File system

This extension gives you access to the file system of the pc the game is running on.

This works only on Windows, Linux and MacOS. Mobile games and web games are currently not supported.

File system versus Storage extension

GDevelop offers two different ways to save data permanently. The storage extension and the file system extension.

Since GDevelop is built upon web technology, at the time of writing, every game is basically a website that either runs directly in a browser (Web export) or inside a stripped-down browser container “Electron” (Windows, macOS, Linux) / “Apache Cordova” (Android, iOS).

All these platforms offer web storage, integrated into the browser component of their runtimes where data can be saved in a dictionary-like key/value style.

For security reasons, web browsers don't let websites write any data to the file system of the computer they are executed on. Only games exported for Windows, macOS, or Linux can use this functionality.

Storage

- It is available on all platforms, a game can be exported to.
- It can only hold text and numbers, binary data like screenshots can't be saved to the web storage.
- Its maximum capacity is limited to a few megabytes, depending on the type of web browser the player uses.
- The player has no direct access to web storage.

File system

- It is only available on Windows, Linux and macOS.
- You can save any kind of data.
- You are free to choose a folder to save your data.
- There are no limits on the file size.
- The player can edit the files (i.e. manually correct a wrong screen resolution in a config file with a text editor)

Asynchronous execution

By default, actions get executed synchronously in GDevelop. This means that the game loop waits for each action to finish its execution before it runs the following event. Actions that take longer than a few milliseconds to execute will freeze the whole execution of the game until they finish.

To avoid these freezes, GDevelop supports asynchronous execution of specific actions. If you choose the “async” version of the action, the operation will be delegated to a background task that runs in parallel to the game loop. After executing an asynchronous action, GDevelop will immediately execute the next event in the event sheet without waiting for the action to finish.

To get notified about the status of the background task, the asynchronous actions have an optional result variable. This variable will be updated with the result of the operation, at the moment the background task has finished.

By checking its value, you get to know when precisely the operation has finished and with which result:

- **“ok”**: The operation was performed without errors.
- **“error”**: Something went wrong while trying to perform the action.

Saving the name of the next level after completing the previous one in a linear puzzle game, need not be done asynchronously.

On the other hand, when trying to add an autosave function to an open-world game with procedurally generated unlimited terrain, the game developer will probably not get around saving the game asynchronously due to the heap of data to be written into the file.

It is up to the game developer to decide which variant of the action is suitable for the individual situation.

Conditions

File or directory exists

This condition checks if the given file or directory exists on the file system.

Parameters

Path to file or directory: The absolute path to the file or directory which shall be checked.

Actions

Create a directory

This action creates a new folder at the given file path.

Parameters

Directory: The absolute file path to the directory which should be created. It is advised to use the expressions for special folders (see below), to keep you game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Save a text into a file

This action saves the given text (string) into a file on the file system synchronously.

Parameters

String (text): The text that will be saved to the file.

Save path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Save a text into a file (async)

This action saves the given text (string) into a file on the file system asynchronously.

Parameters

String (text): The text that will be saved to the file.

Save path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred. The variable will be updated, at the moment the file operation has finished.

Save a scene variable into a JSON file

This action saves the given scene variable into a file on the file system in JSON format synchronously.

Parameters

Scene variable: The scene variable to save into the file. This can be a number-, text variable or a structure with children.

Save path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Save a scene variable into a JSON file (async)

This action saves the given scene variable into a file on the file system in JSON format asynchronously.

Parameters

Scene variable: The scene variable to save into the file. This can be a number-, text variable or a structure with children.

Save path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred. The variable will be updated, at the moment the file operation has finished.

Load a text from a JSON file

This action loads the JSON formatted text from a file and converts it into a scene variable structure. synchronously.

Parameters

Scene variable: The name of the scene variable to which the loaded structure will be added.

Load path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Load a text from a JSON file (async)

This action loads the JSON formatted text from a file and converts it into a scene variable structure. asynchronously.

Parameters

Scene variable: The name of the scene variable to which the loaded structure will be added.

Load path: The path on the file system where the file should be saved. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred. The variable will be updated, at the moment the file operation has finished.

Load a text from a file

This action loads the text from a file into the given scene variable. synchronously.

Parameters

Scene variable: The scene variable to store the text.

Load path: The path on the file system where the file is located. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Load a text from a file (async)

This action loads the text from a file into the given scene variable. asynchronously.

Parameters

Scene variable: The scene variable to store the text.

Load path: The path on the file system where the file is located. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred. The variable will be updated, at the moment the file operation has finished.

Delete a file

This action deletes the file at the given file path synchronously.

Parameters

File path: The path on the file system where the file is located. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred.

Delete a file

This action deletes the file at the given file path asynchronously.

Parameters

File path: The path on the file system where the file is located. It is advised to use the expressions for special folders (see below) to keep your game platform independent.

(Optional) Result variable: Variable to store the result. It can either hold the value 'ok': the task was successful or 'error': an error occurred. The variable will be updated, at the moment the file operation has finished.

Expressions

These expressions return the path to special folders on the users' operating system. If you use these expressions for loading and saving files it will be guaranteed to work on all supported operating systems. (Currently Windows, Linux, and macOS)

The following gives detailed descriptions of what each of the Filesystem expressions are for. To see the syntax of the expressions refer to the filesystem section of this article. It is *strongly recommended* that you do not manually type any expressions, and instead use the expression builder as mentioned here.

Desktop folder

This expression returns the operating system independent path to the *Desktop* folder of the user that runs your game.

Documents folder

This expression returns the operating system independent path to the *Documents* folder of the user that runs your game. This is the standard folder for storing documents.

This games executable folder

This expression returns the operating system independent path to the folder where your game is being executed from.

Pictures folder

This expression returns the operating system independent path to the *Pictures* folder of the user that runs your game. This is the standard folder for storing images.

Temp folder

This expression returns the operating system independent path to the *Temp* folder of the user that runs your game. This folder is used for temporary files that your operating system can delete at any time.

Userdata folder

This expression returns the operating system independent path to the *UserData* folder of the user that runs your game. This folder is used for storing application settings.

Path delimiter

This expression returns the operating system independent path delimiter character. (“\” on Windows and “/” on Linux and macOS). Use this expression to build cross-platform file paths that can be accessed on all supported operating systems.

Example

In order to save a screenshot to the *Pictures* directory you could write:

```
FileSystem::PicturesPath() + FileSystem::PathDelimiter() + "my_screenshot.png"
```

This will work on Windows, Linux, and macOS.