
        minTimeForKeepingFiles, maxTimeForKeepingFiles, fileExtension);
```

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