# UniFileBrowser

Welcome to UniFileBrowser! This provides a fully-featured file browser for use on desktops and mobile (WebGL has security restrictions that prevent local file access). There are several Unitypackages included:

- **UniFileBrowser** (on the Unity Asset Store, this is imported automatically): the core UniFileBrowser functionality, provided as a DLL. Note that the UniFileBrowser DLL is the only thing that's strictly necessary for UniFileBrowser to function. You can uncheck the "UniFileBrowser Assets" folder when importing the UniFileBrowser package if you already have your own GUISkin set up. The default skin for UniFileBrowser matches the default Unity GUISkin in looks.
- **UniFileBrowserExample:** an example scene that you can use to test various functions.
- **AlternateGUISkin:** an example of a different GUISkin that you can use as-is, or examine to see how GUISkins are handled.
- **UniFileBrowserSource:** this is the source code, included in case you want to make modifications. See "Changing From DLL to Source" on page 2.

Note that UniFileBrowser does not do any actual loading or saving of data. It's a file browser only. What you do with the files that are selected is up to you, since data handling depends entirely on the application you are writing. If you're unfamiliar with file IO, see the System.IO.File docs on MSDN: http://msdn.microsoft.com/en-us/library/system.io.file.aspx

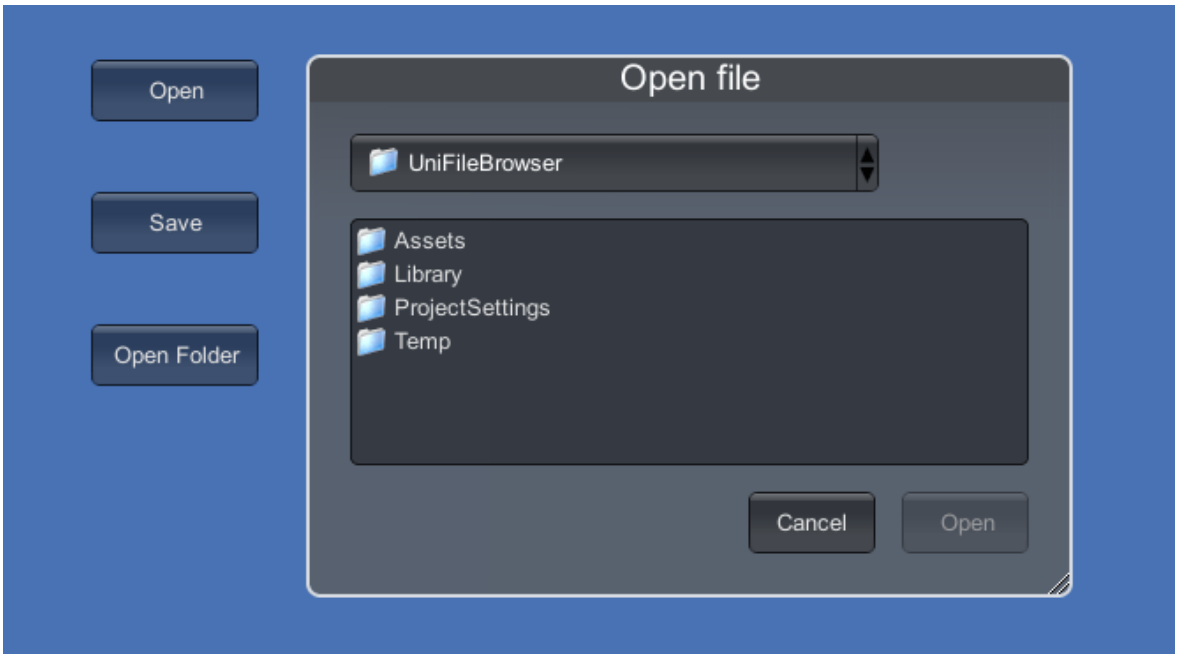
# Contents

# Example scene



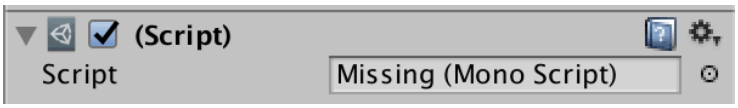
For an example scene demonstrating usage, you can import the **UniFileBrowserExample.unitypackage** file. The example scene is called **UniFileBrowserExample**. It uses a simple script called UniFileBrowserExample, which is provided in both Unityscript and C#. When this scene is opened and run, you'll see three buttons: Open, Save, and Open Folder. When one of these is clicked, it opens up the UniFileBrowser window with the appropriate open or save actions. You can browse through files and "open" or "save" a pretend file (or folder in the case of "Open Folder"), so you can see how it works. No file is actually opened or saved in this example; only the selected file or folder names are displayed temporarily.

**WARNING:** there's an option to enable a Delete button, and even in the UniFileBrowserExample scene, the Delete button is "live" and will really delete files. There's a confirmation dialog, but still, be careful about using this. If you enable the Delete button, it's very strongly recommended that you should also enable the LimitToInitialFolder option, which will limit the scope of possible damage by preventing navigation to other folders.




# Changing From DLL to Source

In the event that you want to edit the source code, you can import the UniFileBrowserSource package. This conflicts with the DLL, so you'll need to delete the DLL after importing the source. If you already had any objects in your project with UniFileBrowser attached, deleting the DLL will cause a missing reference:




To fix this, open the Plugins/UniFileBrowser folder in the project view, then drag the UniFileBrowser script onto the appropriate script reference in the inspector.

Changing from source to DLL is essentially the same process, but you'll need to open the DLL and drag the UniFileBrowser object onto the reference.

# General File Window Usage 

In order for UniFileBrowser to work, the UniFileBrowser DLL or script must be attached to some object in the scene. It's probably simplest if it's attached to an empty GameObject, but it can be attached to anything (but make sure that the object it's attached to is not destroyed, of course). For example, in Unity, select the **GameObject -> Create Empty** menu item, followed by the **Component -> Scripts -> UniFileBrowser** menu item. It will survive scene changes, and will prevent more than one instance of itself from existing. When UniFileBrowser is attached to an object, you can see a number of options available as public variables in the inspector (see the next section).

Note that UniFileBrowser uses specific file names to auto-populate several textures in the UniFileBrowser inspector, as well as the GUISkin. Specifically, the highlight, window tab, and message window textures, and drive, file and folder icons. This way, adding the UniFileBrowser script or DLL will automatically set them up without having to drag and drop anything. (Also, using the Reset function restores them to the default values.) These textures must be located in a Textures folder in a Resources folder (not necessarily UniFileBrowser Assets/Default skin, though that's where they are by default). The names are "UFBdriveicon", "UFBfileicon", "UFBfoldericon", "UFBhighlight", "UFBwindow2", and "UFBwindowtab". The GUISkin is "UniFileBrowserGUISkin". Case is not important. If these files are missing or renamed, the fields will not be auto-populated, but other files can be drag-and-dropped into the appropriate spots.

Actual file window operation at runtime should be pretty obvious to anyone familiar with typical file browsers. The window can be dragged with the title bar, and resized by dragging the lower-right corner. The popup menu at the top allows you to navigate to folders earlier in the current directory path.

Files are normally presented in two groups, folders listed first and files listed second, with each group sorted alphabetically. If the "Show Volumes" option is checked, then there are three groups, with the first being a list of volumes (on OS X) or drive letters (on Windows). If the list of files can't fit in the window, the rest can be seen by using the scroll bar or mouse wheel. Files can be opened by either single-clicking them and then selecting "open", or by double-clicking them. Files can be saved with the same methods, or by typing in a new name. Trying to save over an existing file will bring up a confirmation dialog that asks the user to cancel or replace. The optional delete button allows deletion of files. (It's recommended not to have this enabled unless the "Limit To Initial Folder" option is active.)

Full keyboard control is also implemented: up and down arrow keys will move the highlighted file up and down respectively. Holding alt while using the up and down arrow keys will jump to the top or bottom of the list. Holding Command/Apple (OS X) or Control (Windows/Linux) while pressing the up arrow will go to the next higher directory level, and pressing down with a folder selected will enter that folder. The Escape key will either cancel a confirmation dialog or close the file browser window with no action, whichever is appropriate. Likewise, Return/Enter will OK a confirmation, select a highlighted file, or open a highlighted folder, whichever is appropriate.

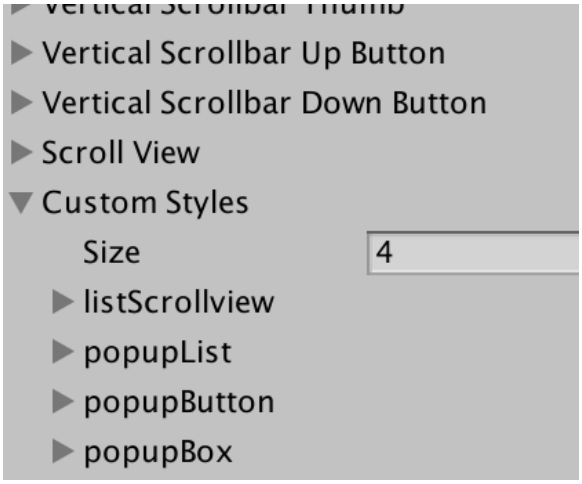
If the "Allow Multi Select" option is checked, then multiple files can be selected for opening. Holding down Shift and clicking selects a range of file names, and holding down Command/Control (depending on OS) and clicking adds or removes individual file names.

If the "OpenFolderWindow" function is used, then a Select button will appear next to the popup menu, and clicking this button will return the path to the folder that appears in the popup menu.

# GUISkin



The file browser window can be re-skinned as usual by using a GUISkin. See the Unity documentation on GUI skins if you're unsure about this process. However, there are a few extras you should be aware of. Namely, there are four custom styles that UniFileBrowser relies on, which must be present in the GUI skin. In the inspector, Custom Styles is an array entry at the bottom of a custom GUISkin list. The size should be 4 (or more if you're using your own custom styles for other things), and the entries should be named "listScrollview", "popupList", "popupButton", and "popupBox". You can look at the included GUISkin to see how these are set up, but the important parts are listed below:




**listScrollview:** This is used to show the folders and files in the file browser list. In the supplied GUISkin, it's similar to the default GUI.Label style except with tighter spacing.

**popupList:** This is similar to the listScrollview style, but the Hover and On Hover entries here should have a highlight texture for the Background, so users can see what they're highlighting in the folder popup list.

**popupButton:** This should have the texture used for the folder list popup button in the Background of the Normal entry. None of the other entries need to have anything in them. In the supplied GUISkin, the texture has arrows on the right-hand side to indicate that it's a popup button, so the Padding values here prevent text from being potentially printed on top of the arrows. (Specifically, left 11, right 30, top 8, bottom 2.) The Border values make the button display properly. For the default skin, these are: left 5, right 14, top 4, bottom 4. The border and padding values may be different depending on the texture you use.

**popupBox:** This should have the texture used for the border/background of the popup folder list in the Background of the Normal entry. The border values of 4, 4, 4, 4 make the text be inset by 4 pixels.

# Public Inspector Variables 

These are the variables you see in the inspector when you have an object selected with the UniFileBrowser component attached. They can also be referenced in code by prefacing them with “UniFileBrowser.use.”. Note that all variables start with a lowercase letter, even though Unity displays them with uppercase in the inspector. For example,

```
    Debug.Log (UniFileBrowser.use.allowMultiSelect);
```

The variables are grouped into sections:

# File extensions

**Auto Add Extension:** if checked, this makes an extension automatically be added to any files that are saved. For example, if you have a routine for saving PNG files, you can specify that ".png" be added to any file that's saved.

**Added Extension:** this is the extension that will be added to saved files if Auto Add Extension is checked. It can be entered either with a dot (".jpg") or without ("jpg").

# File saving

**Default File Name:** you can use this to specify a default name that's automatically supplied when saving a file.

**Max File Name Length:** when saving a file, this will limit file names to the specified number of characters, not including the auto-added extension (if one is used). The max length must be at least 3, and not greater than 128.

**Prevent Illegal Char Input:** if checked, then users are unable to enter any of the illegal characters (as defined below) when typing file names into the "Save as" textfield. If not checked, then it would be a good idea to "scrub" file names in your own saving code.

**Illegal Chars:** a string that contains the characters that are not allowed to be input, if PreventIllegalCharInput is checked. By default this contains all characters that are typically not allowed in common file systems.

# General

**Use Delete Button:** if checked, this makes a "Delete" button appear to the left of the "Cancel" button. Unlike Open/Save, functionality for this button is built-in and doesn't need any file handling in your own scripts. **WARNING!** Obviously this can be quite destructive. You should almost certainly use "LimitToInitialFolder" (see below) if you enable this button.

**Limit To Initial Folder:** if checked, the user will not be able to navigate away from the initial folder/directory. Only files will be shown in the file browser window, not folders or volumes, and the folder popup menu will not be shown. When using this option, you probably want to specify the path using the SetPath function (see the "Using the filebrowser functions in code" section above). Typically you would use this if you have a particular gamesave folder that you want all saves to go into. Note that if you use OpenFolderWindow function, limitToInitialFolder is always false, since it wouldn't make sense to browse for folders and be limited to the initial folder at the same time.

# File list

**Show Volumes:** if checked, then a list of volumes (on OS X and Linux) or driver letters (on Windows) will appear at the top of the folder list, to allow navigation to different volumes. (Although this can always be done on OS X by browsing to "/Volumes" manually, but having a list is more convenient.) If LimitToInitialFolder is checked, then ShowVolumes will have no effect.

**Volumes Are Separate:** if checked, then the volumes will be shown in a separate non-scrolling box, rather than included in the list of folders and files. Has no effect if volumes aren't shown.

**Show Date:** if checked, then the file modification date is shown next to each file and folder.

**Date Width Add:** the number of pixels to add the the date field (if used), in case the automatic width computation is not accurate enough.

**Sort Type:** this sorts the file list by Name, DateNewest, or DateOldest. Name sorts alphabetically, DateNewest sorts by newest date on top, and DateOldest sorts by oldest date on top. Volumes and folders are always grouped together regardless of sort type, though folders will still be sorted as specified.

**Allow Multi Select:** if checked, then multiple files can be selected when using OpenFileWindow, using Shift to add a range of files and Command/Control (depending on OS) to add or subtract individual files. In this case a string array of paths is returned instead of a single string.

**Allow App Bundle Browsing:** on macOS, apps are bundled into folders with an ".app" extension. If this is checked, these app bundle folders can be browsed like any other folder. Otherwise they are invisible.

**Show Hidden Mac Files:** on macOS, files and folders are normally hidden if the name starts with ".". If this is checked, hidden files and folders are shown. Otherwise they are invisible.

# File filtering

**Filter Files:** if checked, this makes files be filtered by specified file extensions, such as .jpg, .txt, .unity, etc. So only the filtered files (and all folders) will be displayed.

**Filter File Extensions:** this is an array of file extensions to be used if file filtering is on. These can be entered either with a dot (".jpg") or without ("jpg"). This array can be left blank if you don't use the filter files option.

**Use Filter Button:** if checked, this makes a "Filter" button appear in the lower-left of the file browser window inside a small tab. This allows the user to toggle between files being filtered or not.

**Window Tab Offset:** only used if UseFilterButton is enabled. This is the number of pixels the window tab texture is offset from the default position in the x/y dimensions, so it can be moved around to account for different GUISkins.

**Window Tab:** the texture you want to use for the "filter file extensions" tab should be dragged onto this slot. This is optional. The default texture is UniFileBrowserAssets/Textures/UFBwindowtab. If you use your own texture called UFBwindowtab, it will be automatically placed in this slot (as detailed in HighlightTexture, above).

# Window settings

**Allow Window Resize:** if checked, then the user may resize the window by dragging the lower-right corner. Otherwise the window can't be resized.

**Allow Window Drag:** if checked, then the user may drag the window around the screen with the title bar. Otherwise the window can't be moved.

**File Window ID:** the window ID used by OnGUI for the main file browser window. Make sure this doesn't conflict with any of your own windows.

**Message Window ID:** the window ID used by OnGUI for the error/confirmation dialog window. Again, make sure this doesn't conflict with any windows you may be using.

**File Window Inset:** the number of pixels that the GUI elements will be inset from the edges of the window. Essentially a border.

**Default File Window Rect:** this is the default position and size of the file browser window when it's first opened.

**Min Window Width:** the file browser window is not allowed to be resized to anything smaller than this number of pixels wide.

**Min Window Height:** the file browser window is not allowed to be resized to anything smaller than this number of pixels high.

**Message Window Size:** this is the number of pixels wide and high the error/confirmation dialog window is. It always appears in the middle of the screen.

**Gui Depth:** use this to make the file browser window appear on top of or behind elements drawn by other OnGUI functions. The default of -1 makes it drawn on top of anything that sets a GUI.depth of 0 or higher.

# Window buttons

**Popup Rect:** this is the position and size of the popup folder list button, relative to the file window inset. So normally the x/y values would be 0.

**Button Size:** the Cancel, Delete, and Open/Save buttons are this number of pixels wide and high.

# Gui items

**Gui Skin:** the GUISkin you want to use for the file browser should be dragged onto this slot. This is required, because it must contain several custom styles, as detailed below. The default is UniFileBrowserAssets/UniFileBrowserGUISkin. If you use your own skin called UniFileBrowserGUISkin, it will be automatically placed in this slot when the UniFileBrowser script is first attached to a Game Object, or when you use the reset function (the location of the GUISkin doesn't matter, as long as it's in the project's Assets folder somewhere). So in that case, it won't be necessary to drag it into this slot.

**Highlight Texture:** the texture you want to use for files and folders that have been selected in the file browser. This is optional. The default texture is UniFileBrowserAssets/Textures/UFBhighlight. If you use your own texture called UFBhighlight, it will be automatically placed in this slot when the UniFileBrowser script is first attached to a Game Object, or when you use the reset function (the location of the texture doesn't matter, as long as it's in the project's Assets folder somewhere).

**Highlight Text Color:** the color that text is changed to for files and folders that have been selected in the file browser. This should show up nicely against whatever you use for the highlight texture.

**Message Window Texture:** the texture you want to use for the background of the message/error window. The regular Window style should have a background texture that depicts the resize widget (which is a 25x25 pixel area) in the lower-right corner, but message/error windows can't be resized, so the resize widget shouldn't be included in the background graphic for the message window texture. Otherwise it can be a duplicate of the regular window background texture. The default texture is UniFileBrowserAssets/Textures/UFBwindow2. If you use your own texture called UFBwindow2, it will be automatically placed in this slot.

**Drive icon:** the texture you want to use for the drive icon in the file list should be dragged onto this slot. This is optional, but recommended if you're using the Show Volumes option. The default texture is UniFileBrowserAssets/Textures/UFBdriveicon. If you use your own texture called UFBdriveicon, it will be automatically placed in this slot.

**Folder icon:** the texture you want to use for the folder icon in the file list should be dragged onto this slot. This is optional, though it's a good idea to have, or else folders and files are not easily distinguishable. The default texture is UniFileBrowserAssets/Textures/UFBfoldericon. If you use your own texture called UFBfoldericon, it will be automatically placed in this slot.

**File icon:** the texture you want to use for the file icon in the file list should be dragged onto this slot. This is optional. The default texture is UniFileBrowserAssets/Textures/UFBfileicon. If you use your own texture called UFBfileicon, it will be automatically placed in this slot.

There are 25 public functions that you can access through code. The functions should be prefaced with "UniFileBrowser.use." in order to access them. For example:

```
UniFileBrowser.use.OpenFileWindow (OpenFile);
```

# OpenFileWindow

```
OpenFileWindow (OpenFileFunction) : void
```

This causes the file browser window to open in Open File mode, where users can select files to be opened. The OpenFileFunction is a function in your script that handles the string returned from UniFileBrowser when the user selects a file. The function can be named whatever you desire (it doesn't have to be called "OpenFileFunction"), but it must have a string as a parameter and return void. If you're using the AllowMultiSelect option, then the function should accept a string array rather than a string. (Which is true even if only one file is selected...in this case the string array will have just one entry.) So, the OpenFileFunction should be defined like this, if not using AllowMultiSelect:

```
function OpenFile (path : String)    // JS
void OpenFile (string path)          // C#
```

Or like this if using AllowMultiSelect:

```
function OpenFile (paths : String[]) // JS
void OpenFile (string[] paths)       // C#
```

Note that if using AllowMultiSelect, the string array is not returned in any defined order, so if you want it to be sorted in some way you should write code to do this in your OpenFileFunction. Also, while it's possible for the user to select folders using multi-selection, these will be ignored and only actual files will be returned in the array.

The path string (or strings) contains the entire path, including the file name. If you want the file name alone, you can write code to separate it, such as by using Path.GetFileName (see the UniFileBrowser example script).

# SaveFileWindow

```
SaveFileWindow (SaveFileFunction) : void
```

This causes the file browser window to open in Save File mode. The SaveFileFunction is a function in your script that handles the string returned from UniFileBrowser when the user selects or creates a file name. The SaveFileFunction should be defined like the OpenFileFunction, although only with a string and not a string array, since Allow Multi Select only affects opening files and not saving them. Namely:

```
function SaveFile (path : String)    // JS
void SaveFile (string path)          // C#
```

The path string contains the entire path, including the file name. If you want the file name alone, you can write code to separate it (see the UniFileBrowser example script).

# OpenFolderWindow

```
OpenFolderWindow (showFiles : boolean, OpenFolderFunction) : void
```

This is similar to OpenFileWindow, except the purpose is to let the user select a directory/folder instead of a file. If "showFiles" is true, then both folders and files are shown when using this function (though only folders can be selected). If false, then only folders are shown. The OpenFolderFunction should be defined like OpenFileFunction or SaveFileFunction, using a string as a parameter:

```
function OpenFolder (path : String)  // JS
void OpenFolder (string path)        // C#
```

An example of usage:

```
UniFileBrowser.use.OpenFolder (true, OpenFolder);
```

# CloseFileWindow

```
CloseFileWindow () : void
```

Normally you shouldn't need to call this, since closing the window is typically initiated by the user, so this is handled automatically when appropriate after OpenFileWindow, SaveFileWindow, or OpenFolderWindow are called. But if you still need to do this for whatever reason, this function will, as expected, cause the file browser window to close immediately. The only exception is if an error or confirmation dialog window is currently open, since those are important and need to be handled properly.

# SetPath

```
SetPath (path : String) : void
```

By default, the path is initially set to the directory where your program is launched from. By passing the string "path", you can set it to be the path defined by this string instead. This should be called before using any of the Open or Save functions. If you use SetPath, make sure you use it in Start (or later) rather than Awake, since UniFileBrowser is not necessarily initialized when your Awake function runs. (Or you can use script execution order settings in Unity to make sure UniFileBrowser runs first.) For example:

```
UniFileBrowser.use.SetPath (Application.persistentDataPath);
```

# SetWindowTitle

```
SetWindowTitle (title : String) : void
```

In some cases you might want to use a different title for the file browser window other than the default "Open file"/"Save file"/"Select folder" titles. For example, say you have an app that allows saving of several different types of files, and you want the title to reflect which type is being saved. In this case, you can use the SetWindowTitle function to override the default title. Just call SetWindowTitle with a string that holds the desired title, after calling OpenFileWindow or SaveFileWindow. For example,

```
    UniFileBrowser.use.SaveFileWindow (SaveFile);
    UniFileBrowser.use.SetWindowTitle ("Save image");
```

It's important to call SetWindowTitle after the open or save function, since otherwise the default title will be used.

# SetFileWindowPosition

```
SetFileWindowPosition (position : Vector2) : void
```

If you want to set the file browser window's position through code, rather than relying on the "default file window rect" inspector variable, call this function with the desired x/y GUI coords. The file browser window will not allow itself to be placed off-screen, so it's possible that the actual position won't match the requested position, depending on the size of the screen. UniFileBrowser will attempt to use the closest possible position without reducing the window size below the minimum.

```
    UniFileBrowser.use.SetFileWindowPosition (new Vector2(425, 325));
```

# SetFileWindowSize

```
SetFileWindowSize (size : Vector2) : void
```

Similarly, you can set the width and height of the window with this function (where the x/y of the Vector2 equals width/height). Again, UniFileBrowser will constrain the window to the screen, so the actual size may be less than the requested size, if the new size is too big for the screen at the window's current position.

```
    UniFileBrowser.use.SetFileWindowSize (new Vector2(500, 450));
```

# FileWindowOpen

```
FileWindowOpen () : boolean
```

This returns a boolean that's true or false depending on whether the file browser window is open or not. For example:

```
     if (UniFileBrowser.use.FileWindowOpen()) {
          Debug.Log ("Browser window is open");
     }
```

# GetFileWindowRect

```
GetFileWindowRect () : Rect
```

This returns a Rect with the current GUI coordinates of the file browser window. For example:

```
     if (UniFileBrowser.use.GetFileWindowRect().Contains(Event.current.mousePosition)) {
          Debug.Log ("Mouse is inside file browser window");
     }
```

# SetFileExtensions

```
SetFileExtensions (extensions : String[]) : void
```

If you want to set different file filter extensions at runtime, use this function. It takes a String[] array, where each entry is a file extension. The file extensions can start with "." or not, so for example both "jpg" or ".jpg" are acceptable. If the string array doesn't contain anything, file filtering is turned off.

# SetAutoAddedExtension

```
SetAutoAddedExtension (extension : String) : void
```

If you're using AutoAddExtension, you can use this function to change the extension at runtime. The extension can start with "." or not.

```
     UniFileBrowser.use.SetAutoAddedExtension ("jpg");
```

# SendWindowCloseMessage

```
SendWindowCloseMessage (CloseWindowFunction) : void
```

If your app needs to know when the browser window is closed, you can create a function and pass it to SendWindowCloseMessage, and your function will be called as soon as the user closes the browser window. The function should have no arguments and should return void. For example:

```
// JS
function Start () {
     UniFileBrowser.use.SendWindowCloseMessage (FileWindowClosed);
}

function FileWindowClosed () : void {
     Debug.Log ("File browser window was closed");
}
```

```
// C#
     void Start () {
          UniFileBrowser.use.SendWindowCloseMessage (FileWindowClosed);
     }

     void FileWindowClosed (){
          Debug.Log ("File browser window was closed");
     }
```

# DontSendWindowCloseMessage

```
DontSendWindowCloseMessage () : void
```

If you've previously used SendWindowCloseMessage and don't want your CloseWindowFunction to be called anymore, then call DontSendWindowCloseMessage:

```
UniFileBrowser.use.DontSendWindowCloseMessage();
```

# SendWindowChangeMessage

```
SendWindowChangeMessage (WindowChangedFunction) : void
```

If you want to know when the file browser window is moved by the user, you can create a function and pass it to SendWindowChangeMessage, and your function will be called whenever the browser window is moved or resized. The function should have no arguments and should return void. For example:

```
// JS
function Start () {
     UniFileBrowser.use.SendWindowChangeMessage (FileWindowChanged);
}

function FileWindowChanged () : void {
     Debug.Log ("File browser window was moved");
}
```

```
// C#
     void Start () {
          UniFileBrowser.use.SendWindowChangeMessage (FileWindowChanged);
     }

     void FileWindowChanged () {
          Debug.Log ("File browser window was moved");
     }
```

# DontSendWindowChangeMessage

```
DontSendWindowChangeMessage () : void
```

If you've previously used SendWindowChangeMessage and don't want your WindowChangedFunction to be called anymore, then call DontSendWindowChangeMessage:

```
UniFileBrowser.use.DontSendWindowChangeMessage();
```

# SetCustomFunction

```
SetCustomFunction (CustomFunction) : void
```

This allows you to inject OnGUI code into the file window, which you can use for custom controls. The function should have no arguments and should return void. This example will make a button labeled "X" on the right-hand side of the file window near the top, which will print a message when clicked:

```
     // JS
     function Start () {
          UniFileBrowser.use.SetCustomFunction (MyCustomButton);
     }

     function MyCustomButton () : void {
          if (GUI.Button (Rect(UniFileBrowser.use.GetFileWindowRect().width-50, 50, 25,
25), "X")) {
               Debug.Log ("You clicked the X button!");
          }
     }
```

```
     // C#
          void Start () {
               UniFileBrowser.use.SetCustomFunction (MyCustomButton);
          }

          void MyCustomButton () {
               if (GUI.Button (new Rect(UniFileBrowser.use.GetFileWindowRect().width-50,
50, 25, 25), "X")) {
                    Debug.Log ("You clicked the X button!");
               }
          }
```

# RemoveCustomFunction

```
RemoveCustomFunction () : void
```

If you don't want the custom function to run anymore, calling this function will stop it:

```
     UniFileBrowser.use.RemoveCustomFunction();
```

# RefreshFileList

```
RefreshFileList () : void
```

You can use this to manually update the file list. You might want to do this if, for example, you have a custom function that changes what types of files are displayed.

```
     UniFileBrowser.use.RefreshFileList();
```

# UseFolderFilterFunction

```
UseFolderFilterFunction (CustomFunction) : void
```

This will allow you to supply a function that has custom logic for determining what folders are displayed. You could, for example, only show folders that contain .jpg and .png files. The function should accept a FileInfo array and return a boolean. The logic is applied to each folder in the list (if there are any). If the function returns true, then the folder is displayed, and if it returns false, then the folder will not be included in the list. For information on the FileInfo class, see http://msdn.microsoft.com/en-us/library/system.io.fileinfo.aspx. This example will only allow folders which have at least one file with a ".asset" extension inside:

```
     // JS
     import System.IO;

     function Start () {
          UniFileBrowser.use.UseFolderFilterFunction (MyFolderFilter);
     }

     function MyFolderFilter (files : FileInfo[]) : boolean {
          var containsAsset = false;
          for (var i = 0; i < files.Length; i++) {
               if (files[i].Extension == ".asset") {
                    return true;
               }
          }
          return false;
     }
```

# DontUseFilderFilterFunction

```
DontUseFolderFilterFunction () : void
```

If you've previously called UseFolderFilterFunction and don't want the custom filter to be applied anymore, then call this function:

```
     UniFileBrowser.use.DontUseFolderFilterFunction();
```

# UseFileFilterFunction

```
UseFileFilterFunction (CustomFunction) : void
```

Along the same lines as UseFolderFilterFunction, you can use this to filter all the files in the list by using custom logic, in those cases where the standard extension filtering isn't enough. The function should accept a string and return a boolean. The logic is applied to each file in the list (if there are any). If the function returns true, then the file is shown, otherwise it's not. This example will cause only files that start with "Foo" and end with ".bar" to be included, such as Foo1.bar, FooQQ.bar, etc.:

```
// JS
function Start () {
     UniFileBrowser.use.UseFileFilterFunction (MyFileFilter);
}

function MyFileFilter (filename : String) : boolean {
     if (filename.StartsWith ("Foo") && filename.EndsWith (".bar")) {
          return true;
     }
     return false;
}
```

```
// C#
     void Start () {
          UniFileBrowser.use.UseFileFilterFunction (MyFileFilter);
     }

     boolean MyFileFilter (string filename) {
          if (filename.StartsWith ("Foo") && filename.EndsWith (".bar")) {
               return true;
          }
          return false;
     }
```

# DontUseFileFilterFunction

```
DontUseFileFilterFunction () : void
```

If you've previously called UseFileFilterFunction and don't want the custom filter to be applied anymore, then call this function:

```
UniFileBrowser.use.DontUseFileFilterFunction();
```

# Scale

```
Scale (referenceWidth : float, referenceHeight : float) : void
```

This can be used to create a resolution-independent file-browser window, where it appears the same relative size on different resolution screens. For example, using 1920 and 1080 for the reference width and height respectively would cause the window to be the same size on both a standard HD screen and a 4K display. Note that Scale does not take the screen aspect ratio into consideration, so if you expect that Scale will be used with an arbitrary resolution, you should take care to supply appropriate values. For example, using the 1920 and 1080 examples above, the window would be distorted on a 16:10 2560 x 1600 screen. In that case you'd use 1728 and 1080 instead (Screen.width/Screen.height * 1080; note that the calculation should use floating-point division and not integer division).

```
    UniFileBrowser.use.Scale (1920, 1080);
```

# DontScale

```
DontScale () : void
```

If you've previously called Scale and don't want the window to be scaled anymore, then call this function:

```
    UniFileBrowser.use.DontScale();
```