

PDF Pro Panel 3

Module Developer Guide

Build, package and publish a tool of your own

This guide is complete on its own. It carries a working module in full and specifies the package format exactly, so nothing beyond PDF Pro Panel 3 and a text editor is needed to build one.

Everything a module does happens on the user's own computer.

Applies to	PDF Pro Panel 3 Desktop 1.14.2 and later, Windows and macOS
Revision	2026-10-02
Terms	pdfpropanel.com/module-developer-agreement
Contact	modules@pdfpropanel.com

(c) 2026 CCI Software Toolkits. All rights reserved.

Contents

Welcome	3
1. How a module comes to run	3
2. Your first module in four steps	3
3. The manifest, field by field	4
4. The script	6
5. What the application gives you	7
6. Permissions	8
7. Packaging and limits	9
8. Updates, paid modules and upgrades	9
9. What a module must not do	10
10. When a module will not run	10
11. Getting in touch	11

Welcome

A module adds a tool to PDF Pro Panel 3 Desktop. It is one JavaScript file and one manifest, loaded into the application as a plain script. There is no build step, no bundler, no framework and nothing to install: a text editor is enough.

This guide is complete on its own. It carries the whole of a working module, so you can copy it from these pages, and it specifies the package format exactly, so you can produce one with any zip tool you already have.

Everything a module does happens on the user's own computer. PDF Pro Panel's central promise to the people who buy it is that their documents never leave their machine, and a module is inside that promise rather than an exception to it. Please read Section 9 before you publish anything.

1. How a module comes to run

Two kinds of module exist. Eighteen ship inside the application: they are part of it, can be turned off but not removed, and need no permission from the user beyond installing the application. Yours is the other kind.

A module a user installs runs only because that user was shown what it was and allowed it. Before anything is put in place, PDF Pro Panel shows them your module's name, version, the author you declared and the exact list of what it is asking to be allowed to do, all read from the manifest inside the package rather than from the download page it came from. They confirm, or nothing is installed.

The application then records a fingerprint of every file in your module and checks it again each time it starts. If any file changes, the module stops running and the user is asked again. That covers your own updates as well as anything less welcome.

What PDF Pro Panel does	refuses to run a module the user has not approved, and notices if one changes afterwards
What it does not do	review, test, audit, certify or endorse your module. Approval is the user's decision.
What the fingerprint proves	only that the module has not changed since they approved it. Nothing about its origin or behaviour.
What we can do later	refuse a module by its id, or by one particular build of it, in an application update, if it turns out to be harmful

Please note. There is no signing key and we do not sign modules. If you receive a request bearing our name asking you to sign a module with a key we supplied, or to send us a private key, it is not from us. Please report it to security@pdfpropanel.com.

2. Your first module in four steps

Step one. Make a folder and put two files in it, named exactly as shown. The names of the folder and the files may contain letters, digits, dots, dashes and underscores only, and must begin and end with a

letter, digit, dash or underscore.

```
my-module/  
plugin.json  
hello-module.js
```

Step two. Copy the manifest from Section 3 and the script from Section 4 into those two files, and change the "id" to your own. The id is your module's name everywhere: its folder on the user's computer, its entry in the Module Manager, and the thing an update replaces. Choose it once and keep it.

Step three. Try it before packaging it. In PDF Pro Panel open Settings & Advanced > Module Manager, switch on Developer mode at the foot of the window, and use Open module folder. Put your folder inside the folder that opens, naming it exactly your module's id, then restart PDF Pro Panel. Developer mode runs a module without asking you to approve it each time you rebuild, which is the only comfortable way to iterate. Turn it off when you have finished: it applies to every module in that folder, not only yours.

Step four. Package it. Section 7 gives the format.

3. The manifest, field by field

plugin.json describes your module to the application and to the person installing it. Required fields are marked. A manifest that breaks any rule below makes the module appear in the Module Manager as "Not usable", with the reason shown on its card.

```
{  
  "id": "com.example.hello",  
  "name": "Hello Module",  
  "version": "1.0.0",  
  "description": "What this module does, written for the person deciding whether to install  
it.",  
  "author": "Your Name",  
  "category": "utility",  
  "licenseTier": "free",  
  "engines": { "app": ">=1.14.2" },  
  "activationEvents": ["onDemand"],  
  "permissions": ["pdf:read", "ui:modal", "ui:dashboard", "storage:isolated"],  
  "contributes": {  
    "tools": [{  
      "id": "hello",  
      "title": "Hello Module",  
      "group": "Utilities",  
      "description": "One line, shown on the dashboard card.",  
      "category": "utility",  
      "icon": "\uD83D\uDC4B",  
      "primaryActionText": "Write the report",  
      "requiresDocument": true,  
      "supportsBatch": true,  
      "entry": "hello-module.js"  
    }],  
    "commands": [{  
      "id": "utility.hello",  
      "title": "Hello Module",  
      "keywords": ["hello", "example", "report"]  
    }]  
  }  
}
```

id (required)	Lower case, reverse-DNS, at least two parts: com.yourname.thing. Letters, digits, dots and dashes. 64 characters at most, because it becomes a folder name.
name (required)	Three characters or more. This is the first thing the user reads in the install dialog.
version (required)	Three numbers: 1.0.0. A higher version replaces an installed copy with the same id.
description	Shown in the install dialog and on the module's card. Write it for the person deciding whether to trust you, not for a search engine.
author	Shown in the install dialog as "by ...". Be accurate: misrepresenting authorship is grounds for refusal.
category (required)	Exactly one of: organize, security, convert, extract, advanced, legal, editor, utility.
licenseTier	"free" or "pro". "pro" marks the module as needing a licensed application, and is a label rather than a lock.
engines.app	A version range, such as ">=1.14.2". If the running application falls outside it the module reports "Needs a newer version".
activationEvents	["onDemand"] loads your code when your tool is first opened. Omit the field and it loads at startup instead. Prefer onDemand.
permissions	See Section 6. Ask for as little as you need; each one is a line the user reads next to your name.
dependencies	Optional, as { "com.other.module": ">=1.0.0" }. Your module waits, listed as "Missing a dependency", until that one is available.
entitlement	Optional. See Section 8. Only for a module sold separately.

Each entry in `contributes.tools` adds one card to the dashboard:

id (required)	Lower case letters, digits and dashes only: kebab-case. An underscore or a capital letter refuses the module.
title (required)	The card's heading.
category (required)	One of the eight, as above.
entry (required)	The filename of your script, in the module's own folder. Not a path, not a subfolder, not a web address. 4 MB at most.
group	Which dashboard section the card sits in, e.g. "Utilities".
icon	One character or emoji.
badge, badgeColor	A small label on the card, and its colour as a hex value.

primaryActionText	The main button in your dialog.
requiresDocument	true if the dialog should ask for a file first.
supportsBatch	true if it accepts more than one.

contributes.commands adds an entry to the Ctrl+K palette, and contributes.viewerActions adds a button to the PDF viewer. Both are optional and take an id, a title, and for a command a list of keywords.

4. The script

One file. It registers one object, and that object is the whole of the interface between your module and the application. Copy this and change it.

```
(function () {
  'use strict';

  const fields = [
    { type: 'file', id: 'source', icon: '[]',
      label: 'Click or Drag & Drop Documents Here',
      hint: 'One or more files', multiple: true },
    { type: 'text', id: 'note', label: 'A note for the top of the report',
      placeholder: 'Optional' }
  ];

  const controller = {
    fields: () => fields,

    renderModal(toolId, container, ctx) {
      container.textContent = '';
      this._form = ctx.ui.form(fields);
      container.appendChild(this._form.element);
      ctx.ui.setPrimaryButtonText('Write the report');
    },

    async execute(toolId, selectedFiles, options, ctx) {
      if (!selectedFiles || !selectedFiles.length) {
        throw new Error('Please choose at least one document.');
```

```

const previous = (await ctx.storage.get('runs')) || { count: 0 };
await ctx.storage.set('runs', { count: previous.count + 1 });
} catch (_) {
// the user may have withdrawn this permission; carry on without it
}

return {
  success: true,
  bytes: new TextEncoder().encode(lines.join('\n')),
  filename: 'report',
  ext: 'txt'
};
}
};

if (typeof window !== 'undefined' && window.PluginManager) {
  window.PluginManager.registerModule(controller);
}
if (typeof module !== 'undefined' && module.exports) {
  module.exports = { controller, fields };
}
})();

```

The functions the application will call, all optional except execute:

fields()	Describes your dialog before your code has loaded, so the application can show the card.
onLoad(ctx)	Called once when the module activates. It blocks the window while it runs, and a module gets ten seconds to load, so do nothing slow here. Three failures in a row and the module is held back until the user retries it.
renderModal(toolId, container, ctx)	Builds your dialog. Called each time the tool is opened, with an empty container.
onFilesSelected(files, ctx)	Called when the user picks files in your dialog.
execute(toolId, files, options, ctx)	Does the work. files is an array of { name, size, data }, where data is a Uint8Array. options holds the values from your fields.

What execute returns is what the application saves, through the same save dialog and the same file history as every built-in tool:

```

return { success: true, bytes: <Uint8Array>, filename: 'report', ext: 'txt' };

```

If your tool produces nothing to save, show the result with `ctx.ui.showResult(...)` and return `{ success: true }`.

To fail, throw an `Error`. Its message is shown to the user word for word, so write it for them: "Please choose at least two PDFs to merge." rather than "ENOENT: no such file".

5. What the application gives you

Your `execute` and `renderModal` receive a context object carrying only what the user has allowed. Anything beyond that throws.

```

ctx.pdf.hasReadAccess()           // is pdf:read granted right now?
ctx.pdf.hasWriteAccess()          // is pdf:write granted right now?
ctx.pdf.getActiveDocument()       // the document open in the viewer, or null
ctx.pdf.getPageCount()            // its page count, or 0
ctx.pdf.getEngine()               // the application's PDF engine

ctx.ui.form(fields)                // build your dialog from your field list
ctx.ui.createDropzone(...)         // or the pieces, if form() is not enough
ctx.ui.createFormGroup(...)
ctx.ui.createInput(...)
ctx.ui.createSelect(...)
ctx.ui.setPrimaryButtonText(text)
ctx.ui.setPrimaryButtonEnabled(on)
ctx.ui.reportProgress(message)     // for work taking more than a moment
ctx.ui.showResult(...)             // a result read on screen rather than saved
ctx.ui.notify(message, type)       // 'info' | 'success' | 'error'
ctx.ui.copyToClipboard(text)

ctx.storage.get(key)               // your module's own storage, not the application's
ctx.storage.set(key, value)

ctx.jobs.run(operation, data, options) // heavy work in a separate process

```

Field types for `ctx.ui.form` are file, text, number, select, checkbox and password.

6. Permissions

Every permission you list appears in the install dialog and on your module's card, where the user can switch any of them off at any time, including while your module is running. Ask for as little as you need, and handle being refused.

pdf:read	Read the open document
pdf:write	Change the open document
ui:modal	Show its own dialog
ui:dashboard	Add a card to the dashboard
ui:viewer	Add a button to the viewer
storage:isolated	Keep its own settings
jobs:background	Run background jobs

Anything not on that list is a validation error and the module will not run. Because a permission can be withdrawn while your module is loaded, guard the calls that depend on one:

```

try {
  await ctx.storage.set('runs', count);
} catch (_) {
  // the user turned this off; carry on without it
}

```

Please note. The permission list describes and limits the interface the application gives your module. It is not an operating-system sandbox: a module is program code running inside the application. That is precisely why users are shown what you ask for and why Section 9 exists.

7. Packaging and limits

A module is distributed as a single file ending `.ppmodule`. It is an ordinary zip archive, so any zip tool will make one. Zip the contents of your module folder, not the folder itself, then rename the archive:

```
cd my-module
zip -r ../com.example.hello.ppmodule .
# or, on Windows: select the files inside the folder,
# Send to > Compressed (zipped) folder, then rename the .zip to .ppmodule
```

A package whose files sit one folder down is also accepted, since that is the commonest way to zip one by hand. Name the file after your module id; it keeps versions apart on a download page.

Archive format	zip, stored or deflated only. No zip64, no encryption, no symbolic links.
plugin.json	64 KB at most
Your entry script	4 MB at most
Files in a module	256 at most
Unpacked size	32 MB at most
The .ppmodule file	32 MB at most
Folder depth inside it	8 at most
File and folder names	letters, digits, dot, dash, underscore; must begin and end with a letter, digit, dash or underscore

Please note. That last rule is stricter than your own computer's, deliberately. It means no leading dots, so a `.git` or `.env` folder cannot ride along in a package by accident, and no name ending in a dot or a space, which Windows treats as the same file as one without. A package breaking it is refused as a whole rather than having that one file skipped.

8. Updates, paid modules and upgrades

An update is a new `.ppmodule` with the same id and a higher version. Installing it replaces the copy already there, the user is asked once about the new version, and the Module Manager tells them what it replaced. If anything goes wrong mid-install, the copy they already had is left in place.

Two separate things govern paid modules, and they are easy to confuse:

licenseTier: "pro"

Means your module expects a licensed application. The Module Manager marks it Pro. It is a label: the application already refuses to process documents without a live licence or trial, so a pro module loads and then says so when used.

entitlement

Means your module has been bought on its own. Set it to a name in the same form as an id, and the module runs only if the user's licence carries that name. This is enforced and fails closed: no name, no module.

A module with an entitlement the licence does not carry is listed as "Not purchased", with your name and description visible so the user can see what they would be buying, and is given no code to run. Entitlement names reach a licence from our licensing service on activation and at every check-in, so one bought after activation appears without the user reactivating, and a refunded one stops working. A trial confers none.

A premium version of an existing module is a separate module with its own id and the entitlement the buyer receives. An extension to one is a module that declares a dependency on it.

Selling or distributing a module means accepting the Module Developer Agreement at pdfpropanel.com/module-developer-agreement. It covers what you keep, what you warrant, how paid modules work, and what happens if a module turns out to be harmful. Read it before you publish.

9. What a module must not do

A module doing any of these will not be listed by us, will be delisted if it already was, and can be refused by id in an application update:

- * read, move or send anything the user did not hand to the tool they opened
- * reach the network without saying so plainly in its description, before it is installed
- * collect anything about the user or their documents: no telemetry, no analytics, no advertising, no tracking
- * ask for a permission it does not use
- * try to reach another module's storage, the application's settings or licence state, or the file system beyond what the user provided
- * verify, circumvent or interfere with the application's licensing
- * misrepresent who wrote it, or imitate another module or product
- * ship obfuscated, packed or minified code that cannot be reviewed

If you find a security problem in a module of yours that users already have, tell us at security@pdfpropanel.com and publish a fix. We can refuse a single build by its fingerprint, which leaves your other versions working, so reporting it early costs you much less than leaving it.

10. When a module will not run

The Module Manager states the reason on the card, and Copy diagnostics at the top of the window puts the whole picture on the clipboard. Attach that when you ask us for help.

Running

active

Loads when used	fine, not yet needed
Waiting for you	installed but not yet approved. Review and approve, or switch on developer mode.
Changed - not running	a file changed since it was approved. Expected while you are editing; otherwise find out what changed it.
Turned off	the user turned it off
Did not start	your script threw, or took longer than ten seconds
Held back	failed three times in a row. Try again clears it.
Needs a newer version	engines.app excludes this build
Missing a dependency	waiting for another module
Not purchased	it names an entitlement this licence does not have
Not usable	the manifest is wrong - the reason says how
Blocked by PDF Pro Panel	we have refused this module. Nothing overrides this, including developer mode.

The common causes, in the order they actually happen: a category that is not one of the eight; a tool id that is not kebab-case; an entry naming a file that is not there, or naming a path instead of a filename; an engines.app range the running application falls outside; and a module copied into the folder by hand rather than installed, which is waiting to be approved. If the tool opens but nothing happens, your renderModal threw: the log is under Settings & Advanced > Open Log Folder.

11. Getting in touch

Submitting a module	modules@pdfpropanel.com
Security, including in your own module	security@pdfpropanel.com
Anything else	support@pdfpropanel.com
The developer terms	pdfpropanel.com/module-developer-agreement
How modules affect users	pdfpropanel.com/license-agreement, the Modules section

You support your own module. Please do not send the people who install it to our support address; we will send anyone who writes to us about your module on to you.