Nexus
Extension Development Guide

The complete reference for building Nexus extensions

nexusprism.org · MIT License

Covers Nexus v1.0.0 and later

About this guide

This is the authoritative reference for building Nexus extensions. It documents every contract surface — every manifest field, every Elixir callback, every JavaScript registration API, every payload — that an extension author needs to ship a working extension.

The example threaded through every section is the Foundation Smoke Test — a real, working extension that exercises every surface in a single install.

1. What an extension is

A Nexus extension is a GitHub repository containing four things:

- A `manifest.json` declaring what the extension contributes.
- An Elixir module implementing the server-side parts.
- An optional JavaScript bundle implementing the browser-side parts.
- A README.

When an admin installs your extension, Nexus fetches the latest release tarball, compiles your Elixir source into the running BEAM VM, runs any migrations through Nexus's own Repo, starts any background processes under Nexus's supervisor, and registers your hooks, routes, slots, and other surfaces in an in-memory ETS registry. Your JavaScript bundle is injected into every page before React mounts.

The in-VM model

There is no separate service to deploy, no webhook delivery, no Docker networking, no inter-service authentication. Event hooks are direct function calls into your loaded module.

What this gets you:

- No version skew. Your code compiles against the exact Elixir/OTP version Nexus is running.
- Shared dependency tree. Ecto, Req, Jason, Oban, Image, Phoenix.PubSub and more — available without declaration.
- Shared database. Use Nexus.Repo directly. Define your own tables; reference Nexus's by string name.
- One artifact. Manifest, source, and JS bundle ship in a single GitHub release tarball.

What it asks of you:

- Trust. Your compiled code runs with the same privileges as Nexus itself.
- Dispatch discipline. Push heavy work to Oban jobs — `handle_event/3` runs synchronously inside Nexus's dispatch task.
- Manifest discipline. Every surface must be declared in the manifest before its implementation will be wired up.

2. Quick start

Scaffold a new extension:

```
mix nexus.extension.new foundation_smoke_test \
  --author "Your Name" \
  --description "Exercises every surface"
cd extensions/foundation-smoke-test
```

To install from the admin panel: Admin → Extensions → Install from URL, paste a GitHub `manifest.json` URL. Nexus reads the manifest, finds the latest release, downloads the tarball, and runs the install pipeline.

Editor validation

Add the schema reference to your `manifest.json` for live validation in VS Code, JetBrains, and other editors:

```
{
  "$schema": "https://YOUR-NEXUS-HOST/manifest_schema.json",
  "manifest_version": 2,
  ...
}
```

Replace `YOUR-NEXUS-HOST` with any Nexus instance hostname. The schema is identical across all instances.

3. Project layout

```
foundation-smoke-test/
■■■ manifest.json          The contract
■■■ mix.exs                Elixir project file
■■■ lib/
■   ■■■ foundation_smoke_test.ex   Elixir module
■■■ priv/
■   ■■■ static/
■       ■■■ foundation-smoke-test.js   JS bundle
■       ■■■ logo.webp                  200x200 icon (optional)
■       ■■■ banner.webp                800x400 hero (optional)
■■■ README.md
```

Where files end up after install

Source path	Installed location
manifest.json	Stored in the extensions.manifest DB column
lib/**/*.ex	Compiled into the BEAM VM — no on-disk artifact

Source path	Installed location
priv/static/.js	/app/uploads/extensions//assets/.js
priv/static/	/app/uploads/extensions//assets/

Assets are served at `/ext/<slug>/assets/<filename>`. Never construct these paths manually — use `Nexus.Extensions.Storage` helpers for runtime files.

4. The manifest — identity & metadata

The manifest is the contract. Every surface your extension contributes must be declared here before Nexus will wire it up. The validator runs at install time and halts on errors with specific field-level messages.

4.1 Required identity fields

```
{
  "manifest_version": 2,
  "name": "Foundation Smoke Test",
  "slug": "foundation-smoke-test",
  "version": "1.7.0",
  "module": "FoundationSmokeTest"
}
```

Field	Constraint	Notes
manifest_version	Must be 2	Only accepted version. Earlier formats are rejected.
name	Non-empty string	Shown on extension cards and admin lists.
slug	^[a-z0-9-]+$, unique	URL prefix, asset directory name. Cannot be renamed after install.
version	Semver string	Compared against GitHub release tag to detect updates.
module	Valid Elixir module name	Used by the loader to find your module after compilation.

4.2 Metadata fields (all optional)

Field	Notes
description	One-sentence summary. Truncated past ~200 chars in card views.
author	Free-form. GitHub username by convention.
homepage	Canonical URL. Must start with http://, https://, or /.
repository	Explicit GitHub URL for release polling when homepage points elsewhere.

Field	Notes
license	Free-form. SPDX identifiers (MIT, Apache-2.0) recommended.
tags	1–4 short strings shown as pills on the extension card.
logo_url	Square icon. 200×200 px PNG or WebP.
banner_url	Wide hero. 800×400 px PNG/WebP/JPEG.
compatible_with	Semver range (^1.0). Currently informational.
js_bundle	Filename of your JS bundle relative to priv/static/. Omit for server-only extensions.

5. The manifest — settings

Settings are configuration values an admin sets per-installation. Stored in `extensions.settings`, passed to every callback, and rendered as form fields under Admin → Extensions → Settings.

5.1 `settings_schema`

```
"settings_schema": {
  "enable_debug_log": {
    "type": "boolean",
    "label": "Verbose handler logging",
    "default": false,
    "description": "When true, logs to server console on every hook fire."
  },
  "api_key": {
    "type": "string",
    "label": "API key",
    "secret": true
  },
  "log_level": {
    "type": "select",
    "label": "Log level",
    "default": "info",
    "options": [
      {"value": "debug", "label": "Debug (verbose)"},
      {"value": "info",  "label": "Info"},
      {"value": "warn",  "label": "Warnings only"}
    ]
  }
}
```

Type	Form control	Stored as
string	Single-line text input	string
text	Multi-line textarea	string

Type	Form control	Stored as
boolean	Toggle switch	true/false
number	Numeric input	number
select	Dropdown	string (requires options)
color	Hex color picker	#RRGGBB string

Attribute	Purpose
label	Form label. Defaults to field key with underscores as spaces.
default	Initial value. Type-appropriate literal.
description	Helper text under the form control.
secret	Masked password input. Stored plaintext — masking is UI-only.
required	Block save while empty.
placeholder	Placeholder text for empty inputs.
options	Required for select. Array of {value, label} objects.

■ `settings_schema` keys and `permissions` keys share the same storage column. Don't reuse a key across both. Convention: `can_verb_noun` for permissions keeps namespaces distinct.

5.2 `settings_tabs`

```
"settings_tabs": [
  {
    "key": "credentials",
    "label": "Credentials",
    "icon": "fa-key",
    "fields": ["api_key", "api_endpoint"]
  },
  {
    "key": "behavior",
    "label": "Behavior",
    "icon": "fa-gear",
    "fields": ["retry_count", "feature_enabled"]
  }
]
```

Groups settings into tabs. Each tab has a key (URL fragment), label, optional icon (short-form Font Awesome class), and fields (keys from `settings_schema`). Fields not listed in any tab render in an implicit "Other" tab.

6. The manifest — backend surfaces

6.1 `hooks`

```
"hooks": [
  {"event": "post_created", "priority": 50},
  "user_registered"
]
```

Events Nexus fires when forum activity happens. Each entry is a string event name or an object with `event` and optional `priority` (default 50). Lower numbers run first.

Event	Fires when	Payload keys
post_created	User creates a post	user_id, post_id
post_updated	Post is edited	user_id, post_id
post_deleted	Post is deleted	user_id, post_id
reply_created	User replies to a post	user_id, reply_id, post_id
reply_deleted	Reply is deleted	user_id, reply_id, post_id
reaction_added	Reaction added	user_id, emoji, post_id, reply_id
reaction_removed	Reaction removed	user_id, emoji, post_id, reply_id
report_created	User submits a report	user_id, report_id
report_resolved	Moderator resolves a report	user_id, report_id, status
user_registered	New account created	user_id
user_login	User logs in	user_id

6.2 `digest_sections`

```
"digest_sections": [
  {
    "key": "my_section",
    "label": "My Section",
    "icon": "fa-chart-bar",
    "enabled_by_default": true
  }
]
```

Sections contributed to digest emails. Requires `handle_digest_section/3`.

6.3 `side_data`

```
"side_data": [
  {"entity": "post", "kind": "my_attachment"}
]
```

Composer attachment types. entity is `post`, `reply`, or `user`. Namespace `kind` with your slug. Requires `persist_attachment/3`.

6.4 `notification_types`

```
"notification_types": [
  {
    "key": "my_notif",
    "label": "My notification",
    "icon": "fa-bell",
    "channels": ["web", "email"],
    "default_preferences": {"web": true, "email": false},
    "payload_schema": {"post_id": "The triggering post"}
  }
]
```

6.5 `capabilities`

Forward-compatible declarations of privileged operations. Currently informational — unknown values warn but don't block install.

6.6 `permissions`

```
"permissions": [
  {"key": "can_view_gallery",   "default": "everyone"},
  {"key": "can_upload_image",   "default": "member"},
  {"key": "can_manage_gallery", "default": "moderator"}
]
```

Access gates enforced by `Nexus.Extensions.Permissions.check/3`. The four role tiers:

Tier	Who passes
everyone	Guests and members — only if site-wide guest browsing is enabled.
member	Any logged-in user.
moderator	Moderators and admins.
admin	Admins only.

Group-aware gates: Admins configure permission keys using a gate picker that combines a role tier with zero or more custom Groups. Access is granted when either the user's role meets the required tier or the user belongs to any of the configured groups. Your extension calls `Permissions.check/3` identically regardless — group evaluation happens inside the check function automatically.

7. The manifest — frontend surfaces

7.1 `slots`

```
"slots": ["post_footer", "profile_sidebar"]
```

Slot	Where it renders	Props
post_footer	Below post body on /post/:id, above replies	{ post_id }
profile_sidebar	Left rail of /profile/:username	{ username, current_user }
compose_attachments	Below post body in the composer, above footer. Post composer only.	{ attachments, set_attachments }

7.2 `routes`

```
"routes": [
  {"path": "/",           "title": "Home"},
  {"path": "/items/:slug","title": "Item detail"}
]
```

Path is relative to your extension's namespace — do not include `/ext/`. Bound to components via `registerRoute`.

7.3 `admin_panel`

```
"admin_panel": { "label": "My Extension", "icon": "fa-cog" }
```

Adds a page to the admin sidebar under Installed extensions. The registered component receives no props.

7.4 `explore`

```
"explore": { "label": "My Extension", "icon": "fa-puzzle-piece", "path": "/" }
```

Adds one entry to the Explore section of the left sidebar.

7.5 `right_widgets`

```
"right_widgets": [
  {"id": "my-widget", "label": "My Widget", "scope": "extension", "priority": 50}
]
```

Scope	Where the widget appears
extension	Your pages only (default)
global	Everywhere
{"path": "/x"}	A specific path
{"corePages": [...]}	feed, post, profile, members, leaderboard, badges, search, notifications, messages, saved, drafts

7.6 `toolbar_buttons`

```
"toolbar_buttons": [
  {
    "id": "my-button",
    "icon": "fa-solid fa-flask",
    "tip": "My button",
    "scope": "both",
    "priority": 50
  }
]
```

■ The `icon` field for toolbar buttons requires the FULL class with style prefix (`fa-solid fa-flask`). Every other surface uses the short form (`fa-flask`). Mixing them up renders as plain text.

`scope` is `both` (default), `posts`, or `replies`.

7.7 `profile_tabs`

```
"profile_tabs": [
  {"id": "my-tab", "label": "My Tab", "icon": "fa-star",
   "visibility": "always", "priority": 50}
]
```

■ `visibility: "own_only"` is a UX hint only — it hides the tab button but does NOT enforce access control. Your component must enforce access server-side.

8. The Elixir module

```
defmodule MyExtension do
  use Nexus.Extensions.Behaviour
end
```

`use Nexus.Extensions.Behaviour` provides no-op defaults for every callback. Override only callbacks for surfaces you declared in your manifest. Settings keys are always strings, never atoms: `settings["key"]` not `settings[:key]`.

8.1 `handle_event/3`

```
@impl true
def handle_event("post_created", %{"user_id" => uid, "post_id" => pid}, settings) do
  if settings["enable_debug_log"] do
    require Logger
    Logger.info("post_created uid=#{uid} pid=#{pid}")
  end
  :ok
end

# Catch-all required when hooks are declared
def handle_event(_event, _payload, _settings), do: :ok
```

Runs in a supervised Task — return value is ignored, crashes are caught and logged. Handlers for the same event run sequentially in priority order. Push heavy work to Oban jobs.

8.2 Lifecycle callbacks

```
@impl true
def on_install(settings) do
  # Runs once on first install, after migrations
  :ok
end

@impl true
def on_update(from_version, to_version) do
  # Runs on update, after new migrations
  :ok
end

@impl true
def on_uninstall do
  # Runs before migrations roll back
  :ok
end
```

■ These callbacks do NOT run on Nexus restarts — only on their discrete events (first install, update, uninstall).

8.3 `migrations/0`

```
@impl true
def migrations do
  [
    MyExtension.Migrations.V1CreateNotes,
    MyExtension.Migrations.V2AddIndex,
  ]
end
```

Use simple sequential names — V1, V2, V3. Nexus derives a collision-free version integer by hashing your extension's slug together with the sequence number, so two extensions that both have a V1 migration will never collide in `schema_migrations`. No date prefixes, no awareness of Nexus core's version range required.

Prefix table names with your slug to avoid naming conflicts at the database level: `my_extension_notes`, not `notes`. Migrations replay on every Nexus boot — already-applied versions are safely skipped by Ecto.

8.4 `routes/0` — API plug routes

Returns Elixir plug routes — distinct from manifest routes (which are SPA page paths). These handle API calls from your JavaScript bundle.

```
@impl true
def routes do
  [{"/", MyExtension.ApiRouter, []}]
end
```

■ The Nexus router intercepts all requests to `/ext/<slug>/api/*path` and strips the `/api/` prefix before your plug sees them. A request to `/ext/my-extension/api/status` arrives with `path_info: ["status"]` and `request_path: "/status"`. Your Plug.Router must define routes WITHOUT the `/api/` prefix.

```
defmodule MyExtension.ApiRouter do
  use Plug.Router
  plug :match
  plug :dispatch

  get "/status" do
    # called by: fetch(`/ext/${SLUG}/api/status`)
    send_resp(conn, 200, Jason.encode!(%{ok: true}))
  end

  match _ do
    send_resp(conn, 404, ~s({"error":"not found"}))
  end
end
```

`conn.assigns.current_user` is set if the request has a valid JWT — otherwise nil. Authentication is not enforced automatically; use `Nexus.Extensions.Permissions.check/3` for access control.

8.5 `child_specs/0`

```
@impl true
def child_specs do
  [{MyExtension.Scheduler, []}]
end
```

Started under `nexus_ext_sup_<slug>`. A crashing child only restarts within your supervisor. For Oban jobs use the `:extensions` queue and namespace workers under your root module.

8.6 `handle_digest_section/3`

```
@impl true
def handle_digest_section("my_section", period, settings) do
  %{
    title: "My section",
    layout: "list",
    items: [%{label: "Item", sublabel: "Detail", value: "42"}],
    cta: %{label: "See all", url: "/ext/my-extension"}
  }
end

def handle_digest_section(_key, _period, _settings), do: %{items: []}
```

Layouts: `list`, `leaderboard`, `stat_bars`, `pill_grid`, `card`. Empty items silently drops the section. For fully custom layout return `%{"_rendered_html" => html_string}`. A 4-arity form receives a branding map with `branding.accent`.

8.7 `persist_attachment/3`

```
@impl true
def persist_attachment("post", post_id, %{"kind" => "my_note", "data" => data}) do
  # persist to your own tables
  :ok
end

def persist_attachment(_entity, _entity_id, _attachment), do: :ok
```

Called after post/reply commit when the user queued an attachment via `attach()`. Best-effort — errors are logged and dropped. Subscribe to `post_deleted`/`reply_deleted` hooks to clean up linked rows.

8.8 `list_side_data/2`

An optional callback that lets Nexus aggregate side-data across extensions for a given entity. Return a list of maps; each entry should include `"kind"` and `"data"` fields so clients can identify what type of attachment they're looking at.

```
@impl true
def list_side_data("post", post_id) do
  Repo.all(from g in MyExt.PostGame, where: g.post_id == ^post_id)
  |> Enum.map(&%{"kind" => "game_link", "data" => %{"game_id" => &1.game_id}})
end

def list_side_data(_entity, _entity_id), do: []
```

This callback is optional — extensions are not required to expose their side-data this way. Many will prefer to expose their own endpoints under `/ext/<slug>/` for richer responses.

8.9 Database, storage, permissions

```
# Reference Nexus tables by string name
from(u in "users", where: u.id == ^uid, select: u.username) |> Nexus.Repo.one()

# File storage
alias Nexus.Extensions.Storage
abs_path = Storage.path("my-extension", "exports/report.pdf")
url      = Storage.url("my-extension",  "exports/report.pdf")

# Permission check — evaluates role AND group membership automatically
case Nexus.Extensions.Permissions.check("my-extension", "can_view", conn.assigns[:current_u
ser]) do
  :ok    -> # proceed
  :error -> conn |> put_status(403) |> json(%{error: "Forbidden"})
end
```

8.10 Available Elixir packages

Your extension shares Nexus's dependency tree. No declaration needed: Ecto, Ecto.SQL, Phoenix.PubSub, Oban, Req, Jason, Image (libvips), Floki, Swoosh, Joken, Bcrypt, ExAws, ExAws.S3, plus the full Elixir standard library.

9. The JavaScript bundle

```
(function() {
  "use strict";
  const NE   = window.NexusExtensions;
  const SLUG = "my-extension";

  // define components and call NE.register*
})();
```

Wrap in an IIFE to avoid polluting global scope. Use `window.React` and `window.ReactDOM` — Nexus's instances. Don't ship your own copy of React.

9.1 `registerRoute`

```
NE.registerRoute(SLUG, "/",           HomePage,   { title: "Home" });
NE.registerRoute(SLUG, "/items/:slug",DetailPage, { title: "Detail" });
```

Path is relative to your namespace — do not include `/ext/`. Component receives URL params plus `currentUser`.

9.2 `registerSlot`

```
NE.registerSlot({ slug: SLUG, slot: "post_footer", component: PostFooter, priority: 50 });
```

Slot	Props
post_footer	{ post_id } — post UUID
profile_sidebar	{ username, current_user }
compose_attachments	{ attachments, set_attachments }

`compose_attachments` renders below the post body in the post composer (not the reply composer). `attachments` is the live array of all queued attachments for the in-flight post — filter to your own kinds. `set_attachments` lets you remove an attachment the user wants to discard before posting.

9.3 `registerAdminPanel`

```
NE.registerAdminPanel(SLUG, { label: "My Extension", icon: "fa-cog", component: MyPanel });
```

Component receives no props. Use `window.NexusExtensionTemplates.SimpleSettingsPanel` or `TabbedPanel`.

9.4 `registerExploreItem`

```
NE.registerExploreItem({ slug: SLUG, path: "/", label: "My Extension",
  icon: "fa-puzzle-piece", priority: 50 });
```

9.5 `registerRightWidget`

```
NE.registerRightWidget({ slug: SLUG, id: "my-widget", label: "Widget",
  component: MyWidget, scope: "extension" });
```

Component receives `{ currentUser, pageProps }`.

9.6 `registerToolbarButton`

```
NE.registerToolbarButton({
  slug: SLUG,
  id:   "my-button",
  icon: "fa-solid fa-flask",  // full class with style prefix required
  tip:  "My button",
  scope: "both",
  priority: 50,
  onClick({ attach, currentUser, context }) {
    attach({ kind: "my_note", data: { text: "hello" } });
  },
});
```

■ `icon` requires the full class with style prefix (`fa-solid fa-flask`). All other surfaces use short form (`fa-flask`).

9.7 `registerProfileTab`

```
NE.registerProfileTab({ slug: SLUG, id: "my-tab", component: MyTabContent });
```

Component receives `{ username, current_user }`. Tab label, icon, visibility, and priority come from the manifest.

■ `visibility: "own_only"` hides the tab button but does NOT enforce access. Your component must check server-side.

9.8 `registerNotificationType`

```
NE.registerNotificationType("my_notif", {
  icon:      "fa-flask",
  iconColor: "var(--ac)",
  renderBody(n) { return React.createElement("span", null, "..."); },
  onClick({ n }) { window.NexusExtensions.navigate(`/post/${n.data?.post_id}`); },
});
```

9.9 `registerUserAction`

Adds an action button to the user card popover (shown when clicking a username) and the mobile user menu. Use this for actions on other users — viewing their profile page, sending a message, etc.

```
NE.registerUserAction({
  id:    "my-ext-view-log",
  label: "View Log",
  icon:  "fa-list",
  onClick({ user, currentUser, navigate, closeCard }) {
    closeCard();
    window.NexusExtensions.navigate(`/ext/my-ext/users/${user.username}`);
  },
  authOnly: false,  // hide when viewer is not logged in (optional, default false)
  priority: 50,     // lower = rendered earlier (optional)
});
```

`onClick` receives `{ user, currentUser, navigate, closeCard }`. Call `closeCard()` to dismiss the popover before navigating. No manifest declaration required.

9.10 `registerAccountAction`

Adds an item to the account menu — the desktop topbar dropdown and mobile account sheet. Use this for actions scoped to the current user's own account. For actions on other users' profile cards use `registerUserAction` instead.

```
NE.registerAccountAction({
  id:    "my-ext-my-log",
  label: "My Log",
  icon:  "fa-list",
  onClick({ currentUser, navigate, close }) {
    close();
    window.NexusExtensions.navigate(`/ext/my-ext/users/${currentUser.username}`);
  },
  priority: 50,
});
```

`onClick` receives `{ currentUser, navigate, close }`. Call `close()` to dismiss the menu before navigating. No manifest declaration required.

9.11 `registerPostAction`

Adds an item to the post **...** dropdown menu.

```
NE.registerPostAction({
  id:    "my-ext-link",
  label: "Link Item",
  icon:  "fa-link",
  onClick({ post, currentUser, navigate, closeMenu }) {
    closeMenu();
    // open a modal, navigate, etc.
  },
  // Optional — return false to hide this item for a given post/user combination
  visible({ post, currentUser }) { return true; },
  priority: 50,
});
```

`onClick` receives `{ post, currentUser, navigate, closeMenu }`. The optional `visible` function receives the same object and returns a boolean — return `false` to hide the item for a specific post or user combination. No manifest declaration required.

9.12 `registerFollowingTab`

Adds a custom tab to the Following feed page. The built-in Posts tab is always first and cannot be removed. The tab bar only renders when at least one extension tab is registered — zero footprint on installs that don't use it.

```
NE.registerFollowingTab({
  key:       "my-ext-feed",      // unique string, no spaces
  label:     "My Feed",          // display label in the tab bar
  component: MyFollowingFeed,    // React component, receives { currentUser }
});
```

The component receives `{ currentUser }` and is fully responsible for its own data fetching and rendering. No manifest declaration required.

9.13 `registerModerationSection`

Adds extension content to both the forum-side moderation page and Admin → Moderation. Use for content that needs moderator review.

```
NE.registerModerationSection({
  slug:    SLUG,
  label:   "Gallery",
  logo_url: "/uploads/extensions/gallery/logo.png",
  approvals: {
    badge:     () => pendingApprovalCount,
    component: ApprovalsQueue,
  },
  reports: {
    badge:     () => pendingReportCount,
    component: ReportsQueue,
  },
});
```

At least one of `approvals` or `reports` is required. Component receives `{ currentUser, context }` where `context` is `"moderator"` or `"admin"`.

■ Actions your component triggers must enforce permissions server-side. Client-side mounting is a UI affordance, not access control.

9.14 Calling APIs

Your own extension API

```
const token = localStorage.getItem("nexus_token");
const r = await fetch(`/ext/${SLUG}/api/stats`, {
  headers: token ? { "authorization": `Bearer ${token}` } : {}
});
```

Do not use `window._nexusApi` for your own endpoints — it hardcodes the `/api/v1` prefix.

`window._nexusApi` — *Nexus core endpoints*

For Nexus's own `/api/v1/*` endpoints. Handles JWT pass-through and 401 token refresh automatically.

```
const data = await window._nexusApi.get("/notifications/unread");
await window._nexusApi.post("/notifications/extension", { ... });
await window._nexusApi.patch("/some/path", { ... });
await window._nexusApi.delete("/some/path");

// File upload
await window._nexusApi.upload("/uploads/ext/your-slug", file, { type: "extension_image" });
```

Sending notifications

```
await window._nexusApi.post("/notifications/extension", {
  slug:           SLUG,
  target_user_id: userId,
  type:           "my_notif",
  data:           { post_id: postId },
  post_id:        postId,
});
```

Uploading files

```
const { url, original_url, upload, error } =
  await window.NexusExtensions.uploadFile(file, {
    slug: SLUG,
    type: "extension_image",  // or "extension_file"
  });
if (error) throw new Error(error);
```

`extension_image` accepts JPEG/PNG/GIF/WebP, auto-converts to WebP. `extension_file` accepts video, audio, PDF, ZIP, plain text, CSV, Markdown, JSON, and Office Open XML. SVG and executables are never accepted.

9.15 Host-provided UI primitives

Primitive	Use for
window.React, window.ReactDOM	Nexus's React instances — use hooks directly
NexusComponents.Toggle	Boolean toggle switch — props: value, onChange, label, hint
NexusComponents.Select	Styled dropdown — props: value, onChange, options
NexusComponents.Av	Avatar component — props: user, size
NexusComponents.Md	Renders Nexus Markdown — props: text
NexusComponents.toast(msg, type?)	Fire-and-forget toast. Types: err, warn, or omit
window.NexusExtensions.navigate(url)	SPA navigation — use everywhere

CSS variables

Variable	Purpose
--ac	Accent color
--bg	Page background
--s1, --s2, --s3	Surface levels (cards, popovers, overlays)
--t1 – --t5	Text colors (highest to lowest contrast)
--b1, --b2, --b3	Border colors

Variable	Purpose
--green, --red, --amber	Semantic colors
--av-radius	Avatar border-radius

Reusable classes: `btn-primary` (accent button), `btn-ghost` (outlined button), `md-body` (Nexus Markdown styles + image lightbox).

10. Install, update, uninstall, and boot

10.1 Install

Admin enters a manifest URL in Admin → Extensions → Install from URL. Nexus:

- Fetches and validates the manifest against the schema.
- Derives the GitHub repo from the URL or manifest's repository field.
- Fetches the latest GitHub release.
- Creates a DB row with `load_status: "not_loaded"`.
- Downloads the tarball, compiles, runs migrations, copies assets, starts processes, registers surfaces.
- Calls `on_install/1`.

10.2 Load status reference

Status	Meaning
loaded	Running normally.
disabled	Admin disabled it. Module stays loaded; dispatch is filtered out.
not_loaded	DB row exists but loader hasn't run.
compile_failed	Source didn't compile, or assets/supervisor failed.
manifest_invalid	Manifest failed validation, or declared module doesn't match.
migration_failed	A migration raised during install or update.
download_failed	Tarball couldn't be downloaded. Usually transient.
install_failed	on_install/1 returned error or raised. Extension is otherwise loaded.
update_failed	on_update/2 returned error or raised. New version is loaded.
no_release	Repo has no published GitHub releases.
no_repo	Manifest URL didn't resolve to a user/repo pair.

10.3 Updating

Install Updates appears when a newer release exists. Nexus stops the old extension, downloads the new tarball, recompiles, runs new migrations, re-registers surfaces, calls `on_update/2`.

10.4 Uninstalling

Uninstall in the … overflow: calls `on_uninstall/0`, cancels Oban jobs, rolls back migrations, unloads module and supervisor, deletes file storage and tarball cache, removes DB row and layout config.

Force-uninstall skips `on_uninstall/0` and migration rollback (tables persist and must be dropped manually), but still cleans up supervisor, jobs, storage, cache, DB row, and layout config. Use when normal uninstall fails outright.

10.5 Boot behaviour

On every Nexus restart, all enabled extensions are reloaded. `on_install/1` and `on_update/2` do not run on boot.

Tarball cache: every successful download is cached. On subsequent boots Nexus uses the cache — no GitHub request needed. GitHub is only involved at install and update time.

Situation	What happens
Fresh install	GitHub download → cache → compile → migrate → register
Subsequent boots	Cache hit → extract → compile → migrate → register (no network)
After an update	GitHub download for new version → cache → compile → migrate → register

11. The admin panel

11.1 The extension page

- Identity strip — name, version, status pill, enable/disable toggle, … overflow menu (View repo, View homepage, Sync from GitHub, Run migrations, Uninstall).
- Status banner — visible when not loaded. Shows load status and specific error message.
- Registered admin panel — if the extension declared one.
- Settings form — auto-rendered whenever settings_schema has fields.
- Runtime registrations panel — under the Advanced collapsible. Shows manifest declarations vs live registrations side-by-side.

11.2 Avoiding double-render

■ If you use `SimpleSettingsPanel` inside your registered admin panel to render `settings_schema` fields, remove those keys from `settings_schema`. Otherwise the host's auto-rendered fallback form renders them a second time below your panel.

Appendix A — Manifest quick reference

Field	Req	Purpose
manifest_version	✓	Must be 2
name	✓	Display name
slug	✓	Machine ID, URL prefix — immutable after install
version	✓	Extension version (semver)
module	✓	Root Elixir module name
js_bundle		JS bundle filename
description		One-sentence summary
author		Author name or handle
homepage		Canonical URL
repository		GitHub repo URL
license		SPDX identifier
tags		1–4 category pills
logo_url		200×200 icon
banner_url		800×400 hero image
compatible_with		Informational semver range
settings_schema		Admin-configurable settings
settings_tabs		Settings tab grouping
hooks		Event subscriptions
digest_sections		Digest email sections
side_data		Composer attachment types
notification_types		Notification categories
capabilities		Privileged operation declarations
permissions		Access tier gates

Field	Req	Purpose
slots		UI injection slots
routes		SPA page routes
admin_panel		Admin sidebar page
explore		Left sidebar explore entry
right_widgets		Right sidebar panels
toolbar_buttons		Composer toolbar buttons
profile_tabs		Profile page tabs

Appendix B — Elixir callback quick reference

Callback	Arity	Required when
handle_event	3	Any hooks declared
on_install	1	Optional — runs on first install only
on_update	2	Optional — runs on update only
on_uninstall	0	Optional — runs before uninstall
migrations	0	Extension needs DB tables
routes	0	Any API endpoints needed
child_specs	0	Background processes needed
handle_digest_section	3	Any digest_sections declared
handle_digest_section	4	Digest sections needing branding colors
persist_attachment	3	Any side_data declared
list_side_data	2	Optional — expose side-data to Nexus aggregator

Appendix C — JavaScript register API quick reference

Function	Required when
NE.registerRoute(slug, path, component, opts)	Any routes declared
NE.registerSlot({slug, slot, component, priority})	Any slots declared

Function	Required when
NE.registerAdminPanel(slug, {label, icon, component})	admin_panel declared
NE.registerExploreItem({slug, path, label, icon})	explore declared
NE.registerRightWidget({slug, id, label, component, scope})	Any right_widgets declared
NE.registerToolbarButton({slug, id, icon, tip, scope, onClick})	Any toolbar_buttons declared
NE.registerProfileTab({slug, id, component})	Any profile_tabs declared
NE.registerNotificationType(type, opts)	Any notification_types declared
NE.registerUserAction({id, label, icon, onClick, authOnly, priority})	Actions on user card / mobile user menu
NE.registerAccountAction({id, label, icon, onClick, priority})	Items in current user's account menu
NE.registerPostAction({id, label, icon, onClick, visible, priority})	Items in post … dropdown
NE.registerFollowingTab({key, label, component})	Custom tabs on the Following feed page
NE.registerModerationSection({slug, label, ...})	Extension needs moderation UI
NE.uploadFile(file, {slug, type})	File upload from browser needed
NE.navigate(url)	SPA navigation from bundle