

# JavaScript type hinting & validation (typescript)

---



- [Writing TypeScript](#)
  - [Strict checking](#)
  - [Type tests](#)
  - [Typing model fields added by plugins](#)
  - [Usage](#)
  - [Troubleshooting](#)
  - [Enabling for a theme or plugin](#)
  - [Bundled plugins and themes](#)
  - [Relative imports](#)
  - [Enable checking for a file](#)
  - [Limitations](#)
  - [Known Issues](#)
- 

Discourse ships type information for its JavaScript code. This can provide inline documentation, autocomplete, and other useful IDE features. It can also be used for some basic type validation, using the `@ts-check` directive.

Much of this will be automatically consumed by IDEs with TypeScript/JavaScript support. But for functionality in `.gjs` files, you'll need some specific configuration and/or IDE plugins.

## Writing TypeScript

Core, themes and plugins can be authored directly in TypeScript. Use a `.ts` extension for plain modules, or `.gts` for Glimmer components with a `<template>` tag. Type syntax is stripped at build time, so no separate compilation step is required. Linting (`@discourse/lint-configs`) and type-checking (`pnpm lint:types`) both understand these files.

# Strict checking

`pnpm lint:types` runs in TypeScript's `strict` mode, and TypeScript in core and in the bundled plugins and themes must pass it. Plugins and themes outside this repository extend the same config, so they are checked under `strict` too. To opt out, set `"strict": false` in the `compilerOptions` of the plugin's or theme's own `tsconfig.json`.

JavaScript files are not type-checked, but the published `@discourse/types` declarations for them are generated under `strict`. A field initialized to `null` is declared with the type `null`, so give public values like this a JSDoc type such as `/** @type {number | null} */`.

## Type tests

Types whose meaning a runtime test can't capture (a generic's resolved type, an overload pick, a return type derived from arguments) can be asserted at compile time with `expect-type`, in `.ts/.gts` files under `frontend/discourse/type-tests/` (e.g. `type-tests/truth-helpers/`). They are checked by `pnpm lint:types` but kept out of the test and production bundles.

## Typing model fields added by plugins

Each core model declares the fields core sends in an interface named after it, such as `TopicFields`, `PostFields`, `UserFields` or `AdminUserFields`. The model's type extends that interface. A plugin that adds fields, on the server with `add_to_serializer` or on the client with `api.addModelField` or `api.addTrackedTopicProperties`, can declare them by augmenting that interface.

The store has two registries that work the same way. `ModelRegistry` maps a store type name to its model, so `store.find` and `store.createRecord` return that model. `ModelCreateAttrsRegistry` maps a type name to the attributes `createRecord` accepts, for models whose inputs differ from their fields.

```
// plugins/my-plugin/assets/javascripts/discourse/model-types.ts
import type Sticker from "../models/sticker.js";

declare module "discourse/models/topic" {
  interface TopicFields {
    /** Sent only when the topic has stickers. */
    sticker_count?: number;
  }
}
```

```

    }
}

declare module "discourse/services/store" {
  interface ModelRegistry {
    sticker: Sticker;
  }
}

```

With this file in place, `topic.sticker_count` is a `number | undefined`, and `store.createRecord("sticker")` returns a `Sticker`.

- Type each field from what the server sends. Make it optional unless every payload that builds the model includes it, and add `| null` when the serializer can return `nil`. A field added with an `include_condition` is optional.
- Declare only the fields the plugin adds. Redefining a core field with a different type is reported as an error.
- A model the plugin defines follows the same pattern as core: export a `<Model>Fields` interface, merge it into the class with `export default interface <Model> extends <Model>Fields {}`, and register the class in `ModelRegistry`. Core disables the `@typescript-eslint/no-empty-object-type` and `@typescript-eslint/no-unsafe-declaration-merging` rules on the merge lines.
- The file must be a module, so keep at least one `import type` or an `export {}`. Without one, the `declare module` blocks replace the core modules instead of extending them. Avoid a bare `import "discourse/models/topic";` for this, because it stays in the bundle as a runtime import.
- Use a `.ts` file, not `.d.ts`. Type checking skips declaration files, so a field that clashes with the type core declares is only reported from a `.ts` file. The file compiles to an empty module.
- Registry keys and lookup names are matched after the resolver's model name normalization. For example, `postStream`, `post_stream` and `post-stream` match the same key. The exported `NormalizeModelName` type describes this normalization. Keys that normalize to the same name produce a union of their registered types. Names without a matching registration and dynamic strings fall back to `RestModel`. Only the model class lookup is normalized. The identity map and `__type` retain the raw string, and request URLs are still derived from that raw string through the adapter.

## Usage

- **CLI:** Run `pnpm lint:types`

- **VSCode:** Install the [TypeScript 7](#) and [Glint v2](#) extensions. Glint 1.4.0 and newer hands `.gts`/`.gjs` files to TypeScript's own language server. Both are part of our [recommended config](#), so you may already have it. If anything isn't working, you may need to trigger "Restart extension host" from VSCode's command palette, or restart the IDE.
- **JetBrains** (RubyMine, WebStorm, IntelliJ, etc.): Install the [EmberExperimental](#) plugin.

# Troubleshooting

Ensure that you've run `pnpm install` recently

## Enabling for a theme or plugin

Official themes/plugins, and the official skeletons, are all wired up for types. To enable it for your own plugin/theme, pull in the latest changes from the relevant skeleton (`package.json`, `tsconfig.json`)

## Bundled plugins and themes

Plugins and themes in this repository depend on `@discourse/types` through the pnpm workspace and reference the core project from their `tsconfig.json`. `pnpm lint:types` builds core first, so a core type change is checked against every bundled plugin in the same run, with no release or version bump in between.

In the editor, the TypeScript 7 language server resolves core imports straight to core source. Editors still on TypeScript 6 read the declarations that the last `pnpm lint:types` emitted, so run it, or keep `pnpm types:watch` running, after changing core types.

External plugins and themes consume the published `@discourse/types` package. It is published from `main` by the `publish-types` workflow.

## Relative imports

Relative imports name the file with its extension: `./foo.ts`, `./foo.js`, `./foo.gts`, `./foo.gjs`. TypeScript 7 only resolves a `.gts` or `.gjs` module when the specifier carries the extension, and the same rule is applied to every extension so that the type checker, the build and ESLint agree.

ESLint reports a missing extension and adds it with `--fix`.

# Enable checking for a file

`.ts` and `.gts` files are always type-checked. For `.js` / `.gjs` files, type-checking is opt-in: add `/** @ts-check */` at the top. For some examples, search Discourse core for `@ts-check`.

## Limitations

We do not provide any guarantees about the accuracy of the types - they're provided on a best-effort basis. PRs to improve the type documentation in core are welcome.

## Known Issues

- Autocomplete inside `<template>` tags requires complete syntax. For example, if you start typing:

```
<DBu
```

This will not autocomplete to `DBButton`, because the template syntax cannot be parsed. The workaround is to close the brackets, and then go back to the variable you'd like to autocomplete:

```
<DBu />
```

Upstream issue [here](#)

---

👤 #2

👤 2026-10-31 13:28:40 UTC

👤 2026-10-31 13:28:41 UTC