

Using Plugin Outlet Connectors from a Theme or Plugin



- [Choosing an outlet](#)
 - [Wrapper outlets](#)
 - [Defining the connector](#)
 - [Using outlet arguments](#)
 - [Adding more complex logic](#)
 - [Conditional rendering](#)
 - [Introducing new outlets](#)
-

Discourse includes hundreds of Plugin Outlets which can be used to inject new content or replace existing content in the Discourse UI. 'Outlet arguments' are made available so that content can be customized based on the context.

Choosing an outlet

To find the name of a plugin outlet, search Discourse core for "`<PluginOutlet`", or use the [plugin outlet locations](#) theme component. (e.g. `topic-above-posts`).

Wrapper outlets

Some outlets in core look like `<PluginOutlet @name="foo" />`. These allow you to inject new content. Other outlets will 'wrap' an existing core implementation like this

```
<PluginOutlet @name="foo">
  core implementation
</PluginOutlet>
```

Defining a connector for this kind of 'wrapper' outlet will replace the core implementation. Only one active theme/plugin can contribute a connector for a wrapper plugin outlet.

For wrapper plugin outlets, you can render the original core implementation using the `{{yield}}` keyword. This can be helpful if you only want to replace the core implementation under certain conditions, or if you would like to wrap it in something.

Defining the connector

Once you've chosen an outlet, decide on a name for your connector. This needs to be unique across all themes / plugins installed on a given community. e.g. `brand-official-topics`

In your theme / plugin, define a new `.gjs` connector with a path formatted like this:

```
“ :art: {theme}/javascripts/discourse/connectors/{outlet-name}/{connector-name}.gjs

:electric_plug: {plugin}/assets/javascripts/discourse/connectors/{outlet-name}/{connector-name}.gjs
```

The content of these files will be rendered as an Ember Component. For general information on Ember and the `.gjs` format, check out [the Ember guides](#).

For our hypothetical "brand official topics" connector, the file might look like

```
<template>
  <div class="alert alert-info">
    This topic was created by a member of the
    <a href="https://discourse.org/team">Discourse Team</a>
  </div>
</template>
```

[quote] [details=i Legacy wrapper elements] In the past, connectors were authored using `.hbs` files. When these files are used, the plugin outlet may automatically introduce a wrapper element. The element type is defined by `@connectorTagName` on the `<PluginOutlet />`.

Modern `.gjs`-based connectors have full control of their DOM. No automatic wrapper element will be introduced. [/details] [/quote]

Using outlet arguments

Plugin Outlets provide information about the surrounding context via `@outletArgs`. The arguments passed to each outlet vary. An easy way to view the arguments is to add this to your template:

```
{{log @outletArgs}}
```

This will log the arguments to your browser's developer console. They will appear as a `Proxy` object - to explore the list of arguments, expand the `[[Target]]` of the proxy.

In our `topic-above-posts` example, the rendered topic is available under `@outletArgs.model`. So we can add the username of the team member like this:

```
<template>
  <div class="alert alert-info">
    This topic was created by
    {{@outletArgs.model.details.created_by.username}}
    (a member of the
    <a href="https://discourse.org/team">Discourse Team</a>)
  </div>
</template>
```

[quote] [details=i Legacy ways to access arguments] In many Plugin Outlets, by default it is possible to access arguments using `{{argName}}` or `{{this.argName}}`. For now, this still works in existing outlets.

New plugin outlets (with `@defaultGlimmer={{true}}`) render connectors as 'template only glimmer components', which do not have a `this` context. Eventually, existing Plugin Outlets will also be migrated to this pattern. The `@outletArgs` technique is best because it will work consistently in both classic and glimmer plugin outlets. [/details] [/quote]

Adding more complex logic

Sometimes, a simple template is not enough. To add Javascript logic to your connector, upgrade your `.gjs` file to export a class-based component. This functions just the same as any other component definition, and can include service injections.

In our `topic-above-posts` example, we may want to render the user differently based on the 'prioritize username in ux' site setting. The `.gjs` file might look something like this:

```
.../connectors/topic-above-posts/brand-official-topic.gjs:
```

```
import Component from "@glimmer/component";
import { service } from "@ember/service";

export default class BrandOfficialTopics extends Component {
  @service siteSettings;

  get displayName() {
    const user = this.args.outletArgs.model.details.created_by;
    if (this.siteSettings.prioritize_username_in_ux) {
      return user.username;
    } else {
      return user.name;
    }
  }

  <template>
    <div class="alert alert-info">
      This topic was created by
      {{this.displayName}}
      (a member of the
      <a href="https://discourse.org/team">Discourse Team</a>)
    </div>
  </template>
}
```

[quote] [details=i Legacy ways to define complex logic] In older versions of Discourse, connectors were defined as a `.hbs` template plus an adjacent `.js` file, and it wasn't possible to export a custom component definition. Instead, you could export an object with `setupComponent(args, component)` and `teardownComponent(component)` functions. These older techniques are not officially deprecated yet, but we recommend switching to a single `.gjs` file with a class-based component going forwards. [/details] [/quote]

Conditional rendering

If you only want your content to be rendered under certain conditions, it's often enough to wrap your template with a handlebars `{{#if}}` block. If that's not enough, you may want to use the `shouldRender` hook to control whether your connector template is rendered at all.

Firstly, ensure you have a class-based `.gjs` connector as described above. Then, add a `static shouldRender()` function. Extending our example:

```
import Component from "@glimmer/component";

export default class BrandOfficialTopics extends Component {
  static shouldRender(outletArgs, helper) {
    const firstPost = outletArgs.model.postStream.posts[0];
    return firstPost.primary_group_name === "team";
  }
  // ... (any other logic)

  <template>
    {{! ... }}
  </template>
}
```

Now the connector will only be rendered when the first post of the topic was created by a team member.

`shouldRender` is evaluated in a Glimmer autotracking context. Future changes to any referenced properties (e.g. `outletArgs`) will cause the function to be re-evaluated.

[quote] [details=i Legacy shouldRender implementations] **Autotracking:** Before Discourse 3.1, `shouldRender` would only be evaluated during initial render. Changes to referenced properties would not cause the function to be re-evaluated.

Non-class syntax: For now, defining a `shouldRender` function in a plain (non-class) javascript object is still supported, but we recommend moving towards a class-based or templateOnly-based syntax going forward. [/details] [/quote]

Introducing new outlets

If you need an outlet that doesn't yet exist, please feel free to make a pull request, or open a topic in #dev.

👤 #2

👤 2026-10-31 13:28:09 UTC

👤 2026-10-31 13:28:10 UTC