

Converting modals from legacy controllers to new DModal component API



- [Step 1: Move Files](#)
- [Step 2: Update the JS file](#)
- [Step 3: Update the Template](#)
 - [Step 4: Update the showModal call sites](#)
- [Step 5: Tests](#)
- [Profit!](#)
- [Examples](#)

“:information_source: If you're implementing a new Modal, check out the main docs [here](#). This topic describes how to migrate an existing controller-based Modal to the new Component-based API.

In the past, Discourse used an Ember-Controller-based API for rendering modals. To invoke the modal, you would pass a string with the name of the controller to `showModal()`. Under the covers, this made use of Ember's `Route#renderTemplate` API, which is deprecated in Ember 3.x and will be removed in Ember 4.x.

To allow Discourse to upgrade to Ember 4.x and beyond, we've introduced a new component-based API for modals. This new API embraces Ember's 'declarative' design patterns, and aims to provide clean DDAU (data down actions up) semantics.

Step 1: Move Files

Move the controller JS file and the template file to the `/components/modal` directory. This makes them a 'colocated component' which can be imported just like any other JS module.

Step 2: Update the JS file

Then, update the component JS definition to extend from `@ember/component` instead of `@ember/controller` [^1]. Remove the `ModalFunctionality` mixin and update any uses of its functions according to the table below:

[^1]: Classic Ember Components are recommended in this guide because they provided the easiest migration path from Ember Controllers. But for simple modals, or if you're happy to spend some time refactoring, modern Glimmer components are the better choice.

Before	After
<code>flash()</code> and <code>clearFlash()</code>	Create a <code>flash</code> property in your component and pass it to the <code>@flash</code> argument of <code><DModal></code> . By default the alert will be styled with the <code>alert</code> class which is a copy of the <code>'error'</code> class, but it can be overridden using the <code>@flashType</code> argument.
<code>showModal()</code>	Import the <code>showModal</code> function from <code>discourse/lib/show-modal</code>
<code>closeModal</code> action	Invoke the <code>closeModal</code> argument which is automatically passed into your component

Old-style Modal Controllers would live 'forever', which meant we had to manually cleanup state. With the new Component-based API, the component will be created and destroyed when the modal is shown/hidden. In many cases that means your old lifecycle hooks are no longer required.

If you still need some lifecycle-based logic, use this table:

Before	After
<code>onShow()</code>	Use standard Ember component lifecycle (<code>init()</code> or Ember modifier)
<code>afterRender</code>	Use standard Ember component lifecycle (<code>init()</code> or Ember modifier)
<code>beforeClose()</code>	create a wrapper around the <code>@closeModal</code> argument which is passed into your component. Pass a reference to your close wrapper into <code>DModal</code> like <code><DModal @closeModal={{this.myCloseModalWrapper}}></code>
<code>onClose()</code>	Use standard Ember component lifecycle (<code>willDestroy()</code> or Ember modifier)

Step 3: Update the Template

Replace the `<DModalBody>` wrapper with `<DModal>`. Add some new attributes:

- Pass through the new `@closeModal` argument
- Add an explicit class. To match the old behavior, take your controller filename and add `-modal`.

For example, if your modal controller was called `close-topic.js`, the new `<DModal>` invocation would look something like this:

```
<DModal @closeModal={{@closeModal}} class="close-topic-modal">
```

If the `DModalBody` invocation includes any other arguments, update them based on the table below:

Before	After
<code>@title="title_key"</code>	<code>@title={{i18n "title_key"}}</code>
<code>@rawTitle="translated title"</code>	<code>@title="translated title"</code>
<code>@subtitle="subtitle_key"</code>	<code>@subtitle={{i18n "subtitle_key"}}</code>
<code>@rawSubtitle="translated subtitle"</code>	<code>@subtitle="translated subtitle"</code>
<code>@class</code>	<code>@bodyClass</code>
<code>@modalClass</code>	Use angle-bracket syntax with regular html attribute: <code><DModal class="blah"></code>
<code>@titleAriaElementId</code>	Use angle-bracket syntax with regular html attribute: <code><DModal aria-labelledby="blah"></code>
<code>@dismissable</code> , <code>@submitOnEnter</code> , <code>@headerClass</code>	Unchanged

If there was any footer content rendered after the old `<DModalBody>` component, use the new `<:footer>` named block to introduce it inside `<DModal>`. When using any named blocks, the body content should be wrapped in `<:body></:body>`. For example:

```
<DModal @closeModal={{@closeModal}}>
  <:body>
    Hello world, this is the content of the modal
  </:body>
  <:footer>
    This is the footer content. A `modal-footer` wrapper will be added
    automatically
  </:footer>
</DModal>
```

Step 4: Update the showModal call sites

Previously, modals would be rendered using the `showModal` API, which would take a string (the controller name) and a number of opts. It would return an instance of the controller which could be manipulated:

```
import showModal from "discourse/lib/show-modal";

export default class extends Component {
  showMyModal() {
    const controller = showModal("my-modal", {
      title: "My Modal Title",
      modalClass: "my-modal-class",
      model: { topic: this.topic },
    });

    controller.set("updateTopic", this.updateTopic);
  }
}
```

To render new component-based Modals you should inject the 'modal' service (or access it using something like `getOwner(this).lookup("service:modal")`) and call the `show()` function.

`show()` takes a reference to the new Component class as the first argument. The only opt still supported is 'model', which can be used to pass all data/actions required for your Modal.

No reference to the component instance will be returned. Instead, `show()` returns a promise which will resolve when the modal is closed. The promise will resolve with any data which was passed to `@closeModal`.

```
import MyModal from "discourse/components/my-modal";
import { service } from "@ember/service";

export default class extends Component {
  @service modal;

  showMyModal() {
    this.modal.show(MyModal, {
      model: { topic: this.topic, updateTopic: this.updateTopic },
    });
  }
}
```

```
}
```

Alternatively, migrate to the declarative API described in [the main DModal documentation](#).

The functionality of the old options can be replicated as follows:

Old <code>showModal</code> opt	Solution
<code>admin</code>	n/a for component - remove it
<code>templateName</code>	n/a for components - remove it
<code>title</code>	move to <code><DModal @title={{i18n "blah"}}></code>
<code>titleTranslated</code>	move to <code><DModal @title="blah"></code> . This could be computed based on data from <code>model</code> if needed
<code>modalClass</code>	move to <code><DModal class="blah"></code>
<code>titleAriaElementId</code>	move to <code><DModal aria-labelledby="blah"></code>
<code>panels</code>	Use the <code><:headerBelowTitle></code> named block to implement tabs in your component (example)
<code>model</code>	unchanged

Step 5: Tests

Any tests should largely remain the same. The most common issue are:

- Modals no longer have a default class based on their name. Classes must be specified explicitly in the template (see beginning of Step 3)
- The `d-modal` wrapper no longer persists in the DOM when the modal is closed. To check all modals are closed, use a check like `assert.dom('.d-modal').doesNotExist()`

Profit!

Your modal should now work as it did before. To take further advantage of the new API, you may want to consider [replacing `showModal` calls with a declarative strategy](#), and converting your Modal to be a Glimmer component.

Examples

Here are some example commits which demonstrate converting some of Discourse core's modals to the new API:

- <https://github.com/discourse/discourse/pull/22154>
- <https://github.com/discourse/discourse/pull/22164>

👤 #2

👤 2026-10-31 13:28:03 UTC

👤 2026-10-31 13:28:04 UTC