# Storage Tools For Android

v1.0.0

## Introduction

Storage Tools For Android is a Unity asset for the Android platform that provides access to native storage and utility functions. It uses Android’s Storage Access Framework (SAF), which grants read and write access to files and folders that are selected by the end user. Selecting a folder also grants access to its descendants.
Raw paths inside your app’s private data storage can be accessed without permission.

The included storage functions are meant to work with raw paths and the SAF paths returned by the file and folder pickers in this asset, and are not guaranteed to work with SAF paths obtained from other sources. Raw paths must start with a slash “/”, and must not include “file://”.

Supports Android API level 26 and above, and Unity 2021 and above.

## Getting Started

To add the asset to your project, go to Window > Package Manager in Unity, find “Storage Tools For Android” in “My Assets”, then click the Download button. When the asset has been downloaded, click the Import button. A new folder called “StorageToolsForAndroid” will be added to your asset tree.

This is also a good time to add “Android Logcat” from “Unity Registry”. This allows you to monitor the Android device logs directly in Unity when the app is running on the device.
This is important because the asset only operates on the Android platform, and will not work in the Unity editor.

## Demo Scene and Scripting Examples

A basic demo scene is included, which allows you to pick a file or a folder, and it displays some basic info about the selected item. More detailed info is sent to the device log.

You can find the demo scene in the asset tree at:
Assets > AndroidStorageTools > Scenes > StorageDemo.unity

An example C# script is included, which includes example code for all included functions.

You can find the example script at:
Assets > AndroidStorageTools > Scripts > StorageExamples.cs

## Setup

### Android Platform Setup

In Unity, open File > Build Profiles, select “Android” from platforms, and click “Switch Platform”.
Open the project settings window. Edit > Project Settings…
Select “Player” in the left pane, and go to the “Other Settings” section.
Set the minimum API lever to 26 or higher.
Set “Write Permission” to “External(SDCard)”. Runtime permission is not required for SAF, because the user will be granting permission automatically by selecting files and folders. But this setting might cause a permission prompt at startup. You can prevent it by adding this line to your AndroidManifest.xml, right before the activity tag.

```
<meta-data android:name="unityplayer.SkipPermissionsDialog" android:value="true" />
```

No permission is needed for raw paths in your app files folder.

### Scripting Setup

In your C# script, add the using statement at the beginning of the file.

```
using AndroidStorageTools;
```

Next, in your start method, initialize the native plugin’s callback relay, and register the error callback. A GameObject called “AndroidStorageRelay” will be created in your scene at runtime using DontDestroyOnLoad.

```
void Start()
{
    if (!StorageRelay.isInitialized)  StorageRelay.InitializeRelay();
    StorageTools.RegisterStorageErrorCallback(OnError);
}
```

Now create a method to receive error messages.

```
public void OnError(string error)
{
    Debug.LogError("error: " + error);
}
```

**Setup is now complete, and you can start using the plugin.**

## Basic Usage

### Pick a file

public static void OpenFilePicker(string mimeTypes, Action<string, string> callback)

To pick a file, call the “OpenFilePicker” method in the StorageTools namespace, and specify a method to receive the results of the file picker. You also need to specify which mime types will be selectable in the file picker. Refer to StorageExamples.cs to see more mime type examples.

```
public void OpenFilePicker()
{
    string mimeTypes = "text/*,audio/*,video/*,image/*";
    StorageTools.OpenFilePicker(mimeTypes, OnFilePicked);
}

void OnFilePicked(string filePath, string fileName)
{
    Debug.Log("Path of selected file: " + filePath);
    Debug.Log("Name of selected file: " + fileName);
}
```

### Pick a folder

public static void OpenFolderPicker(Action<string> callback)

To pick a folder, call the “OpenFolderPicker” method in the StorageTools namespace, and specify a method to receive the results of the folder picker.

```
public void OpenFolderPicker()
{
    StorageTools.OpenFolderPicker(OnFolderPicked);
}

void OnFolderPicked(string folderPath)
{
    Debug.Log("Path of selected folder: " + folderPath);
}
```

**Pick multiple files**

public static void OpenMultiFilePicker(string mimeTypes, Action<string[], string[]> callback)

The multi file picker works the same way as the file picker. Specify your callback method, and specify the mime types you want to be selectable. The results will be returned as string arrays of paths and names, which you can then process in the callback method or pass to another method.

```
public void OpenMultiFilePicker()
{
    string mediaMimeTypes = "text/*,audio/*,video/*,image/*";
    StorageTools.OpenMultiFilePicker(mediaMimeTypes, OnMultiFilePicked);
}

void OnMultiFilePicked(string[] filePaths, string[] fileNames)
{
    if (filePaths != null && fileNames != null && filePaths.Length == fileNames.Length)
    {
        Debug.Log(filePaths.Length + " file(s) picked");

        for (int i = 0; i < filePaths.Length; i++)
        {
            Debug.Log("File name (" + i + "): " + fileNames[i]);
            Debug.Log("File path (" + i + "): " + filePaths[i]);
        }
    }
}
```

## Other Functions

Refer to StorageExamples.cs for usage examples for each of these functions.

## File and Folder Operations

**CreateFile**

public static void CreateFile(string pathOrUri, string fileName, Action<string> callback)

**DeleteFile**

public static void DeleteFile(string pathOrUri)

**CreateFolder**

public static void CreateFolder(string pathOrUri, string folderName, Action<string> callback)

**DeleteFolder**
```
public static void DeleteFolder(string pathOrUri)
```

**CopyFile**
```
public static void CopyFile(string sourceFilePath, string destinationFolderPath, Action<string> callback)
```

**CopyFolder**
```
public static void CopyFolder(string sourcePath, string destinationPath, Action<string> callback)
```

**MoveFile**
```
public static void MoveFile(string sourcePath, string destinationPath, Action<string> callback)
```

**MoveFolder**
```
public static void MoveFolder(string sourcePath, string destinationPath, Action<string> callback)
```

**RenameFile**
```
public static void RenameFile(string filePath, string newName, Action<string> callback)
```

**RenameFolder**
```
public static void RenameFolder(string folderPath, string newName, Action<string> callback)
```

**WriteBytesToFile**
```
public static void WriteBytesToFile(string pathOrUri, byte[] data)
```

**ReadBytesFromFile**
```
public static byte[] ReadBytesFromFile(string pathOrUri)
```

## File and Folder Information

**CheckExists**
```
public static bool CheckExists(string path)
```

**IsFolder**
```
public static bool IsFolder(string pathOrUri)
```

**GetFolderContents**
```
public static string[] GetFolderContents(string folderPath)
```

**GetFileOrFolderName**
```
public static string GetFileOrFolderName(string pathOrUri)
```

**GetParentFolder**

`public static string GetParentFolder(string path)`

If this is used on a picked file or folder that is not a descendant of a picked folder, no permissions will exist for the parent folder, and most functions will not work on it, but the path of the parent will still be returned.

**GetFileSizeInBytes**

`public static string GetFileSizeInBytes(string filePath)`

**GetCreationTime**

`public static string GetCreationTime(string path)`

Returns the creation time of a file or folder in the device time zone as "yyyy-MM-dd HH:mm:ss". Creation time is not available in SAF, so this will only work for raw app data paths.

**GetLastModifiedTime**

`public static string GetLastModifiedTime(string path)`

Returns the last modified time of a file or folder in the device time zone as "yyyy-MM-dd HH:mm:ss".

## Storage Information

**GetTotalStorageInBytes**

`public static string GetTotalStorageInBytes()`

**GetUsedStorageInBytes**

`public static string GetUsedStorageInBytes()`

**GetAvailableStorageInBytes**

`public static string GetAvailableStorageInBytes()`

**GetTotalRamInBytes**

`public static string GetTotalRamInBytes()`

**GetUsedRamInBytes**

`public static string GetUsedRamInBytes()`

**GetAvailableRamInBytes**

`public static string GetAvailableRamInBytes()`

**GetLowRamThresholdInBytes**

`public static string GetLowRamThresholdInBytes()`

**IsLowRamState**

`public static bool IsLowRamState()`

## System information

**GetCurrentDateTime**

```
public static string GetCurrentDateTime()
```

Returns the current date and time in the device time zone as "yyyy-MM-dd HH:mm:ss".

**GetOsVersionCode**

```
public static int GetOsVersionCode()
```

**GetLocale**

```
public static string GetLocale()
```

Returns as "language_COUNTRY" (ex. “en_US”).

**GetDeviceModel**

```
public static string GetDeviceModel()
```

Returns as "manufacturer model" (ex. “Samsung Galaxy S21”).

**GetScreenResolution**

```
public static string GetScreenResolution()
```

Returns as "WIDTHxHEIGHT" (ex. “1920x1080”).

**IsTablet**

```
public static bool IsTablet()
```

**IsDeveloperModeEnabled**

```
public static bool IsDeveloperModeEnabled()
```

**GetBatteryLevelPercent**

```
public static int GetBatteryLevelPercent()
```

**IsBatteryCharging**

```
public static bool IsBatteryCharging()
```

**GetBatteryTemperatureInCelsius**

```
public static int GetBatteryTemperatureInCelsius()
```

**GetBatteryHealth**

```
public static string GetBatteryHealth()
```

Possible values: GOOD, OVERHEAT, DEAD, OVER_VOLTAGE, FAILED, COLD, UNKNOWN

## Misc Utilities

**GetTextFromClipboard**

`public static string GetTextFromClipboard()`

**CopyTextToClipboard**

`public static void CopyTextToClipboard(string text)`

**SendToastNotification**

`public static void SendToastNotification(string message, bool longDuration)`

**OpenAppSettings**

`public static void OpenAppSettings()`

Opens the OS settings page for the current app.

**OpenStoreListing**

`public static void OpenStoreListing()`

Opens the Play Store listing for the current app.

**OpenOtherAppStoreListing**

`public static void OpenOtherAppStoreListing(string appID)`

Opens the Play Store listing for a different app, specified with appID (ex. com.company.app)

## Notes

CopyFile or other file related functions may produce an error in the log when the destination is a SAF path, even if the functions do actually work. The error may look like this.

```
2025/02/20 03:17:00.598 11881 11901 Warn DocumentsContract Failed to copy
document
2025/02/20 03:17:00.598 11881 11901 Warn DocumentsContract
java.lang.UnsupportedOperationException: Copy not supported
```

The error does not come through the native relay, but comes from the SAF DocumentsContract function itself.
This happens because the plugin first tries to use DocumentsContract.copyDocument or similar functions, and if it fails, it falls back to a different method of copying.
These errors can safely be ignored if the functions are working.

Each function has been tested extensively, including combinations of functions, but there are many possible combinations. If you discover any issues with any functions or combinations of functions, please report the issue to the developer by emailing blevok.apps@gmail.com