CSPro Android File Access Library

Starting with Android 11(R) (API level 30) apps only have access to their own internal and external files (and cache) directories and special shared directories. Apps, other than OS default file manager, are restricted from reading or writing files to internal/external files directories of other apps. CSEntry file access library provides an interface to read and write files to and from the "csentry" directory by third party apps.

The library is implemented as an Android Archive Library (AAR) and may be directly consumed by the client apps developed in Android Studio. **The library requires CSEntry 7.7 or higher to be installed on the same device as the client app**.

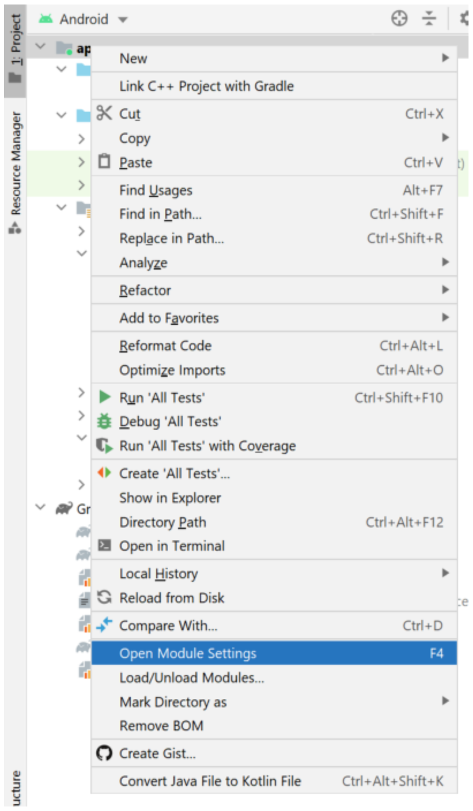
Linking file access library to your Android Studio project

Prerequisites:

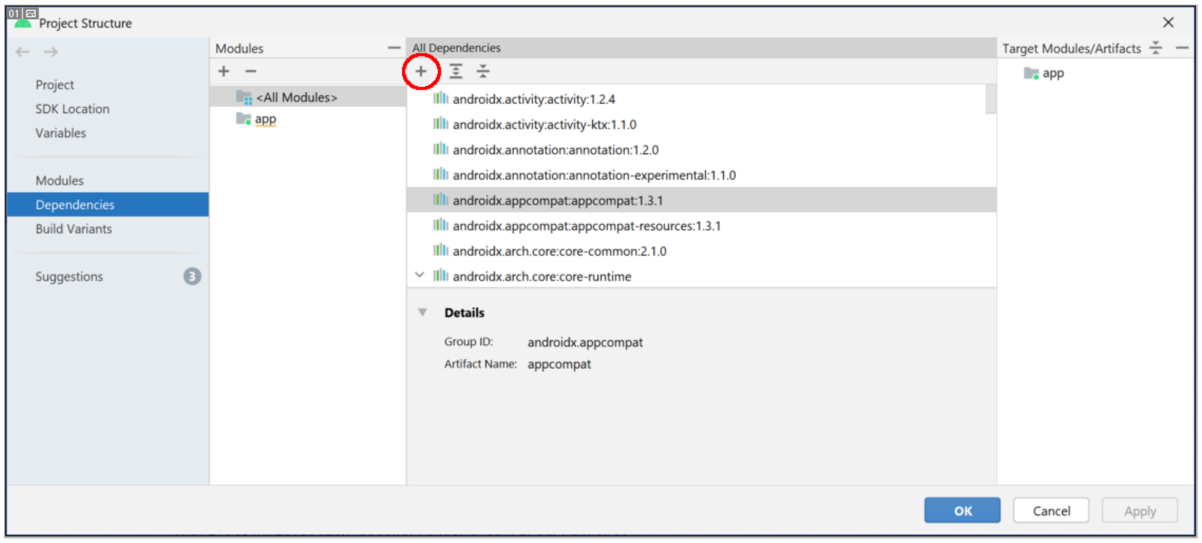
- Target SDK version of the client app must be 30 or higher
- Minimum SDK version of the client app must be 16 or higher
- Client app needs to provide an AppCompatActivity context to the FileAccessHelper interface instance

The following guide shows how to link the library in Android Studio 4.2.2

1. Copy *fileaccess.aar* file to your *app/libs/* directory
2. In Project view Right-click on your app module and select *Open Module Settings*



3. In the Project Structure dialog under All Dependencies click on the (+) button and select *JAR/AAR Dependency* option from the drop-down menu

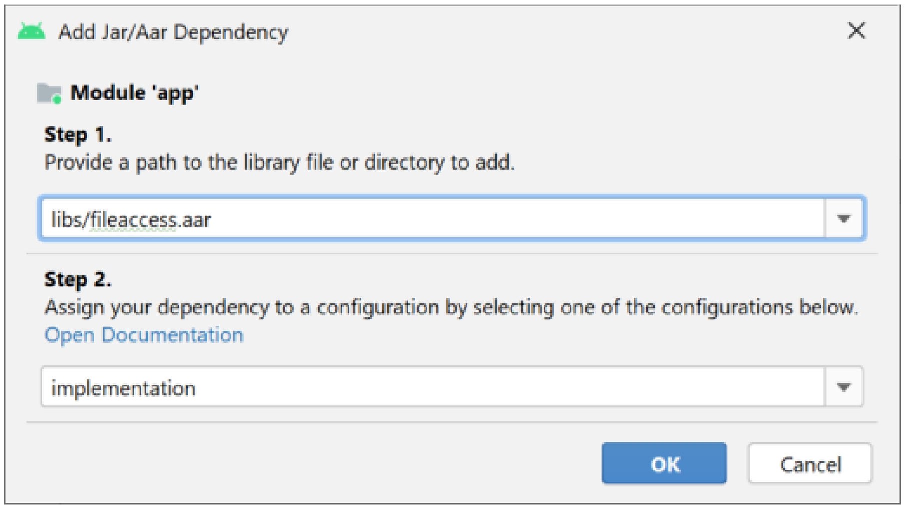


androidx.activity:activity:1.2.4
androidx.activity:activity-ktx:1.1.0
androidx.annotation:annotation:1.2.0
androidx.annotation:annotation-experimental:1.1.0
androidx.appcompat:appcompat:1.3.1
androidx.appcompat:appcompat-resources:1.3.1
androidx.arch.core:core-common:2.1.0
androidx.arch.core:core-runtime

Details

Group ID: androidx.appcompat
Artifact Name: appcompat

4. In *Step 1.* textbox write *libs/fileaccess.aar*. In *Step 2*. select *implementation* and click OK



Module 'app'

Step 1.
Provide a path to the library file or directory to add.

libs/fileaccess.aar

Step 2.
Assign your dependency to a configuration by selecting one of the configurations below.

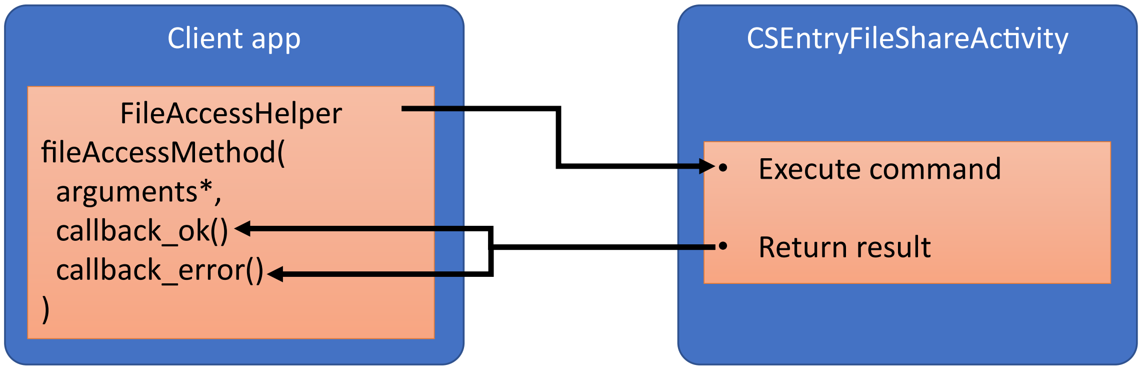
implementation

After performing these steps you should see the line added to dependencies section of your module's build.gradle file that says:

```
implementation files('libs/fileaccess.aar')
```

Using FileAccessHelper class instance to manipulate files in csentry/ directory

The following diagram describes interaction pipeline between the client app and CSEntry app:



All of the file manipulation commands sent to CSEntryFileShare activity are abstracted by the FileAccessHelper class and exposed as public methods to the client. To use the FileAccessHelper class add the following import statement to any of your source files that need access to it:

```
import gov.census.cspro.csentry.fileaccess.FileAccessHelper
```

In the most basic scenario FileAccessHelper is instantiated as a singleton object at application start and reused throughout the client app lifecycle:

```
val fileAccessHelper = FileAccessHelper(this)
```

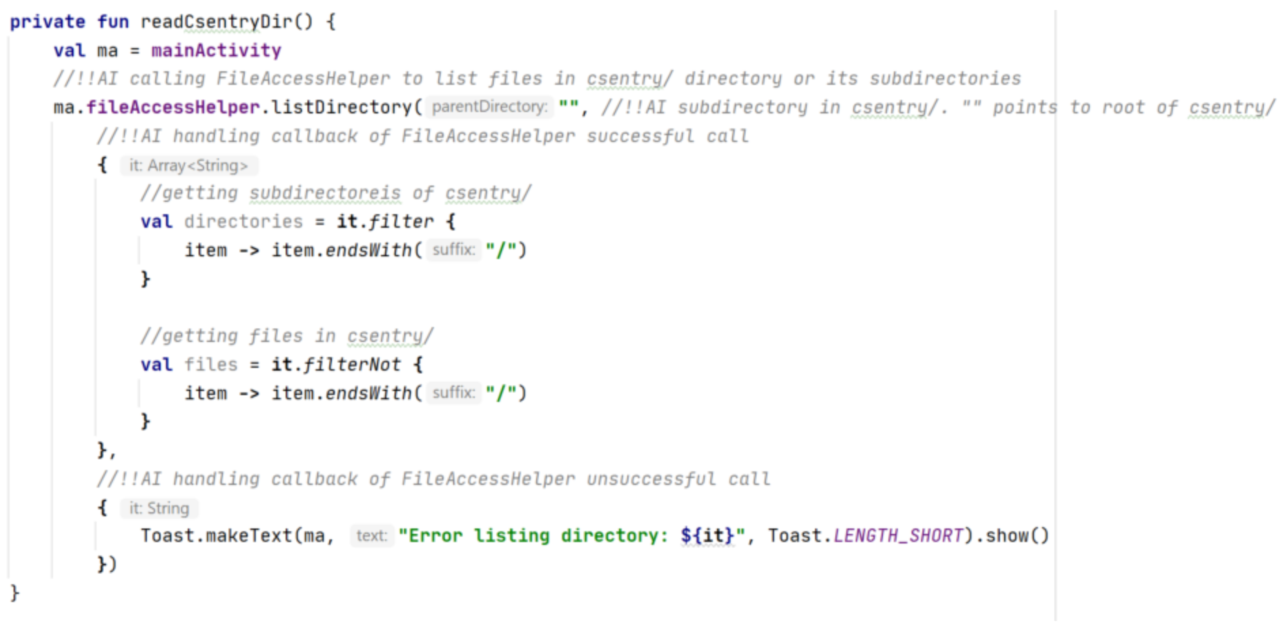
The constructor expects a single argument of type AppCompatActivity. This activity is used as a context for building intent objects for launching CSEntryFileShareActivity and processing its result.

Once the file access helper object is created, it can be used to send commands to CSEntry through several public methods declared in FileAccessHelper. All methods follow the same basic pattern:

fileAccessMethod(arguments*, callback_ok(), callback_error());

- fileAccessMethod is one of command methods. Each method is documented if FileAccessHelper methods chapter of this article
- arguments* - input arguments needed for command execution
- callback_ok() – optional function that handles command result in case the call was successful
- callback_error() – optional function that handles command result in case the call was unsuccessful

callback_ok() function provides argument of type Array<String> which contains result arguments returned by CSEntryFileShareActivity. The following example demonstrates a call to list all directories and files in csentry/ directory:



```
private fun readCsentryDir() {
    val ma = mainActivity
    //!!AI calling FileAccessHelper to list files in csentry/ directory or its subdirectories
    ma.fileAccessHelper.listDirectory( parentDirectory: "", //!!AI subdirectory in csentry/. "" points to root of csentry/
        //!!AI handling callback of FileAccessHelper successful call
        {  it: Array<String>
            //getting subdirectoreis of csentry/
            val directories = it.filter {
                item -> item.endsWith( suffix: "/")
            }

            //getting files in csentry/
            val files = it.filterNot {
                item -> item.endsWith( suffix: "/")
            }
        },
        //!!AI handling callback of FileAccessHelper unsuccessful call
        {  it: String
            Toast.makeText(ma,  text: "Error listing directory: ${it}", Toast.LENGTH_SHORT).show()
        })
}
```

In this example the command is to list all subdirectories and files in the csentry/ directory. The method that generates this command is *.listDirectory*. The first argument expects a path to a directory within csentry/ directory. Empty string represents the root of csentry/.

Note that csentry/ directory is the root of the file access. This interface does not allow accessing any files outside of csentry/.

Second argument is an optional function that is executed upon successful completion of command execution. Its own argument is an array of type String that contains the list of returned files and directories. This example shows how to handle this callback and get a list of directories and files from the array argument

Third argument is an optional function that is executed upon unsuccessful completion of command execution. Its own argument is a string with the error message reported by CSENtryFileShareActivity.

FileAccessHelper methods

Method	Arguments	Description
`listDirectory`		Lists all files and subdirectories of a given directory under csentry/ Lists only top level files/directories
	`parentDirectory: String`	A directory under csentry/ for which the listing is to be made. This argument is optional. If not set or set to empty string, lists files/directories in the root of csentry/
	`onComplete: ((res: Array<String>) -> Unit)`	Callback function for successful command completion. “res” argument is an array of listed files and directories. If an element of this array ends with “/” this element represents a directory, otherwise it's a file
	`onError: ((err: String) -> Unit)`	Callback function for unsuccessful command completion.
`makeDirectory`		Creates a subdirectory inside csentry/. If directory already exists, does nothing
	`directory: String`	Path to a directory to be created. May be a “deep path”. In this case all parent directories will be created.
	`onComplete: ((res: Array<String>) -> Unit)`	Callback function for successful command completion. “res[0]” returns the same value as “directory” argument
	`onError: ((err: String) -> Unit)`	Callback function for unsuccessful command completion.
`pushFiles`		Transfers files accessible to the client app to csentry/ directory
	`localSourceDirectory: String`	Locally accessible directory from which the files are transferred to csentry/ directory
	`pattern: String`	Files/directories that match this pattern are included in the transfer. May contain wildcard characters such as “*” and “?”. Pattern matching ignores case.

		In general the pattern matching rules are the same as for wildcard_filter argument in CSPro dirlist() function
	`remoteDestinationDirectory: String`	Path to directory inside csentry/ directory. If this directory doesn’t exist, it will be created
	`includeSubdirectories: Boolean`	If “true” the pattern search will recursively include all subdirectories of “localSourceDirectory”. The subdirectory structure will be recreated in the destination directory. If “false” only top directory will be included in pattern search
	`overwirteExistingFiles: Boolean`	If “true” existing files in the destination directory will be overwritten by files included in this transfer.
	`onComplete: ((res: Array<String>) -> Unit)`	Callback function for successful command completion.
	`onError: ((err: String) -> Unit)`	Callback function for unsuccessful command completion.
`pullFiles`		Transfers files from csentry/ directory to a directory accessible by the client app
	`remoteSourceDirectory: String`	Path to directory inside csentry/ directory.
	`pattern: String`	Files/directories that match this pattern are included in the transfer. May contain wildcard characters such as “*” and “?”. Pattern matching ignores case. In general the pattern matching rules are the same as for wildcard_filter argument in CSPro dirlist() function
	`localDestinationDirectory: String`	Locally accessible directory to which the files are transferred from csentry/ directory
	`includeSubdirectories: Boolean`	If “true” the pattern search will recursively include all subdirectories of “remoteSourceDirectory”. The subdirectory structure will be recreated in the destination directory.

		If “false” only top directory will be included in pattern search
	`overwirteExistingFiles: Boolean`	If “true” existing files in the destination directory will be overwritten by files included in this transfer.
	`onComplete: ((res: Array<String>) -> Unit)`	Callback function for successful command completion.
	`onError: ((err: String) -> Unit)`	Callback function for unsuccessful command completion.
`delete`		Deletes files or directories in csentry/ directory or its subdirectories
	`parentDirectory: String`	A directory under csentry/ from which the files are to be deleted.
	`pattern: String`	Files/directories that match this pattern are included in the deletion operation. May contain wildcard characters such as “*” and “?”. Pattern matching ignores case. In general the pattern matching rules are the same as for wildcard_filter argument in CSPro dirlist() function
	`deleteFiles: Boolean`	If “true” files will be included in the patten matching.
	`deleteDirectories: Boolean`	If “true” subdirectories of parentDirectory will be included in the pattern matching.
	`onComplete: ((res: Array<String>) -> Unit)`	Callback function for successful command completion.
	`onError: ((err: String) -> Unit)`	Callback function for unsuccessful command completion.