Page contents

- Set up a test project and script sources
- Write a minimal activation script
- Compile, test, and diagnose
- What this example does not cover
- Sources

Offline copy of Papyrus activation, compiling and diagnostics .

Fallout 4 Papyrus: an activation script from source to release

Documentation status: In progress. **Sources checked:** 2026-11-11. **Technical validation:** source-reviewed example; it has not been compiled or run in a local Fallout 4 installation for this publication. The procedure targets PC Creation Kit projects, not console development. Verify the game, editor, and script-source versions in your own installation before shipping.

Papyrus is Fallout 4's gameplay scripting language. A `.psc` file is editable source; the compiler produces a `.pex` file for the game to execute. The Papyrus compiler documentation describes this source/output distinction and warns that Skyrim and Fallout 4 compiled scripts are not interchangeable. A plugin can hold a script attachment and property values while the compiled script is a separate file to package.

This chapter makes one small, observable behavior: a placed object writes a diagnostic line when the player activates it. It deliberately avoids changing inventory, quests, or saves. Once this works, the same event can become the entry point for a real feature.

Set up a test project and script sources

Use a dedicated Fallout 4 test profile and a new plugin with `Fallout4.esm` as a master. Keep an untouched copy of the plugin and a disposable test save. Install the Creation Kit for the game build you are targeting. Its distributed script-source archives supply declarations such as `ObjectReference.psc`, `Game.psc`, and `Debug.psc`; the BGS

Handbook's Fallout 4 setup notes describe the source ZIPs under `Data/Scripts/Source` and the `Institute_Papyrus_Flags.flg` compiler flags file. Confirm the actual paths in your installation instead of copying a path from an older release without checking it.

Keep your own source under a distinct project name and record its location. In the Creation Kit, compile through the script manager or the script attachment workflow. For a command-line build, the Fallout 4 compiler reference documents `-import`, `-output`, and `-flags`. These switches identify the source search paths, compiled output directory, and user-defined flags file. Do not reuse a Skyrim compiler or flags file. Inspect the compiler's version banner and errors before assuming a `.pex` was produced.

Write a minimal activation script

Create `ModDocsActivationProbe.psc` with this original example:

```
Scriptname ModDocsActivationProbe extends ObjectReference

Event OnActivate(ObjectReference akActionRef)
    If akActionRef == Game.GetPlayer()
        Debug.Trace("ModDocsActivationProbe: player activated the referen
    EndIf
EndEvent
```

The Fallout 4 `ObjectReference.OnActivate` reference documents the event parameter as the activating reference. The Game.GetPlayer reference returns the player actor, so the condition excludes other activators. Debug.Trace writes a diagnostic message. The example's script and trace text are ours; the API names and signatures are checked against the cited references.

Compile the source, then attach the compiled script to a placed test reference that the player can reach and activate. The object must support activation and the attachment must be saved in your plugin. The exact editor panes and buttons can vary by Creation Kit build; verify the attachment in the saved plugin rather than relying on the script file existing on disk. Test on a fresh save first, then on a disposable existing save. Activate the reference with the player and look for the unique `ModDocsActivationProbe` text in Papyrus diagnostics.

`OnActivate` has important timing limits. The reference notes that the event can arrive even when normal activation is blocked, and that a pickup may already have moved

into inventory by the time the handler runs. A locked door may also send the event on an attempted activation. Do not use this handler alone to infer that an item was successfully collected or a door opened. For those features, check the state you actually need and consult the corresponding event documentation.

Compile, test, and diagnose

Symptom	Check	Next action
Compiler reports unknown base type or function	Fallout 4 source imports and compiler version	Point the compiler at the matching Fallout 4 script sources and retry
Compiler reports a flag error	`Institute_Papyrus_Flags.flg` path	Use the flags file shipped with the matching editor
Compilation succeeds but no event arrives	Script attachment, plugin activation, placed reference, and activation conditions	Reopen the saved plugin and test a reachable reference on a fresh save
Event arrives but no trace appears	Diagnostic logging configuration and compile mode	Test a debug build and inspect the appropriate Papyrus log
Works in development but not in the package	Missing `.pex`, unexpected path, or unlisted dependency	Install the packaged copy in a clean profile with working files disabled

The compiler reference distinguishes debug, release, and final processing. In particular, release processing can remove debug code. A missing trace in a release build is not by itself proof that the event did not run. Keep a separate debug build for diagnosis, and test the actual release behavior with an observable feature rather than a trace alone.

The BGS Handbook setup guide also describes loose-file loading. If a `.pex` is correct but unavailable to the game, inspect the profile's loose-file settings or the archive contents. Do not tell users to overwrite base-game script archives to test your plugin. Preserve the source `.psc`, compiled `.pex`, plugin, editor/game versions, and build notes together so the release can be reproduced later.

What this example does not cover

This is an event and packaging example, not a quest system, custom UI, or F4SE tutorial. It has no properties, persistence, or multiplayer behavior. A later Papyrus chapter must cover property binding, quest aliases, save migration, and a tested gameplay change before the Fallout 4 scripting area can be considered substantial. The official Creation Kit wiki was redirecting to a maintenance page at the time of review, so the cited community-maintained API index and compiler references are identified explicitly.

Sources

- Fallout 4 `OnActivate` event
- Fallout 4 `Game.GetPlayer`
- Fallout 4 `Debug.Trace`
- Fallout 4 compiler switches
- Fallout 4 Papyrus setup and source files