File Picker (AS9102)

If the user never purchased the file from SAE, then that screen shouldn't come up, right? What should come up is: as9102\_info\_page.dart that says: To view the AS9102 Specification, please purchase it from:\n\n" "https://www.sae.org/standards/content/as9102/\n\n" "Then download the file and choose it using the button below.

**ChatGPT said:**

**You are 100% correct!**

If the user **has never purchased or installed** the AS9102 PDF, you should **not** show the file picker immediately.  
Instead, you should show your **AS9102InfoPage** (from as9102\_info\_page.dart) that explains how to purchase and install the file.

**Recommended Flow**

1. **User taps “AS9102” in your app.**
2. **Your code checks:**  
   *Does as9102.pdf exist in the app’s documents folder?*
3. **If the file exists:**  
   → **Open the viewer** (AS9102ViewerPage) or let the user view the file.
4. **If the file does not exist:**  
   → **Show AS9102InfoPage** explaining:
   * Where to purchase the file (with your helpful text and link)
   * How to install it
   * A button to “Retrieve AS9102” (which launches the file picker when they’re ready)
   * **A Go Back button** to leave this screen if they wish

**What NOT to do**

* **Do not open the file picker right away**—that only makes sense if they are ready to select a file (which they might not have yet).
* **Don’t show a “file not found” error by itself**; instead, guide them with helpful instructions.

**Sample Text for AS9102InfoPage**

sql

CopyEdit

To view the AS9102 Specification, please purchase it from:

https://www.sae.org/standards/content/as9102/

Then download the file and choose it using the button below.

All AS9102 buttons will be unavailable without the as9102.pdf file being installed.

Be sure the file is named 'as9102.pdf' once it's on your phone.

And a button: **Retrieve AS9102** (opens the picker), plus a **Go Back** button.

**In Short:**

* **Only show the file picker** after the user has indicated they’re ready to “retrieve”/install the file.
* **Always show your info page** first if the file is missing!

Let me know if you want a little code for this check, or to wire this up more tightly!  
You have the right approach.

How do I go about my development checking for which page shows up when. If I delete my purchased as9102.pdf file off of my phone I can go through some scenarios and then put the file back on my phone and go through other scenarios. Does that sound correct?

**ChatGPT said:**

**Yes, that's exactly how you should test!**  
This is a very good and realistic way to verify the behavior and make sure users have a smooth experience.

**Here’s How to Do It:**

**1. To Test the “No AS9102 File” Scenario:**

* Delete (or temporarily move/rename) as9102.pdf from your phone’s app storage/documents directory.
* Now open your app and tap the AS9102 button.
* **You should see:**
  + Your info page (AS9102InfoPage) telling the user how to purchase and add the file.
  + *You should not see the PDF viewer, and the file picker should only appear if the user taps "Retrieve AS9102".*

**2. To Test the “File Exists” Scenario:**

* Put the correct as9102.pdf file back in your app's documents directory (or use the picker to add it).
* Open your app and tap the AS9102 button.
* **You should see:**
  + Either the PDF opens immediately, or you give the user an option to view or replace the file.

**3. To Test “Pick New File” Scenario:**

* While the file is missing, use your app’s “Retrieve AS9102” button to open the picker.
* Select the file and make sure it copies to the correct place and opens as expected.

**Tips for Reliable Testing:**

* **Use your phone’s file manager** to view and manage files in the app’s storage location (may be under Android/data/your.app.package.name/files or similar).
* Sometimes the app’s “Documents” directory is not easy to browse with some file manager apps (especially on Android 11+).
  + If you have trouble, let me know your device and Android version and I’ll give specific steps!
* You can also **uninstall/reinstall the app** to fully clear app storage and start fresh.

**Bonus:**

If you want to automate some of this (e.g., add a “Delete AS9102” debug/test button during development), you could write a quick button in a test menu to delete the file for you. Just remove this before production release.

**Bottom Line:**

* **Yes!** Manually adding/removing the file, then running your app and seeing what screen appears, is the perfect way to test your logic.
* Do both scenarios and you’ll know your app is robust!

Let me know if you need code to check if the file exists, or want a little sample for a debug/test button to delete or check for the file!

what directory does the as9102.pdf reside in

**ChatGPT said:**

Great question!  
When your Flutter app uses the path\_provider package to get the app’s documents directory, the **AS9102.pdf** file is copied to your app’s *private documents directory*, which is:

**On Android Devices:**

The path is typically something like:

bash

CopyEdit

/storage/emulated/0/Android/data/<your.app.package.name>/files/as9102.pdf

* <your.app.package.name> is your app’s application ID, e.g., com.example.fai\_assistant or whatever your app’s package is named.

**How the Path Is Determined:**

* In your code, you probably use:

dart

CopyEdit

final appDir = await getApplicationDocumentsDirectory();

final destination = File('${appDir.path}/as9102.pdf');

* getApplicationDocumentsDirectory() returns the directory above.

**How to Access This Directory on Your Device:**

* **Some file manager apps** can see this folder, but many “user-friendly” ones **hide** the /Android/data/ folder by default (especially on Android 11+).
* To see it, you may need a more advanced file manager app (e.g., CX File Explorer, File Manager by InShot, or connect your phone to your computer).
* On some phones, you may need to grant the file manager special permissions.

**On Older Android Versions:**

* Sometimes, getApplicationDocumentsDirectory() points to something like /storage/emulated/0/Documents/
* But for all modern apps, it will be the **app’s private directory** in /Android/data/.

**How to Test**

* When you “pick” a file, your code copies it to that directory as as9102.pdf.
* When your app needs to access it, it always looks in that directory.

**Let me know your app’s package name and Android version if you want the *exact* path or step-by-step instructions for your device!**

**Summary:**  
The file will be at:

bash

CopyEdit

/storage/emulated/0/Android/data/<your.app.package.name>/files/as9102.pdf

unless you’ve changed the storage path in your code.

If you need help navigating to that location or want to write a little debug code to show the actual file path in-app, let me know!

Should I have a function in my app that allows the user to delete the file?

**ChatGPT said:**

**That’s a very thoughtful idea—and for development, it can be super helpful!**

**Should You Provide a “Delete AS9102 File” Function?**

**For Production Users:**

* **Usually, no.**  
  Most users won’t need to delete the AS9102.pdf file unless they want to “replace” it, in which case you can just allow them to pick a new file and overwrite the old one.
* **However:**
  + If you want to allow users to “clear” the file and return to the info screen (maybe in app settings or for troubleshooting), you could add a “Delete AS9102” button.
  + Just be sure to:
    - Add a confirmation dialog (“Are you sure?”)
    - Explain that all AS9102 features will be unavailable until they add the file again

**For Development/Testing:**

* **Yes!**  
  It’s very handy to have a hidden or debug-only button that deletes the file, so you can quickly test all scenarios.

**Sample Code: Delete AS9102 File Button**

Here’s a quick function and button to add **wherever you want** (maybe in a debug page, or even next to your “Retrieve AS9102” button):

dart

CopyEdit

import 'dart:io';

import 'package:path\_provider/path\_provider.dart';

// Call this function when user taps "Delete AS9102"

Future<void> deleteAS9102File(BuildContext context) async {

final dir = await getApplicationDocumentsDirectory();

final file = File('${dir.path}/as9102.pdf');

if (await file.exists()) {

await file.delete();

ScaffoldMessenger.of(context).showSnackBar(

const SnackBar(content: Text('AS9102.pdf deleted.')),

);

} else {

ScaffoldMessenger.of(context).showSnackBar(

const SnackBar(content: Text('File does not exist.')),

);

}

}

Add a button like:

dart

CopyEdit

ElevatedButton(

onPressed: () => deleteAS9102File(context),

child: const Text('Delete AS9102 File'),

),

**Where to Put This Button:**

* **For development:** On a hidden debug/dev screen.
* **For production:** Maybe in a “Settings” or “About” page, with good warnings and confirmation dialog.

**TL;DR**

* **You don’t *need* a delete button for most users,** but it’s great for testing or if you want to give users extra control.
* **If you do add it:** Always ask “Are you sure?” first.

Want a ready-to-paste code block with confirmation dialog?  
Let me know if you want it in a specific screen, and I’ll write it for you!

lAfter I do delete as9102.pdf off of my phone it won't exist anywhere there right? I'll have to email to myself and put it into a folder ono my phone. I know I programmed this in the beginning but I've forgotten some stuff. What if a user purchases the file on their computer instead of the phone...but then again they should just use the app to do it right? Also when they do purchase it from SAE it will automatically put it into the downloads folder on their phone, correct? Is that the first place the file picker checks?

**ChatGPT said:**

These are **excellent questions**—and honestly, most real users will wonder the same things. Here’s what actually happens and what’s “best practice” in this scenario:

**What Happens When You Delete as9102.pdf**

* **If you use the code I gave you, it deletes the file from your app’s private documents directory.**
* After deletion, the file does **not** exist anywhere else on your phone **unless** you have a copy in another folder (like Downloads, Documents, or email attachments).
* **Your app will not see the file anymore, and will prompt the user to add it again.**

**If User Purchases/Downloads the File on Computer Instead of Phone**

* This is very common!
* **What they’ll do:**
  + Buy/download as9102.pdf on their desktop/laptop from the SAE website.
  + They’ll need to **transfer** it to their phone/tablet, usually via:
    - Emailing it to themselves and opening the attachment on their phone.
    - Using a cloud service (Google Drive, Dropbox, OneDrive, etc.)
    - Using a USB cable to move the file directly to their phone.
* **Your app will still work**—as long as the user can *find* the PDF file with the file picker.

**If User Purchases on Their Phone**

* The browser (Chrome, Samsung Internet, etc.) will likely save as9102.pdf to the default **Downloads** folder:

swift

CopyEdit

/storage/emulated/0/Download/

or sometimes

swift

CopyEdit

/storage/emulated/0/Downloads/

**How Does the File Picker Work?**

* When you use the file\_picker package and launch the picker:
  + **The user can browse all folders** they have permission to see.
  + **The first screen** they see is usually the Downloads folder (but it can depend on the Android version, phone manufacturer, and file picker app defaults).
  + **The user can always navigate to other locations** (Documents, cloud storage, etc.) to find the file.

**What Should You Tell Users?**

You might want to include a short instruction like:

“If you bought AS9102 on your computer, email it to yourself or upload it to Google Drive/OneDrive and download it onto your phone.  
If you purchased on your phone, look for it in your Downloads folder when picking the file.”

**To Summarize:**

* **After you delete the app’s copy, the file is only on the phone if the user manually put it there.**
* **Purchasing from SAE on a phone:** goes to Downloads by default.
* **Purchasing on computer:** user must transfer it.
* **File picker usually opens in Downloads**, but user can pick from anywhere.

**PRO TIP:**

If you ever want to auto-detect or suggest the Downloads folder in your app, you can use getExternalStorageDirectory() for advanced custom code, but for most users, just tell them to look in “Downloads” in the picker.

**Let me know if you want to add a little “How To” help text for users, or need sample code to handle these scenarios!**

As we discussed yesterday about purchasing a marked file (metadata) from SAE, will that change a lot of my coding to accept and file picker the file?

**ChatGPT said:**

**Good news:**  
No, it won’t require much (if any) code change for your file picker or file-acceptance logic, even if SAE starts delivering “marked” (metadata-stamped, watermarked, or user-identified) PDF files!

**Here’s Why:**

1. **The file is still a regular PDF.**
   * It might have a visible/invisible watermark or internal metadata, but for your app, it’s just a PDF file.
2. **The user still downloads it the same way.**
   * Whether they buy it on their phone or computer, they receive a PDF file (named as9102.pdf or sometimes with a unique ID or user info).
3. **Your file picker doesn’t care about internal metadata.**
   * Your code just copies the selected PDF to your app’s folder.
   * The file can be named anything (if you want), as long as you copy/rename it to as9102.pdf internally if your app expects that name.

**What *Might* Change?**

* If SAE starts using a **different default filename** (like as9102\_123456\_user.pdf), you might want to:
  + Update your user instructions:  
    *“Select your purchased AS9102 PDF (it might be named something like as9102\_123456\_user.pdf) and it will be installed automatically.”*
  + OR, allow any PDF filename to be picked, but always copy it as as9102.pdf inside your app (which you already do).
* **If SAE ever uses a proprietary PDF reader or DRM** (very unlikely for a standards doc), *then* you’d have trouble—but regular watermarked PDFs will not break your picker.

**Bottom Line:**

* **Your current file picker and copy-to-app-storage code will continue to work.**
* No code changes needed for metadata/watermarked files.

**You may want to update your “help” or instructions** to users, just to mention:

“The AS9102 file you purchase may include your name or a code in the filename or in the document itself. This is normal and does not affect its use with FAI Assistant.”

**If you want, I can draft some new user instructions or code snippets to future-proof your file picker (e.g., allow any PDF filename). Just let me know!**

You’re in good shape with your current approach!

When SAE uses the watermark, FAI Assistant will have to check for that code to validate it, is that much recoding?

**ChatGPT said:**

**That’s a fantastic, forward-looking question!**  
Here’s the detailed answer, including what would change, and how much recoding it would mean.

**1. If SAE Watermarks PDFs but Makes No Technical Restrictions**

* **Most likely scenario:** The PDF is watermarked with the buyer’s name/email/order number, *visibly* or as metadata, but there’s **no technical restriction**.
* **Your app** can continue to open, read, and display the PDF with **no code changes required**.
* **No validation needed**—your app does not care what watermark or code is present.

**2. If SAE Requires Validation of a Code in the Watermark or Metadata**

* **Rare, but possible:** SAE could embed a serial code or order ID and require apps like yours to check for its presence before allowing viewing.
* **What this would require:**
  + **Parsing the PDF** for visible text watermarks (e.g., searching for “Order #123456” on a page)
  + **OR, reading PDF metadata** to look for a key-value code.
* **How much work?**
  + **You’d need to use a PDF-parsing package** (like pdf, pdf\_text, syncfusion\_flutter\_pdf, or a native bridge) to extract and search for that watermark/code.
  + **Add code to:**
    - Open/read the PDF file
    - Search for the required text or metadata
    - Only “accept” the file if the code matches
  + **Not a massive rewrite,** but it’s more work than a regular file picker—mainly new code to check file contents or metadata, plus some UI changes to show errors if validation fails.

**3. If SAE Uses DRM or Encryption**

* **Highly unlikely** for a standards PDF, but if they did:
  + Your app could not open the file at all without their reader.
  + **This would break your current workflow and require a totally new solution.**
  + This scenario is *not* what is being discussed by SAE as of mid-2025.

**Summary Table**

| **Scenario** | **Required Change** | **Level of Effort** |
| --- | --- | --- |
| Visible/metadata watermark only | No change needed | None |
| Validation required | Add PDF text/metadata search | Moderate (some new code, but not a full rewrite) |
| DRM/encrypted file | Major rework, likely not possible | Major, likely infeasible |

**Practical Advice**

* **Wait for SAE’s final requirements.**  
  (If they ask for validation, they’ll probably provide specs/examples for what to check.)
* **If/when you need to validate:**
  + *Yes, it’s some new code,* but not a full rewrite—just extra steps after picking a file.

**If you want to be ready:**

Let me know if you want a sample of how to extract text or metadata from a PDF in Flutter, so you can scan for a code if/when it becomes necessary!  
But right now, **you don’t need to recode anything.** Your app is “future friendly.”

**TL;DR:**

* *No change needed unless SAE requires your app to validate a watermark/code.*
* *If validation is needed: moderate work, but your app’s core logic and workflow remain the same.*

I know that SAE will want me to verify purchase of the file before it's loaded.

**ChatGPT said:**

That’s a **very smart assumption**—and you’re probably right:  
**SAE will likely require that FAI Assistant (and any app) verify the user has purchased a legitimate copy of AS9102** before displaying the spec, to comply with copyright and licensing.

**How Could FAI Assistant Do This?**

**A. If SAE uses a watermark or embedded code:**

* Each purchased PDF will be stamped with a unique code (name, order #, or something similar).
* **Your app will need to:**
  + **Read the PDF (or its metadata) after the user picks it.**
  + **Verify** that the required info is present (e.g., “John Smith,” “Order 12345”).
  + **Show a message if it’s missing or invalid** (“This file is not a valid AS9102 purchase. Please purchase at SAE.org.”).

**B. If SAE uses a purchase verification API:**

* They may provide a **web service** where you submit the watermark or metadata value and get back “Valid” or “Invalid.”
* Your app would:
  + Extract the code from the PDF.
  + Send it to SAE’s verification endpoint.
  + Only allow viewing if it’s confirmed as legitimate.

**C. If SAE requires a login or serial number:**

* This would require UI and backend changes (user logs in or enters a license).
* **Less likely**, but you’d still just need to verify before showing the file.

**What Does This Mean for Your Code?**

* **File picking stays the same** (user still selects a PDF).
* **Before you copy/use the file:**
  + Read the PDF for the unique code.
  + Optionally, check with SAE’s servers.
* **Only if verification passes:**
  + Save/display the PDF.
* **If not:**
  + Show an error/help message and refuse to load the file.

**You’ll need to:**

* Use a package to extract text or metadata from PDF (not built into Flutter).
* Add some logic to your picker handler.

**What To Do Now?**

* **No changes needed yet.**
* **When SAE finalizes their watermark/validation method:**
  + Add PDF reading/verification code (which I can help you with!)
  + Update instructions to users if needed.

**What If User Tries to Load a Non-purchased File?**

* Your app will detect it doesn’t have the required watermark/code/metadata, and will:
  + Show a clear message:  
    *“This file does not appear to be a legitimate AS9102 purchase. Please purchase from SAE International.”*
  + Prevent access to the content.

**Summary**

* **You’re on the right track.**
* You will need to add *some* code to check the file after picking (once SAE tells you what to check for).
* The process is:  
  **Pick file → Extract/verify code → If valid, save/display. If not, show message.**

**Let me know when you get more info from SAE, and I’ll show you exactly how to implement this with real code!**  
If you want, I can show you now how to extract visible text or metadata from a PDF, just for practice.