FILE SYSTEM

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/filesystem

The **File System** plugin allows read and write access to files and folders on the local system.

For a demonstration of using file system features, see the Text Editor example.

Scripting

When using JavaScript or TypeScript coding, the features of this object can be accessed via the IFileSystemObjectType script interface.

Browser/platform support

For the File System plugin features to work, the browser must support the full File System Access API. (Note that *Origin Private File System* or OPFS is not used by the File System plugin.) As of November 2024 this is only supported in Chrome and Edge (and most other Chromium-based browsers) on desktop. Notably this means Firefox and Safari are not currently supported. However it is also supported in the Windows WebView2, macOS WKWebView and Linux CEF exporters.

> *The macOS WKWebView export option is based on the Safari browser engine, but Construct has extended it to support the File System plugin features even though the Safari browser does not support the File System Access API. Therefore you can rely on File System plugin features working in macOS WKWebView exports.*

Browser permissions model

Browsers impose security restrictions to ensure that general web browsing is safe. Therefore they start with **no access to local files or folders** at all. The only way to gain access to local files and folders in a browser is to show a **picker** - a "save file" dialog, an "open file" dialog, or a "pick folder" dialog.

Desktop exporters like Windows WebView2 **are an exception to this rule** and can have direct access to certain special folders like the user's documents folder without any dialogs or permission prompts - see below for more details. However in a browser, once the user has successfully completed a picker by making a selection and clicking OK, then permission to access the chosen files or folders is granted.

> *In browsers, pickers are also only allowed to be shown in a user input trigger, such as upon clicking a button, or starting a touch. This is another security measure to ensure pickers*

can't be shown at unexpected times, such as immediately upon loading the page.

In some circumstances browsers will also show a **permission prompt** to access a file. For example choosing a file with an "open file" picker allows reading the file without any further permission prompt, as the picker grants read access. However if later on an attempt is made to write to the opened file, browsers will normally show a permission prompt to verify that the user is willing to allow their previously chosen file to be modified. (This also only applies to browsers - desktop exporters won't show permission prompts.)

To help avoid the need to keep showing pickers, Construct is also able to **save the permission to access previously chosen files** and folders. The *Has picker tag* condition will be true if a picker completed in a prior session and the same chosen files or folders can still be accessed. Browsers may also show permission prompts when accessing files or folders from prior sessions.

Picker tags and file tags

Tags are short pieces of text used to identify different uses of the same feature. The File System plugin uses two kinds of tags. **Picker tags** identify different picker dialogs, such as two different uses of an "open file" dialog. Each successfully completed picker grants access to the chosen files or folders, which are then referred to by the picker tag.

Reading or writing a file then uses a **file tag**. This identifies the actual file system operation of reading or writing to a chosen file or folder. The file tag is optional as it's only needed if you want to track when file system operations complete or encounter an error - if you don't care about the result, you can leave the file tag empty.

The reason two kinds of tags are used is because one picker may grant access to multiple files or folders. For example one folder picker can grant access to a folder, in to which two files are then written. In this case there is one picker tag identifying the folder the user chose, and then two file tags identifying the separate write operations.

In summary picker tags are required in order to identify the files and folders the user chose, but file tags are optional and are used to identify when reading or writing specific files or folders completes or encounters an error.

Accessing known folders

In Windows WebView2, macOS WKWebView and Linux CEF exports, the browser permission model can be bypassed to directly access folders such as the user's documents folder. This is done by using pre-defined folder picker tags like "<documents>". These act like the folder was already picked on startup. You can check if these are available on startup by using the *Has picker tag* condition. If available then actions like *Write text file* can write to picker tag "<documents>" folder path "test.txt", in order to write "text.txt" in the user's Documents folder, without showing any picker dialog or permission prompts at all.

The full table of pre-defined picker tags for known folders is listed below. Note not all of these are guaranteed to be available: a system may not have some types of folders, in which case *Has picker tag* will be false even in a desktop export.

macOS App Sandbox

Note that in macOS WKWebView exports, if you enable the App Sandbox, then macOS creates a *container* for your app. This effectively gives your app its own folders like 'Documents', 'Pictures' and so on, inside a special app-specific path; the app won't actually access the folders the user would normally see in Finder. This is a privacy measure to make sure a sandboxed app can only access files saved by the same application, rather than being able to access all their files. When the sandbox is enabled, you can enable read-only or read-write permission for the Pictures, Movies and Downloads folder which provides access to the real folders shown in Finder. Alternatively if you disable the sandbox then the app will access all the known folders normally, with access to all the user's files.

Known folders table

Note the example paths shown in this table are for illustrative purposes only - the specific paths accessed may be different depending on the username, and may be in entirely different locations depending on the application install directory or system configuration.

Picker tag/platform	Description/path
<app>	Folder main application executable belongs to. **Note:** for security reasons sometimes the system makes this folder read-only (e.g. *Program Files* on Windows).
Windows	C:\Program Files\My app
macOS	/path/to/executable
Linux	/path/to/executable
<web-resource>	The folder that contains web resource files such as index.html, typically in a *www* subfolder to the app folder. **Note:** as with **<app>**, sometimes this will be a read-only folder as well.
Windows	C:\Program Files\My app\www
macOS	/path/to/executable/www
Linux	/path/to/executable/www
<current-app-data>	The local app data folder for this specific application.
Windows	C:\Users\username\AppData\Local\myapp
macOS	/Users/username/Library/Application Support/com.myapp.id
Linux	/home/username/.local/share/com.myapp.id
<local-app-data>	The main system local app data folder for the current user.
Windows	C:\Users\username\AppData\Local
macOS	/Users/username/Library/Application Support
Linux	/home/username/.local/share
<roaming-app-data>	The main system roaming app data folder for the current user.
Windows	C:\Users\username\AppData\Roaming
macOS	*Not supported*
Linux	*Not supported*
<desktop>	User's desktop folder.
Windows	C:\Users\username\Desktop
macOS	/Users/username/Desktop
Linux	/home/username/Desktop
<documents>	User's documents folder.
Windows	C:\Users\username\Documents
macOS	/Users/username/Documents
Linux	/home/username/Documents
<downloads>	User's downloads folder.
Windows	C:\Users\username\Downloads

Picker tag/platform	Description/path
macOS	/Users/username/Downloads
Linux	/home/username/Downloads
<pictures>	User's pictures/photos folder.
Windows	C:\Users\username\Pictures
macOS	/Users/username/Pictures
Linux	/home/username/Pictures
<profile>	Parent folder for all files and folders relating to the current user.
Windows	C:\Users\username
macOS	/Users/username
Linux	/home/username
<saved-games>	User's folder for games to save their current progress to.
Windows	C:\Users\username\Saved Games
macOS	*Not supported*
Linux	*Not supported*
<screenshots>	User's folder for games and applications to save screenshots to. Note this may be a subfolder to the pictures folder.
Windows	C:\Users\username\Pictures\Screenshots
macOS	*Not supported*
Linux	*Not supported*
<videos>	User's folder for games and applications to save recorded videos to.
Windows	C:\Users\username\Videos
macOS	/Users/username/Movies
Linux	/home/username/Videos
<dropped-files>	Used only to reference files dragged and dropped in to the window, when *On dropped files* triggers. This does not actually refer to a folder on disk, but allows accessing the dropped files as if the user picked a read-only folder containing them.

File system conditions

Desktop features supported

Detect whether desktop-specific features of this plugin are available. This is only the case when running after using a supported desktop export option. When true, the known folder pickers like "<documents>" can be used (providing they are available - use the *Has picker tag* condition to check), as well as the desktop-specific actions like *Run file* and *Shell open*.

Has picker tag

True if a given picker tag has been remembered from a previous session, or if a known folder tag is available. In this case the picker tag can still be referred to for file system operations, such as to read a previously chosen file, without having to show a picker again.

On picker complete

On picker error

Triggered after the *Show folder picker*, *Show open file picker* or *Show save file picker* actions with a matching picker tag, depending on the result of the picker. A picker is completed when the user successfully chooses a file or folder and access is then granted to the selection. Cancelling a picker, or otherwise being unable to show a picker (e.g. due to not being in a user input trigger), will trigger *On picker error*.

Is supported

True if file system features are supported. This depends on support for the File System Access API in the browser. If false then none of the features of the plugin will work.

On file operation complete

On file operation error

Triggered after any file or folder operation such as reading a file or creating a folder, with a matching file tag completes/fails.

On any file operation complete

On any file operation error

Triggered after any file or folder operation such as reading a file or creating a folder completes/fails, regardless of its file tag. The associated file tag can be retrieved with the *FileTag* expression.

On dropped files

Triggered when the user drags and drops files in to the window. The dropped files can be accessed using the special picker tag `<dropped-files>`. This allows accessing the dropped files as if the user picked a folder containing the files. The *FileCount* and *FileNameAt* expressions can also be immediately used to identify the number and names of the dropped files without having to use the *List content* action. The dropped files are read-only.

Has file/folder

Use this after the *List content* action to check whether a given file or folder exists in the returned list of files and folders. Note that this checks for the specified type of item - so for example if checking whether a folder exists, and the given name does exist but is actually a file, then the condition will be false. You must choose whether to check for the file name case-sensitively; note that typically on Windows and macOS file names are not case sensitive, but on Linux they are.

File system actions

Add accept type

Add a file type to be shown in the next open file or save file picker. By default the pickers will show all kinds of files; adding an accept type defaults to filtering by those kinds of files, and adding multiple accept types allows the user to switch between different filters. The accept type must specify a MIME type, such as "text/plain" for text files or "image/png" for PNG images. A list of file extensions can also be provided in case they cannot be deduced from the MIME type. These must begin with a dot and multiple file extensions can be specified separated by semicolons, e.g. ".png;.jpg" indicates to allow both .png and .jpg file extensions. A description can also be provided which may be shown in the picker. When the next picker

is shown, the list of added accept types is cleared and must be added again for another picker; typically you should use this action immediately prior to showing a picker.

Show folder picker

Show a folder picker allowing the user to choose a folder on their local system. The *Picker tag* identifies the chosen folder if it is completed successfully. The *Mode* determines what kind of permission prompt the user is shown.

> *Files can be written to a read-only folder, but the browser will show another permission prompt. Obtaining read & write permission initially will avoid later permission prompts.*

The *Picker ID* is an optional extra identifier for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

Show open file picker

Show an open file picker allowing the user to choose one or multiple files on their local system to open. The *Picker tag* identifies the chosen file or files if it is completed successfully, and grants read access. (The files can later be written to, but the browser will show another permission prompt.) *Show accept all* sets whether to show an accept type that shows all kinds of files; if disabled then the *Add accept type* action must be used beforehand. *Multiple* can be enabled to allow the user to select multiple files in the open file picker.

> *When opening multiple files, the selection is treated like an empty folder with just the chosen files in it. The FileCount and FileNameAt expressions will return the list of chosen files, and these names can be used in the Folder path parameter when reading or writing files.*

The *Picker ID* is an optional extra identifier for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

Show save file picker

Show a save file picker allowing the user to choose a file on their local system to save to. The *Picker tag* identifies the chosen file if it is completed successfully, and grants write access.

> *Note that the chosen file is erased, so cannot be read from. It is expected that the file contents will only be written to.*

Show accept all sets whether to show an accept type that shows all kinds of files; if disabled then the *Add accept type* action must be used beforehand. The *Suggested name* is used as the initial filename choice to save to in the picker. The *Picker ID* is an optional extra identifier

for remembering the picker settings, such as the last viewed folder. The *Start in* setting allows choosing a default system folder to show as the initial selection of the folder picker, but is overridden by any remembered settings from the same *Picker ID*.

Read text file

Read binary file

Read the contents of a file from a previously completed picker. The file or folder to read from is identified by the *Picker tag*. The *Folder path* is optional and is only used for folder pickers or from a open picker with *Multiple* enabled, and specifies the filename to be read, e.g. "file1.txt" or "subfolder/file2.txt" when using a folder picker. When using a save file picker or an open file picker for a single file, leave the *Folder path* empty as it is not used, as it will read the single file chosen by the picker. The *File tag* is optional and allows identifying when the read operation completes or fails with the *On file operation complete/error* triggers. In the case of reading a text file, the *FileText* expression is set to the read text content when completed. In the case of reading a binary file, the read file contents will be placed in the chosen Binary Data object when completed.

Write text file

Write binary file

Write text or binary data to a file from a previously completed picker. The file or folder to write to is identified by the *Picker tag*. The *Folder path* is optional and is only used for folder pickers or from a open picker with *Multiple* enabled, and specifies the filename to be written to, e.g. "file1.txt" or "subfolder/file2.txt" when using a folder picker. When using a save file picker or an open file picker for a single file, leave the *Folder path* empty as it is not used, as it will write to the single file chosen by the picker. The *File tag* is optional and allows identifying when the write operation completes or fails with the *On file operation complete/error* triggers. In the case of writing a text file, the given *Text* will be written to the file, and the text content can optionally be appended to the end of the existing file.

> *Note that save file pickers erase the chosen file, so appending is only useful after using an open file picker.*

In the case of writing a binary file, the contents of the chosen Binary Data will be written to the file.

Copy file

Only applies to folder pickers. Copy a file within a previously picked folder. The *Source path* specifies an existing file to copy, e.g. "subfolder/file1.txt". The *Destination path* specifies where to create a copy; this will either create a new file if it doesn't exist, or overwrite an existing file if it already exists. The *File tag* is optional and allows identifying when the copy operation completes or fails with the *On file operation complete/error* triggers.

Create folder

Only applies to folder pickers. Create a subfolder within a previously picked folder. The *Folder path* specifies the folder to create. This can refer to multiple subfolders which will all be created, e.g. "subfolder/otherfolder". The *File tag* is optional and allows identifying when the folder creation operation completes or fails with the *On file operation complete/error* triggers.

Delete

Only applies to folder pickers. Delete a file or folder within a previously picked folder. The *Folder path* specifies the file or folder to delete, e.g. "subfolder/file1.txt" or "subfolder". The *Recursive* option applies only when *Folder path* identifies a folder: if enabled it will delete all files and folders within the identified folder as well as the folder itself, but if disabled it will only successfully delete the folder if it is already empty. The *File tag* is optional and allows identifying when the delete operation completes or fails with the *On file operation complete/error* triggers.

List content

Only applies to folder pickers. Retrieves a list of all files and folders within a previously picked folder. The *Folder path* identifies a subfolder to list the contents for, or can be left empty to list the contents of the originally picked folder. Enabling *Recursive* will also list all files and folders through subfolders of the specified folder.

> *Recursive listing may be slow with a very large folder that contains thousands of files/subfolders.*

The *File tag* is optional and allows identifying when the listing operation completes or fails with the *On file operation complete/error* triggers. Once completed, the available files and folders can be accessed with the *FileCount*, *FileNameAt*, *FolderCount* and *FolderNameAt* expressions. When *Recursive* was enabled, the listed file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.

Move file

Only applies to folder pickers. Move a file within a previously picked folder. The file can be moved from one subfolder to another, or if it stays in the same folder, it renames the file. The *Source path* specifies the file to move, e.g. "subfolder/file1.txt". The *Destination path* specifies where to move (or rename) the file to. The *File tag* is optional and allows identifying when the copy operation completes or fails with the *On file operation complete/error* triggers.

Run file

Only supported in desktop exports when the *Desktop features available* condition is true. This action attempts to execute the provided file path. Command-line arguments can also be optionally provided. This can be used to run another program. Uniquely for this action the

folder picker parameter can be left empty, in which case the provided file path is run directly; this can be useful to run system executables like "cmd.exe" on Windows.

Shell open

Only supported in desktop exports when the *Desktop features available* condition is true. This action requests that the operating system shell (the component that handles the system user interface) opens the specified file. Typically this launches the default app associated with the file type and then opens the file in it, similar to double-clicking the file in the system file explorer.

File system expressions

FileCount

FileNameAt(index)

Retrieve the number and filenames of available files. The available list of files is updated in the following situations:

- After an open file picker, to identify the available filenames (which can be multiple filenames if the *Multiple* option was enabled, otherwise it is just one file)
- After the *List content* action for listing the contents of a folder. If the *Recursive* option was enabled, file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.
- After *On dropped files* triggers, to identify the available dropped files
- After a save file picker to identify the chosen saved filename (which is just one file)

FolderCount

FolderNameAt(index)

Retrieve the number and names of available folders. The available list of folders is updated in the following situations:

- After the *List content* action for listing the contents of a folder. If the *List content* action *Recursive* option was enabled, file and folder names may include slashes, such as *subfolder/file.txt*; for cross-platform consistency these always use forwards slashes, even on Windows.
- After a folder picker has completed, to identify the name of the chosen folder (which is just one folder).

FileText

After a *Read text file* action completes successfully, the contents of the read file.

> *Note that this is replaced when the next Read text file action completes, so it is best to use it immediately after reading a file.*

FileTag

In a file system operation trigger such as *On any file operation complete*, this returns the file tag of the associated file operation.

FolderPath

In a file system operation trigger working within a folder, this returns the folder path of the associated file operation.

PickerTag

In a trigger, this returns the picker tag associated with the picker or file system operation.