FileManagement Class

Cross-platform file IO control without Plug-ins.

eToile

(eToile 2016) V: 1.5

Index

Introduction

Thanks for purchasing FileManagement, this class is designed to be simple and lightweight, so you will not need to learn or implement more than its useful interfaces.
This product is just that: a class, so you can add it to your project without any risks.
You also can access the full source code.
Saving and reading is as fast as the platform can admit, because interpretations and parsing are maintained at minimum.
The example scene allows testing most of the FileManagement functions.
There is a very important feature in FileManagement that allows access PersistentData and StreamingAssets as a single drive. This allows a completely safe cross-platform drive access.

Class Description

FileManagement is a static class, that means you don't have to create/instantiate a FileManagement object, just write FileManagement dot (.) the interface you need.

The FileManagement class uses compiler directives to chose the right functions for each platform. So, there are different versions for the next functions: SaveRawFile(), ReadRawFile() and DeleteFile().

This three functions are platform dependent, so they are used by every other function into this same class.
The three main groups are:

- UNITY_WINRT
- UNITY_WEBGL
- Everything else

There are no any special considerations when exporting to other platforms, neither any special considerations when uploading to digital markets.
Just switch platform from "Build settings" dialog on Unity editor.

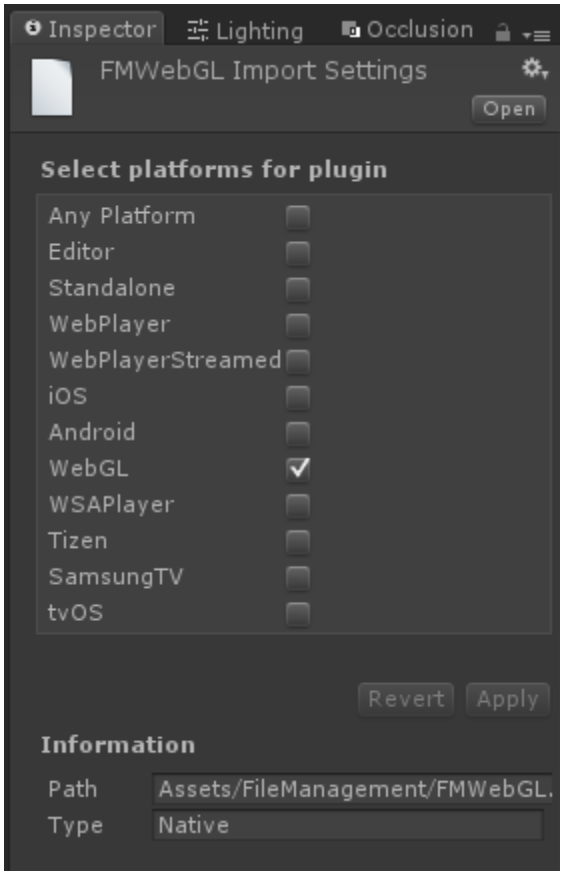
Class Integration

To integrate this class to your project you must include the three main files that are used in this product:

- "FileManagement.cs".
- "FMWebGL.jslib".
- "StreamingAssetsIndexer.cs".

The FileManagement.cs file contains the main FileManagement class. FMWebGL.jslib is a plug-in needed for WebGL exports. This file can be edited with any text editor, it is not a real plug-in, it's some javascript extra functionality needed to save content into browser's data base.

The plug-in importer should look like this:



Note that you don't need to put the plug-in into a "Plugins" folder.

The StreamingAssetsIndexer.cs file runs only in the editor generating the StreamingAssets index automatically.

PlayerPrefs replacement

FileManagement implements the same PlayerPrefs functions, so you only need to rename PlayerPrefs with FileManagement.

This is the list of equivalent functions:

```
public static void DeleteAll()
public static void DeleteKey(string key)
public static float GetFloat(string key, float defaultValue = 0.0F)
public static int GetInt(string key, int defaultValue = 0)
public static string GetString(string key, string defaultValue = "")
public static bool HasKey(string key)
public static void Save()
public static void SetFloat(string key, float value)
public static void SetInt(string key, int value)
public static void SetString(string key, string value)
```

The Save() function has no effect in FileManagement due to there is no parsing of values into a single file. The interface exists just to avoid compilation issues when porting an existing application.

This is the list of non standard functions:

```
public static bool GetBool(string key, bool defaultValue = false)
public static double GetDouble(string key, double defaultValue = 0)
public static void SetBool(string key, bool value)
public static void SetDouble(string key, double value)
```

These functions works in the same way that PlayerPrefs does but adds some missing functionality.

FileManagement Public Interfaces

This is the complete definition of FileManagement public interfaces.

SaveRawFile()

```
public static void SaveRawFile(string name, byte[] content, bool enc = false,
bool fullPath = false)
```

Use a key, ID or file name to identify your file.

The enc argument enables or disables the encryption functionality.

The fullPath argument allows to treat the provided name as a path.

The name can include a relative or absolute destination path. If that path doesn't exists it will be created automatically.

NOTE: This function is used by every other function that writes to disk.

This example saves a binary file:

```
byte[] byteArray = {10, 20, 30, 40, 50};
FileManagement.SaveRawFile("data.bin", byteArray);
```

ReadRawFile()

```
public static byte[] ReadRawFile(string name, bool enc = false, bool checkSA = true,
bool fullPath = false)
```

Use a key, ID or file name to identify your file. It returns empty content if the file do not exists.

The enc argument enables or disables the encryption functionality.

The checkSA argument allows FileManagement to search the "StreamingAssets" folder if the requested file is not found.

The fullPath argument allows to treat the provided name as a path.

This example reads an encrypted text file (the content value is decrypted):

```
byte[] content = FileManagement.ReadRawFile ("data.txt", true);
```

DeleteFile()

```
public static void DeleteFile(string name, bool fullPath = false)
```

Deletes the file. It fails if the access is denied using fullPath.

The fullPath argument allows to treat the provided name as a path.

This example deletes the file:

```
FileManagement.DeleteFile("data.txt");
```

ImportAudio()

```
public static AudioClip ImportAudio(string file, bool enc = false, bool checkSA = true,
bool fullPath = false)
```

Imports an AudioClip from file. The wav format is the only widely supported by all platforms, for other formats check the Unity documentation (Or try the test application in your desired platform).

This function implements a WAV file parser to allow WebGL support. Only 16bit WAV files are supported (mono or stereo, 22K, 44K, etc.).

FileExists()

```
public static bool FileExists(string name, bool checkSA = false, bool fullPath = false)
```

Checks if the file exists.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.

The fullPath argument allows to treat the provided name as a path.

This example checks if a file exists:

```
if (FileManagement.FileExists("data.txt"))
      Debug.Log("[FileManagement] This file exists.");
```

SaveFile()

```
public static void SaveFile<T>(string name, T content, bool enc = false,
bool fullPath = false)
```

Saves any C# type of variable, and the following UnityEngine types:

Vector2, Vector3, Vector4, Quaternion, Rect, Color & Color32.

The enc argument enables or disables the encryption functionality.

The fullPath argument allows to treat the provided name as a path.

This example saves two types of data:

```
FileManagement.SaveFile("IntData", 12);
FileManagement.SaveFile("StringData", "Example data", true);  // Encrypt
```

ReadFile()

```
public static T ReadFile<T>(string name, bool enc = false, bool checkSA = false,
bool fullPath = false)
```

Reads any C# type of variable, and reads the following UnityEngine types: Vector2, Vector3, Vector4, Quaternion, Rect, Color & Color32.

The enc argument enables or disables the encryption functionality.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.

The fullPath argument allows to treat the provided name as a path.

This example reads two different types of file:

```
int a = FileManagement.ReadFile<int>("IntData");
string a = FileManagement.ReadFile<string>("StringData", true);      // Decrypt
```

SaveArray()

```
public static void SaveArray<T>(string name, T[] content, char separator = (char)0x09,
bool enc = false, bool fullPath = false)
public static void SaveArray<T>(string name, System.Collections.Generic.List<T> content,
char separator = (char)0x09, bool enc = false, bool fullPath = false)
```

Saves any one dimension Array or List of any ReadFile supported type.

You can specify a custom separator as a char argument, default separator is the horizontal tabulator.

The enc argument enables or disables the encryption functionality.

The fullPath argument allows to treat the provided name as a path.

This example saves an array of strings separated by a semicolon:

```
string[] myArray = {"one", "two", "three"};
FileManagement.SaveArray("MyArray.csv", myArray, ';');
```

ReadArray()

```
public static T[] ReadArray<T>(string name, char separator = (char)0x09, bool enc = false,
bool checkSA = true, bool fullPath = false)
public static T[] ReadArray<T>(string name, string[] separator, bool enc = false,
bool checkSA = true, bool fullPath = false)
```

Reads any one dimension Array of any ReadFile supported type.

You can specify a custom separator as a char argument (default is horizontal tabulator) or as an array of string values (no default).

The enc argument enables or disables the encryption functionality.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.

The fullPath argument allows to treat the provided name as a path.

This example reads an array of strings separated by a semicolon:

```
string[] myArray = FileManagement.ReadArray<string>("MyArray.csv", ';');
```

ReadList()

```
public static System.Collections.Generic.List<T> ReadList<T>(string name,
char separator = (char)0x09, bool enc = false, bool checkSA = true, bool fullPath = false)
public static System.Collections.Generic.List<T> ReadList<T>(string name,
string[] separator, bool enc = false, bool checkSA = true, bool fullPath = false)
```

Reads any one dimension List of any ReadFile supported type.
You can specify a custom separator as a char argument (default is horizontal tabulator) or as an array of string values (no default).
Use this function to import CSV files.
The enc argument enables or disables the encryption functionality.
The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.
The fullPath argument allows to treat the provided name as a path.

This example reads an array of strings separated by a semicolon:

```
string[] myArray = FileManagement.ReadArray<string>("MyArray.csv", ';');
```

ReadAllLines()

```
public static string[] ReadAllLines(string name, bool enc = false, bool checkSA = true,
bool fullPath = false)
```

Reads all the lines from a text file. It uses "\r\n" , "\n" and "\r" literals for line splitting (Window, OSX and Unix compatible).
The enc argument enables or disables the encryption functionality.
The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.
The fullPath argument allows to treat the provided name as a path.

This example imports an image file into a texture from the StreamingAssets folder:

```
Texture2D texture = FileManagement.ImportTexture("image.jpg");
```

ImportTexture()

```
public static Texture2D ImportTexture(string file, bool enc = false, bool checkSA = true,
bool fullPath = false)
```

Imports a JPG or PNG image file from disk into a Unity texture.
The enc argument enables or disables the encryption functionality.
The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.
The fullPath argument allows to treat the provided name as a path.

This example imports an image file into a texture from the StreamingAssets folder:

```
Texture2D texture = FileManagement.ImportTexture("image.jpg");
```

ImportSprite()

```
public static Sprite ImportSprite(string file, bool enc = false, bool checkSA = true,
bool fullPath = false)
```

Imports a JPG or PNG image file from disk into a Unity sprite.
The enc argument enables or disables the encryption functionality.
The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested file is not found.
The fullPath argument allows to treat the provided name as a path.
This example imports an image file into a sprite from the StreamingAssets folder:

```
Sprite sprite = FileManagement.ImportSprite("image.jpg");
```

SaveJpgTexture()

```
public static void SaveJpgTexture(string name, Texture texture, int quality = 75,
bool enc = false, bool fullPath = false)
```

Saves a Texture or Texture2D to a file encoded in JPG format. The quality parameter determines the file compression, the default file compression is 75.
The enc argument enables or disables the encryption functionality.
The fullPath argument allows to treat the provided name as a path.
This examples saves textures into files:

```
FileManagement.SaveJpgTexture("texture.jpg", renderer.material.mainTexture);
FileManagement.SaveJpgTexture("texture.jpg", gameObject.GetComponent<Sprite>().texture);
```

SavePngTexture()

```
public static void SavePngTexture(string name, Texture texture, bool enc = false,
bool fullPath = false)
```

Saves a Texture or Texture2D to a file encoded in PNG format.
The enc argument enables or disables the encryption functionality.
The fullPath argument allows to treat the provided name as a path.
This examples saves textures into files:

```
FileManagement.SavePngTexture("texture.png", renderer.material.mainTexture);
FileManagement.SavePngTexture("texture.png", sprite.texture);
```

AddLogLine()

```
public static void AddLogLine(string name, string content, bool deleteDate = false,
bool enc = false, bool fullPath = false)
```

This adds a single line of text to an existing file saving also date and time. Used for log files and error tracking.
The deleteDate variable disables the automatic printing of formatted date and time for each new line.
The enc argument enables or disables the encryption functionality.
The fullPath argument allows to treat the provided name as a path.
This examples saves textures into files:

```
FileManagement.SavePngTexture(renderer.material.mainTexture, "texture.png");
FileManagement.SavePngTexture(sprite.texture, "texture.png");
```

AddRawData()

```
public static void AddRawData(string name, byte[] content, bool enc = false,
bool fullPath = false)
```

This adds a chunk of byte data to an existing file. If the requested file doesn't exists, a new file will be created.

The enc argument enables or disables the encryption functionality.

The fullPath argument allows to treat the provided name as a path.

This examples saves textures into files:

```
FileManagement.SavePngTexture(renderer.material.mainTexture, "texture.png");
FileManagement.SavePngTexture(sprite.texture, "texture.png");
```

DirectoryExists()

```
public static bool DirectoryExists(string folder, bool checkSA = true,
bool fullPath = false)
```

Checks the existence of a directory.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested path is not found.

The fullPath argument allows to treat the provided name as a path.

This example checks if a directory exists:

```
if (FileManagement.DirectoryExists ("Test1"))
      Debug.Log("[FileManagement] This folder exists.");
```

CreateDirectory()

```
public static void CreateDirectory(string name, bool fullPath = false)
```

Creates a new directory. Fails if the access is denied using fullPath.

The fullPath argument allows to treat the provided name as a path.

This example creates a new directory:

```
FileManagement.CreateDirectory("Test1");
```

DeleteDirectory()

```
public static void DeleteDirectory(string name, bool fullPath = false)
```

Deletes a directory and its content including sub-directories.

Fails if the access is denied using fullPath.

The fullPath argument allows to treat the provided name as a path.

This example deletes an existing directory:

```
FileManagement.DeleteDirectory("Test1");
```

EmptyDirectory()

```
public static void EmptyDirectory(string name = "", bool filesOnly = true,
bool fullPath = false)
```

Deletes the directory content. By default deletes only files.

Fails if the access is denied using fullPath.

The fullPath argument allows to treat the provided name as a path.

This example deletes the whole content of a folder:

```
FileManagement.EmptyDirectory("Test1", false);
```

ListFiles()

```
public static string[] ListFiles(string folder, bool checkSA = true, bool fullPath = false)
public static string[] ListFiles(string folder, string[] filter, bool checkSA = true,
bool fullPath = false)
```

Returns all of the file names contained in the requested folder.

Fails if the access is denied while using fullPath.

The file list can be filtered using the filter array, make sure to include the dot (.) in the extension names like this:

```
string[] filter = {".wav", ".mp3", ".ogg"};
```

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested path is not found.

The fullPath argument allows to treat the provided name as a path.

This example requests the folder content:

```
string[] fileNames = FileManagement.ListFiles("Test1");
```

ListDirectories()

```
public static string[] ListDirectories(string folder, bool checkSA = true,
bool fullPath = false)
```

Returns all of the folder names contained in the requested folder.

Fails if the access is denied while using fullPath.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested path is not found.

The fullPath argument allows to treat the provided name as a path.

This example requests the folder content:

```
string[] folderNames = FileManagement.ListDirectories("Test1");
```

ReadDirectoryContent()

```
public static System.Collections.Generic.List<byte[]> ReadDirectoryContent
(string folder, bool enc = false, bool checkSA = true, bool fullPath = false)
```

Returns all of the files contained in the requested folder as a List of byte arrays. The files are added to the List in the same order that are listed with the ListFiles function.

Fails if the access is denied using fullPath.

The enc argument enables or disables the encryption functionality.

The checkSA argument allows FileManagement to search the StreamingAssets folder if the requested path is not found.

The fullPath argument allows to treat the provided name as a path.

This example requests the folder content:

```
List<byte[]> files = FileManagement.ReadDirectoryContent("Test1");
string[] fileNames = FileManagement.ListFiles("Test1");
```

CopyFile()

```
public static void CopyFile(string source, string dest, bool checkSA = true,
bool fullPathSource = false, bool fullPathDest = false)
```

Copies a file from source to dest. The destination path can include a new name for the copied file.

The checkSA parameter only affects the source path.

This example copies a file:

```
FileManagement.CopyFile ("data.txt", "NewFolder/dataCopy.txt");
```

CopyDirectory()

```
public static void CopyDirectory(string source, string dest, bool checkSA = true,
bool fullPathSource = false, bool fullPathDest = false)
```

Copies a folder from source to dest. The destination path can include a new name for the copied file.

The folder content is copied too, including its sub-directory content.

If the dest path doesn't exists, It will be created automatically.

This example copies a file:

```
FileManagement.CopyFile ("data.txt", "NewFolder/dataCopy.txt");
```

Move()

```
public static void Move(string source, string dest, bool fullPathSource = false,
bool fullPathDest = false)
```

Moves a file or folder from source to dest. The folders are moved with all its content.

This example moves a folder and all its content:

```
FileManagement. Move ("NewFolder/Test2", "NewFolder2/Test2");
```

Rename()

```
public static void Rename(string source, string dest, bool fullPathSource = false,
bool fullPathDest = false)
```

This function is the same as Move(), you can use it to rename files and folders.

This example moves a folder and all its content:

```
FileManagement.Move ("NewFolder/Test2", "NewFolder2/Test2");
```

GetParentDirectory()

```
public static string GetParentDirectory(string path = "")
```

Returns a valid path but removing the file or folder.

This example request a parent folder:

```
string parentPath = FileManagement.GetParentDirectory ("Test1/Test2");
```

Now parentPath's value is "Test1".

Combine()

```
public static string Combine(string path1, string path2)
```

Returns a valid path combining correctly both paths.

This example combines a path and a file:

```
string path = FileManagement.Combine("Test1\\Test2", "icon2.png");
```

Now path's value is "Test1/Test2/icon2.png".

NormalizePath()

```
public static string NormalizePath(string path)
```

Returns a valid path, correcting possible errors in the string. The FileManagement class uses slashes, deletes the ending slash in a path and replaces double slashes.

This example returns a valid directory path:

```
string path = FileManagement.NormalizePath ("Test1\\Test2\\");
```

Now path's value is "Test1/Test2".

GetFileName()

```
public static string GetFileName(string path)
```

Returns the last name of the path. It can be a file or a folder.

This example returns the file name only:

```
string name = FileManagement. GetFileName ("Test1\\Test2\\data5.txt");
```

Now name's value is "data5.txt".

GetFileExtension()

```
public static string GetFileExtension(string path)
```

Returns the file extension of the path (if any). If there is no extension it will return an empty string.

This example returns the file extension:

```
string extension = FileManagement. GetExtension ("Test1\\Test2\\icon2.png");
```

Now path's value is ".png".

FileManagement Private Interfaces

This is the complete definition of FileManagement private interfaces.

CheckNameOnIndex()

```
private static bool CheckNameOnIndex(string name, string type)
```

Checks the name into de StreamingAssets folder automatic Index. Emulates the FileExists and DirectoryExists functions for Android and WebGL builds. There are two slightly different versions of this function for each platform (Android and WebGL).

GetNamesOnIndex()

```
private static string[] GetNamesOnIndex(string name, string type)
```

Checks the name into de StreamingAssets folder automatic Index and returns the list of the associated names. Emulates the ListFiles and ListDirectories functions for Android and WebGL builds. There are two slightly different versions of this function for each platform (Android and WebGL).

CustomParser()

```
private static T CustomParser<T>(string content)
```

This function is responsible of converting the text saved data into the correct data again. It supports: Every C# type, Vector2, Vector3, Vector4, Quaternion, Rect, Color & Color32.
This parser is used into ReadFile and ReadList.

FilterPathNames()

```
private static string[] FilterPathNames(string[] names)
```

This function does the inverse as GetParentDirectory because returns the file or folder name removing the parent.
Its use is internal, so it's private.

SortPathNames()

```
private static string[] FilterPathNames(string[] names)
```

This function sorts the names by alphabet.
Its use is internal, so it's private.

Encrypt()

```
private static byte[] Encrypt(byte[] data, byte[] key)
```

This is the function that makes the encryption of a file content. It only works with byte arrays. This function is local and you don't need to use it directly. This function can be updated with your favorite encryption algorithm.

Decrypt()

```
private static byte[] Decrypt(byte[] data, byte[] key)
```

This is the function that makes the decryption of a file content. It only works with byte arrays. This function is local and you don't need to use it directly. This function can be updated with your favorite encryption algorithm.

XorEncryptDecrypt()

```
private static byte[] XorEncryptDecrypt(byte[] data, byte[] key)
```

Performs XOR encryption and decryption.

Experimental interfaces

The experimental interfaces are not 100% supported by all platforms. FileManagement will print in console if the selected platform hasn't the requested functionality.

AesEncrypt()

```
private static byte[] AesEncrypt(byte[] data, byte[] key)
```

Performs AES encryption. Not supported on Windows Store.

AesDecrypt()

```
private static byte[] AesDecrypt(byte[] data, byte[] key)
```

Performs AES decryption. Not supported on Windows Store.

OpenFolder()

```
public static void OpenFolder(string path = "", bool fullPath = false)
```

Opens the requested folder in the default file system. This functions works in standalone builds only (Windows, Linux, OSX).

ObjectToByteArray()

```
private static byte[] ObjectToByteArray(object obj)
```

Serializes an object to a byte array allowing to be saved to disk with SaveRawFile.

Please note that not every variable/class type can be serialized directly. That's the reason this is not the default save method.

This function is not supported in Windows Store builds.

ByteArrayToObject()

```
public static object ByteArrayToObject(byte[] arrBytes)
```

Deserializes a byte array into an object allowing to be loaded from disk with ReadRawFile.

Please note that not every variable/class type can be serialized directly. That's the reason this is not the default save method.

This function is not supported in Windows Store builds.

ListLogicDrives()

```
public static string[] ListLogicDrives()
```

Get the list of the available logic drives. This function is not supported in Windows Store builds.

WARNING: Not widely tested across all platforms.

AES Encryption

```
#define USE_AES
```

Define or comment this directive to allow AES Encryption.
The AES algorithm uses a fixed length key, it doesn't affects the XOR algorithm due to that algorythm uses any key length.
Don't forget to set your own new key!
IMPORTANT: AES is not supported on Windows Phone platforms.

StreamingAssets indexer

The StreamingAssetsIndexer.cs script works automatically while in editor and generates a file/sub-directory index that can be navigated once the application was built for Android or WebGL.

The index file is "FMSA_Index" and is saved automatically into your StreamingAssets folder to be included within your final build.
The StreamingAssetsIndexer.cs script detects modifications in the file system and regenerates always the index to ensure that every existing file or sub-directory can be detected correctly.
The StreamingAssets folder is inaccessible in Android and WebGL builds due to it is not a real folder, that is the main reason that this Index was implemented.
The Index interpretation is fully integrated within the `FileManagement` class.

Do not modify the StreamingAssets folder in your final build, or the index will not match the content resulting in access errors.
Do it always from Unity editor, then build.

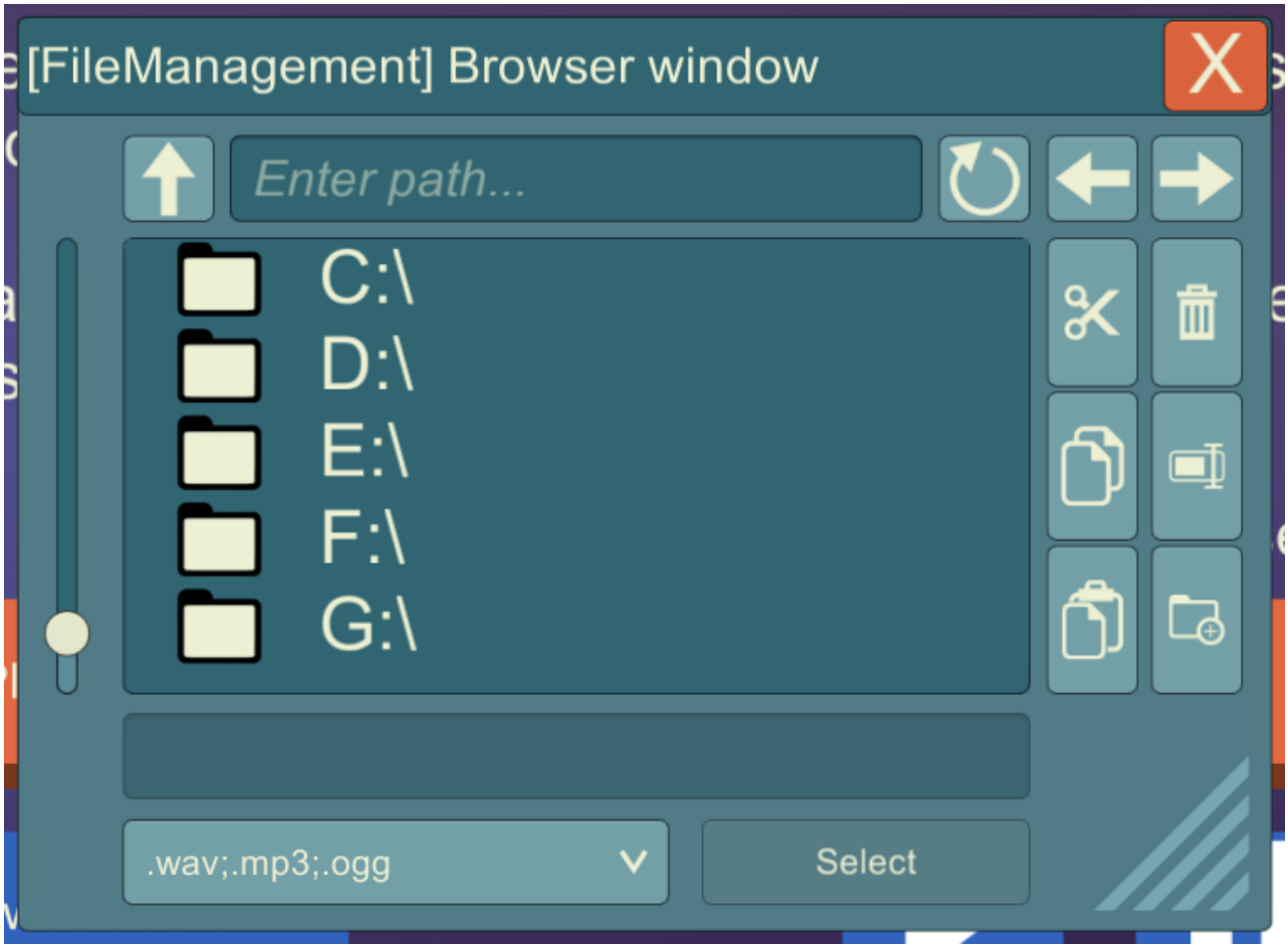
NOTE: This script is not included in your final build.

Please note that the StreamingAssets folder is read only in some platforms, but you have write access in others. You have to use `Application.streamingAssetsPath` in fullPath mode.

FileBrowser prefab

There is a prefab included within this package that allows you to browse the file system and to select a file/folder.

The browser implements some useful functionality that allows it to be used without programming skills:

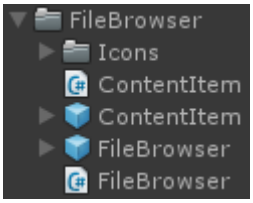


It has built in Back, Fwd, Cut, Copy, Paste, Delete, Rename and New folder buttons.

The file browser is completely developed in Unity UI, so you can customize it completely at your will using the Editor tools.

The use of images are maintained at minimum to allow export easily. The only image resources that uses the prefab are contained in the "Icons" folder.

This is what you need:



The scripts used in this prefab are dependent of the `FileManagement` class, so you will need it to be imported too.

The "Icons" folder contains the image resources.

The FileBrowser doesn't opens the file, it just returns the path to be treated in the main application when it closes.

The way you create a FileBrowser is as follows:

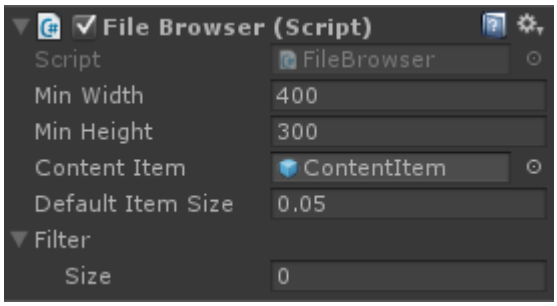
```
public GameObject fileBrowser;  // Drag the FileBrowser prefab here (in the editor).

// Create a FileBrowser window (with default options):
public void OpenFileBrowser()
{
    GameObject browserInstance = GameObject.Instantiate(fileBrowser);
    browserInstance.GetComponent<FileBrowser>().SetBrowserWindow(OnPathSelected);
    // Add file extension filters (optional):
    string[] filter = { ".wav", ".mp3", ".ogg" };
    browserInstance.GetComponent<FileBrowser>().SetBrowserWindowFilter(filter);
}

// You should use this function signature in order to receive properly:
void OnPathSelected(string path)
{
    // Do something with the returned path.
}
```

The FileBrowser destroys itself once a file is selected or the window is closed. You can access some public functions through the browserInstance variable.

You can double-click or double-tap the items to navigate and chose.

FileBrowser.cs

There are a few parameters that can be customized directly from editor in the FileBrowser.cs script:



public int minWidth: The minimum width of the window when resizing (pixels).

Public int minHeight: The minimum height of the window when resizing (pixels).

float defaultItemSize: The default size for items if the size slider is disabled or deleted (percentage of the canvas height in units).

public GameObject ContentItem: This is the reference to the item prefab.

string[] filter: This string List defines the file extensions to be shown in the file browser. Any other file extension will be discarded. You can set this property by coding through the SetBrowserWindowFilter() method (it has several overloads).

Here is the definition of the interface used to set the FileBrowser instance (you will find it in FileBrowser.cs):

```
public void SetBrowserWindow(OnPathSelected selectionReturn, string iniPath = "",
bool fullPath = false, string selectionMode = "F", bool save = false, string lockPath = "")
```

The OnPathSelected delegate has a specific signature in order to be called properly:

```
void MyFunction(string)
```

The ReturnSelectedFile() function executes the delegate, returning the selected path/file and closes the windows.

The iniPath argument sets the first folder to be shown when the window becomes visible.

The fullPath argument allows the browser to navigate outside the unified PersistentData+StreamingAssets path without restrictions.

If you navigate until the root in fullPath mode, the Windows standalone builds will show the available logical drives.

The selectionMode argument allows the browser to select files or folders depending on its value: "F" for files, "D" for directories or logical drives.

The save argument allows the InputField of the selected item to allow writing a custom file or folder name.

The lockPath argument forces the browser to stay always into that directory and sub-directories not allowing further navigation (no matter what navigation mode was selected).

Important: This script assumes that is attached to the root GameObject of the prefab (Uses the RectTransform of the canvas).

The other most important function is the one that sets the file extension filter:

```
public void SetBrowserWindowFilter(List<string> newFilter)
public void SetBrowserWindowFilter(string[] newFilter)
public void SetBrowserWindowFilter(string newFilter)   // Use extensions separated by ";".
```

You can use any of the three overloads to set the file extension list to filter the rendered content. You can set the filter dynamically at any time and it will also set the file extension

Dropdown for you.

File extensions should include the dot before the extension, and extensions can have any length: ".jpg", ".unity". You should use the string method like this: ".jpg;.unity".

Special characters as "*" or "?" are not allowed.

There are some other very useful functions:

Cut: This button calls Cut() remembering the file/folder to cut. It will show a warning a message if the file/folder is read only.

Copy: This button calls Copy() remembering the file/folder to copy.

Paste: This button calls Paste() copying or moving the previously selected file or folder and all its content.

Delete: This button calls PromtDeleteSelection() allowing file/folder deletion. It will show a warning a message if the file/folder is read only.

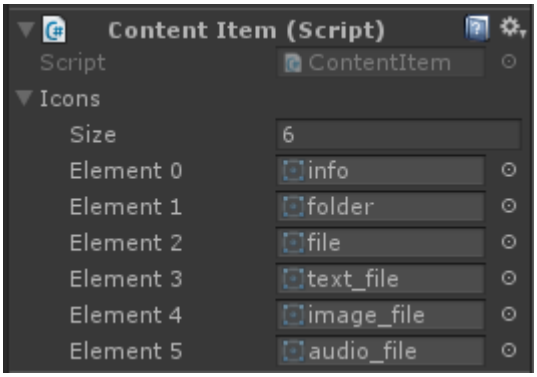
Rename: This button calls PromptForRename() allowing rename a file or folder. It will show a warning a message if the file/folder is read only.

New folder: This button calls PromtNewFolderName() allowing create a new folder. You can create nested folders using a path notation ("Folder1/Folder2").

ContentItem.cs

This script controls the behavior of the rendered items.

You can modify the icons shown in items from here (in the prefab) adding or replacing them in the icons array of Sprite elements.



The icons rendered depending on the file extension are determined by the SetItem method in the ContentItem.cs script. Feel free to modify and add your own file extensions and icons.

Customize the browser window

The file browser is completely built with the Unity UI, so you can use the standard tools to customize the window at your will.

Before start customizing the prefab, please make a copy with Ctrl+D and rename or move as needed. Duplicate the ContentItem prefab too, and once renamed or moved, assign its reference to the ContentItem variable into your new copy of file browser prefab.

There are no scripts attached to the elements of the browser window, everything is connected from FileBrowser.cs.

You just have to keep in mind to use some special names in its hierarchy to allow the FileBrowser.cs script to find the elements that it uses to work properly.

Those elements are:

BrowserWindow: This is the main container panel. Everything is contained by this element (it is the window itself).

InputCurrentPath: This InputField contains the current directory being rendered. It can be edited at any time.

ContentWindow: This ScrollRect is the list controller of rendered items.

ContentWindow>ViewPort>Content: This is the main container of the rendered items. The ContentItem prefabs are added automatically to its Transform. It has a VerticalLayoutGroup component to arrange items.

InputSelection: This InputField contains the name of the selected item. When in save mode, this InputField can be edited to set a custom file or folder name.

ButtonSelect: This is the selection Button, it can be enabled or disabled depending on the current selection.

ButtonSelect>Text: the Text of the ButtonSelect changes depending on the action that can be executed: "Select", "Open" or "Save".

Confirmation: This is the confirmation pop-up with a Label and two buttons: ButtonOk and ButtonCancel.

NewName: This is the text input pop-up with a Label, an InputField called InputNewName and two buttons: ButtonOk and ButtonCancel.

ErrorMessage: This is the message pop-up with a Label, an Image and a button: ButtonOk.

SizeSlider: This Slider allows modifying the item size dynamically. Can be deleted if not used.

FilterDropdown: This Dropdown lists the available file extensions to be rendered in the content view. The first item is always a list with all the available extensions. The extensions will be added separately in the successive items. Can be deleted if not used (filters will still working internally).

Excepting ButtonSelect, every button or interactive element has an event that points to a method contained in FileBrowser.cs, so you can delete them if they are not needed.

Known Issues

- There is no stright compatibility with the legacy interfaces used for reading and writing cookies in Web Browsers. Nevertheless you will find those interfaces as `private` in the `FileManagement` class so you can still using them if needed (WebGL only).

- To grant SDCard access in Android you must enable the option: BuildSettings>PlayerSettings>OtherSettings>WriteAccess = External (SDCard).

- There is no free write access to the SD card across all Android devices. That's an OS limitation (You should adapt your software accordingly if you need this feature).

- Android file system is case sensitive (Windows is not).

- Listing logical drives is not supported across all platforms.

Contact

If you need some new interfaces or if you find some errors in this documentation or the application, don't hesitate on send me an email: jmonsuarez@gmail.com

I you find that the test version is not the same version that you downloaded from the AssetStore, please send me your invoice number and I will send you back the last FileManagement version (new version normally is into approval process).

Please, once you have tested this product, take a minute of your time to write a good review in the Unity Asset Store, so you will help to improve this product:

https://www.assetstore.unity3d.com/en/#!/content/67183

Thanks.