# Building a Protocol Pack for TFI Discover Engine v6

Oct 29, 2026 · @Steven

## What a Protocol Pack is

A Protocol Pack teaches Discover Engine v6 a new domain without changing a line of engine code. The engine collects evidence, splits it into passages, extracts claims and verifies each claim against its source. The pack tells it which fields to look for, how to word the extraction, which rules apply, and which connectors to switch on.

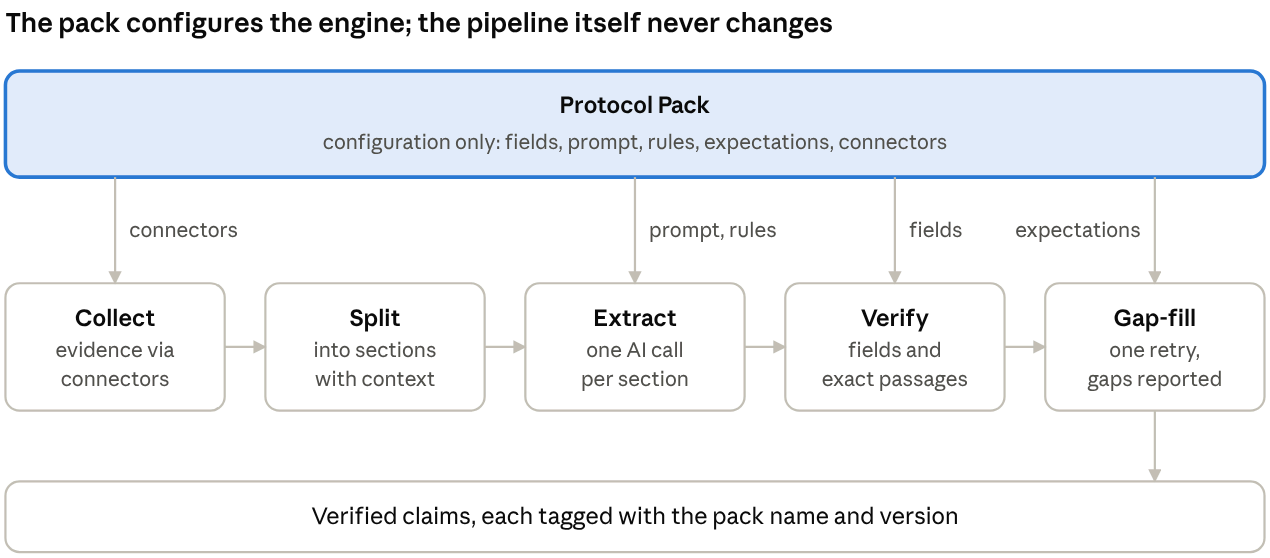
One pipeline already runs several packs unchanged: Business, Employment (job postings) and Patent, plus two placeholder packs used to prove the swap works.

This guide is for developers adding a new pack. It assumes you can read JavaScript and run the test suite with Deno.

**Note on names:** the code calls these *Domain Packs* (`domainPacks.js`, `domain_packs/`, `setActiveDomainPack`). Protocol Pack is the product name. They are the same thing.

## How a pack plugs into the engine

The engine runs the same five stages for every pack. The pack never adds code; it supplies the configuration each stage reads.




how a pack plugs into the v6 pipeline · 5 stages

Anything the AI returns outside the pack's fields is rejected with a reason, never silently kept. A claim counts only when its supporting passage is found word for word in the source, and every stored claim records which pack and pack version produced it.

## Anatomy of a pack

A pack is one JavaScript module in `base44/shared/domain_packs/` that default-exports a plain object. It holds configuration only, no logic. Every domain-specific rule lives in the pack, never in shared core code.

### Required fields

`validatePack()` in `domainPacks.js` checks these every time a pack loads. A pack that fails is refused with the full list of problems.

| Field | Type | Rule |
|---|---|---|
| `name` | string | The registry key, for example `"employment"` |
| `version` | string | Recorded on every claim the pack produces |
| `label` | string | The display name, for example `"Employment Research"` |

| Field | Type | Rule |
|---|---|---|
| `fields` | list | At least one field. Each needs `key`, `label` and `group`. Keys must be unique, and `"unmapped"` is reserved by the core. |
| `extractionPrompt` | string | Must contain the `{taxonomy}` and `{evidence}` placeholders |

## Optional fields the engine reads

| Field | What it does |
|---|---|
| `description` | One line shown in pack lists |
| `identityField` | The field key that names the subject, such as `position.title` or `patent.number` |
| `identityMatching` | `nameThreshold` (0 to 1) and `matchFields` for matching a subject to its sources |
| `confidenceScale` | `confirmed` and `likely` cutoffs for confidence tiers. Defaults are 0.8 and 0.5. |
| `labels` | Display names for fields and groups (`group_<name>`) |
| `extractionRules` | A list of domain rules appended to the prompt. Must be a list if present. |
| `expectations` | Structural checks run after extraction (see below). Each field named must be in the taxonomy. |
| `connectors` | Connector ids the pack switches on. Each must be registered. |
| `sourcePlugins` | Source plugins a scan may use while this pack is active |
| `sectionChars`, `contextFullLimit` | Override how long documents are split and how much context each section carries |
| `gapFillPass` | Set to `false` to skip the second, targeted extraction pass |

### The prompt and its placeholders

The engine fills four placeholders: `{taxonomy}`, `{subject_name}`, `{subject_descriptor}` and `{evidence}`. Evidence is filled last, so a document that happens to contain the text `{taxonomy}` is never treated as a placeholder.

You do not write the universal rules. The engine appends its CORE EXTRACTION RULES to every prompt (extract every item, quote passages word for word, never invent, send unfit statements to `unmapped`). Your `extractionRules` follow under a DOMAIN RULES heading.

### Expectations

An expectation says: every verified claim of field `when` must be accompanied by a verified claim of field `expect` with the same value of `matchOn`. The Patent pack uses one to make sure no independent claim is left without its elements:

```
expectations: [
  { when: "claim.independent", expect: "claim.element",
    matchOn: "location_ref", note: "each independent claim's elements must be
listed" }
]
```

An unmet expectation triggers one targeted retry for that section. If it is still unmet, it is reported as a gap rather than hidden.

## Choosing connectors

Connectors are the engine's plugs to the outside world, kept in one registry (`base44/shared/connectors/registry.js`). A pack lists the connector ids it needs; anything it does not list stays off while that pack is active.

A connector runs only when all four checks in `checkConnectorEligibility()` pass:

1. It is registered and its lifecycle is `implemented`.
2. An admin has not switched it off (the `connector_activation` system setting).
3. The active pack lists it in `connectors`.
4. Any key it needs (`needs_secret`) is set.

If a check fails, the engine returns the reason in plain words, for example `web.page: not one of the connectors pack "business" uses`.

The registry currently holds one connector:

| Id | What it does | Key needed | Known limits |
|---|---|---|---|
| `web.page` | Reads one public web page and returns its readable text, one block per line | None | Script-only pages return little text; follows no links; text over 80,000 characters is cut, and the cut is recorded |

The Employment and Patent packs both declare `connectors: ["web.page"]`. Admins can see every connector, its on/off state and which packs use it through the `getConnectorRegistry` function.

## Build, register and activate a pack

This walkthrough builds a small pack for grant funding announcements. The pack is an illustration for this guide, not one that ships with v6.

### 1. Write the pack module

Create `base44/shared/domain_packs/grant.js`. Build the field list once and reuse it for labels, as the Employment and Patent packs do.

```
// Grant Domain Pack v0.1. Pure configuration: no logic.
const fields = [
  { key: "grant.program",     group: "grant",       label: "Program name" },
  { key: "grant.agency",      group: "grant",       label: "Funding agency"
},
  { key: "grant.amount",      group: "grant",       label: "Award amount" },
  { key: "grant.deadline",    group: "grant",       label: "Deadline" },
  { key: "eligibility.rule",  group: "eligibility", label: "Eligibility rule"
}
].map((f, i) => ({ ...f, order: i + 1 }));

export default {
  name: "grant",
  version: "0.1",
  label: "Grant Research",
  description: "Funding announcements: program, agency, amount, deadlines and
eligibility.",
```

```
  identityField: "grant.program",
  fields,
  labels: {
    ...Object.fromEntries(fields.map(f => [f.key, f.label])),
    group_grant: "Grant",
    group_eligibility: "Eligibility"
  },
  extractionPrompt: `You are an evidence extraction engine for funding
announcements.
Extract structured claims from the document below.

SUBJECT: {subject_name}
SUBJECT DESCRIPTOR: {subject_descriptor}

FIELD TAXONOMY (use these field_key values exactly):
{taxonomy}

DOCUMENT:
{evidence}

Return a JSON object with a "findings" array.`,
  extractionRules: [
    "Each eligibility requirement is its own eligibility.rule claim.",
    "For grant.deadline, put which deadline it is in the conditions field,
for example letter of intent or full proposal."
  ],
  identityMatching: { nameThreshold: 0.85, matchFields: ["grant.program"] },
  confidenceScale: { confirmed: 0.8, likely: 0.5 },
  connectors: ["web.page"],
  sourcePlugins: []
};
```

## 2. Register it

Packs are registered by import, in one place. In `base44/shared/domainPacks.js`, import the module and add it to the `packs` map under its `name`:

```
import grantPack from "./domain_packs/grant.js";

const packs = {
  // ...existing packs
  grant: grantPack
};
```

No other core file changes. `getPack("grant")` now loads and validates it, and `listPacks()` reports it with any problems found.

### 3. Activate it

The active pack is a runtime setting, not code. An admin calls the `setActiveDomainPack` function with the pack name:

```
{ "name": "grant" }
```

The function refuses non-admins (403) and unknown names (400). It stores the choice in the `active_domain_pack` system setting, and every scan reads it from there. With no setting stored, the engine falls back to the default pack, which is defined in one place: `DEFAULT_PACK_NAME` in `domainPacks.js`.

## Testing a pack

The suite runs on Deno against an in-memory database and a scripted AI, so tests use no AI credits:

```
deno test --allow-read tests/
```

The existing pack tests in `tests/packs.test.js` show the four checks every new pack should get. Adapted for the grant pack:

```
import { assert, assertEquals } from "jsr:@std/assert@1";
import { getPack, validatePack } from "../base44/shared/domainPacks.js";
import { buildExtractionPrompt, validateFindings } from
"../base44/shared/extraction.js";

Deno.test("grant pack loads and passes the pack check", () => {
  assertEquals(validatePack(getPack("grant")), []);
});

Deno.test("grant rules reach the prompt, after the core rules", () => {
  const prompt = buildExtractionPrompt(getPack("grant"), "doc", {});
  assert(prompt.includes("CORE EXTRACTION RULES"));
  assert(prompt.includes("letter of intent"));
});

Deno.test("grant taxonomy rejects another domain's fields", () => {
  const { rejected } = validateFindings(
    [{ field_key: "claim.element", value: "x" }], getPack("grant"));
  assertEquals(rejected.length, 1);
});
```

For an end-to-end test, follow the cross-domain test in the same file. It feeds a real fixture document through `processEvidence()` with scripted findings and asserts that each claim is verified and that its stored offsets point back to the exact supporting passage in the source text.

It also confirms the engine is honest about coverage. When most of a document produced no claims, the run is marked `completed_with_gaps` instead of `completed`.

## Common mistakes and a final checklist

The pack check returns every problem at once, in plain words. These are the ones most likely to appear:

| Message | Cause | Fix |
| --- | --- | --- |
| `missing version` | `version` is absent or not a string | Use a string such as `"0.1"`, not a number |

| Message | Cause | Fix |
| --- | --- | --- |
| `extractionPrompt must contain {evidence}` | The prompt never inserts the document | Add the `{evidence}` placeholder (and `{taxonomy}`) |
| `field key "unmapped" is reserved by the core` | A field reuses the engine's catch-all key | Rename the field |
| `duplicate field key ...` | Two fields share a key | Make every key unique |
| `expectation field "..." is not in the taxonomy` | An expectation names a field the pack does not define | Fix the spelling or add the field |
| `connector "..." is not registered` | The pack lists a connector the registry does not have | Use a registered id, such as `web.page` |
| `Unknown domain pack "..."` | The pack was never added to the `packs` map | Register it in `domainPacks.js` |

Before you open a pull request:

- [ ] The module holds configuration only, with no functions or domain logic in core files
- [ ] `name`, `version` and `label` are strings, and `name` matches its key in the `packs` map
- [ ] Every field has a unique `key`, a `label` and a `group`
- [ ] The prompt contains `{taxonomy}` and `{evidence}`
- [ ] Domain rules are in `extractionRules`, worded as instructions the AI can follow
- [ ] Every connector listed is registered and actually needed
- [ ] The pack tests pass with `deno test --allow-read tests/`