Embed Figma files

Use Figma's Embed Kit to integrate Figma files and prototypes into your apps and websites.

Overview

There are many places you may want to integrate live, up-to-date versions of Figma files and prototypes:

- Business and collaborative apps, such as letting users embed Figma files in documents.
- Internal tooling, such as extensions or workflows developed to support your organization's teams.
- Personal and professional websites like blogs or portfolios.

Figma provides comprehensive support for embedding files and prototypes. When you embed a Figma file, you can use attributes to easily enable or limit various aspects of the embed. For example:

- Control whether users can pan or zoom in the embed viewport.
- Choose whether users can select pages in the file.
- Specify whether a file opens in design or Dev Mode.
- Permit or restrict if a file or prototype can be expanded to full screen.
- Apply dark, light, or system themes.

With JavaScript, you can also augment embedded prototype behavior by posting messages and handling events emitted by the prototype.

To see Figma's embed features at work, check out a number of integrations where they're already used:

- Coda and Figma
- Notion and Figma
- Jira and Figma
- Dropbox and Figma
- Confluence and Figma
- P2 and Figma
- StoriesOnBoard and Figma
- Storybook and Figma
- Zeroheight and Figma

Embed Kit 2.0

In 2024, Figma released Embed Kit 2.0. Embed Kit 2.0 provides enhanced security features and gives you more control and flexibility over how you want to

bring live Figma designs, FigJam boards, and prototypes into your own websites or apps.

Enhanced security

To stay up to date with evolving web technology requirements, Embed Kit 2.0 introduces the use of the Storage Access API when third-party cookies are inaccessible due to browser changes like Intelligent Tracking Prevention and Privacy Sandbox. This ensures that embeds remain protected by user authentication while also benefiting from the latest security features defined by Web Standards, such as ensuring embeds exist in a secure context.

For embedded prototypes, Embed Kit 2.0 requires you to register your allowed embed origins with your OAuth app and provide the client ID. This ensures that embedded prototypes only accept messages from and emit events to domains that you control, as Figma will only send to and receive from the origins specified for your OAuth app.

More control and flexibility

Previously, embeds had some limited functionality around what kind of viewer experience you wanted to provide (for example, no control over theme). Embed Kit 2.0 offers substantial improvements:

- Send and receive events from a Figma prototype embed, so your site can control the prototype or react to a user's actions in the prototype.
- Allow users to select which page to view in an embedded Figma file.
- Hide all embed UI, except the Figma logo, to make embeds feel like a native part of your site or app.
- Lock the view of the embed to a particular page or a particular frame or slide.
- Set the embed to always open a file in Dev Mode or Figma Design.
- Use a light or dark theme, or match the user's system's theme.

Embed Kit 2.0 also introduces a new URL structure to make it easier to differentiate the types of embedded files:

```
embed.figma.com/board (FigJam boards)
embed.figma.com/slides (Figma Slides files)
embed.figma.com/deck (Figma Slides files in presentation view)
embed.figma.com/design (Figma files)
embed.figma.com/proto (Figma prototypes)
```

Am I using Embed Kit 2.0?

It's easiest to identify if you're using Embed Kit 2.0 or Embed Kit 1.0 by looking at the URL format.

Embed Kit 2.0 URL format: `embed.figma.com`

Embed Kit 1.0 URL format: `figma.com/embed`

If you see `embed.figma.com` in the URL of your embed, you're using Embed Kit 2.0.

Support for Embed Kit 1.0

Files embedded with Embed Kit 1.0 will be supported indefinitely. You don't need to update unless you want to take advantage of the new features or you use the Embed API to communicate with an embedded prototype.

> **Note:** While files will continue to work, support for using the Embed API with Embed Kit 1.0 has ended. To continue using the Embed API to control your embedded prototypes or get user-triggered events, you must migrate your embedded prototypes to Embed Kit 2.0, and follow the steps to get started with the Embed API.

Security and access

Embedded Figma files and prototypes respect the sharing settings of the file.

- If the file is password protected, the viewer is prompted for the password.
- If the file is shared within an organization, the viewer needs to log in to access the file.
- If the file isn't explicitly shared with the viewer, or the file is shared publicly, access to the file is based on the file's link sharing settings.

This ensures that the viewer has permission to access the file before the embed loads and renders.

Because embeds use iframes, embeds also benefit from the security features in browsers, such as enforcement of same-origin policy.

Embed a Figma file

To embed a Figma design or FigJam board, you create an iframe, provide the URL to the Figma file, and use query parameters to customize how the embed can be viewed and used. This section includes an example of an embedded file and the attributes of the iframe, how to format the URL, and the query parameters that are available.

Example file embed

File not found

Either this file doesn't exist or you don't have permission to view it. Ask the file owner to verify the link and/or update permissions.

You are logged in with these emails:

nirmankalia98@gmail.com

nirman.kalia@razorpay.com

Log in to another account

```
<iframe
  src="https://embed.figma.com/design/nrPSsILSYjesyc5UHjYYa4?
embed-host=figma-embed-docs"
  allowfullscreen>
</iframe>
```

The HTML in the example includes the `iframe` element, which is used to embed your Figma file. The `iframe` in the example has several attributes:

- The `src` attribute is used to specify the URL to the file you want to embed and includes required and optional query parameters
- The recommended `height` and `width` attributes
- The optional `allowfullscreen` attribute, which lets viewers expand the embed to fill their screen

Embeds support more attributes and parameters than those used in the example. For details and reference about setting up your embed, see the following sections:

- iframe format for embeds
- URL format for embeds
- Query parameters for files

iframe format for embeds

Figma files and prototypes are embedded using the `iframe` element. The following example is the simplest version of a functional embed:

```
<iframe src="https://embed.figma.com/{design|board|proto}/:file_key">
</iframe>
```

`src` is the only required attribute for the iframe. See URL format for how to build the URL for your `src` attribute.

You can also add standard attributes supported by iframes, such as `allowfullscreen` / `allow="fullscreen"`, `height`, `width`, and `style`. The order of the attributes doesn't matter. For example:

```
<iframe
  style="border: 1px solid rgba(0, 0, 0, 0.1);"
  width="800"
  height="450"
  src="https://embed.figma.com/{design|board|proto}/:file_key"
  allowfullscreen>
</iframe>
```

URL format for embeds

To get the base URL for the file or prototype that you want to embed, take the current URL for the file or prototype and replace `www` with `embed`. For example:

File URL	Embed URL
`https://www.figma.com/design/BAZsTPbh6W1r66Bdo/Example-design-file`	`https://embed.figma.com/design/BAZsTPbh6W1r66Bdo/Example-design-file`
`https://www.figma.com/proto/AZBcScMePxSpiX9j/Prototype-examples-for-embed?node-id=2654-15`	`https://embed.figma.com/proto/AZBcScMePxSpiX9j/Prototype-examples-for-embed?node-id=2654-15`

When you have the base URL, you can then apply the required and optional query parameters for your embedded file or prototype.

Embeds use the following URL format for the `src` attribute of your `iframe`:

```
https://embed.figma.com/{design|board|proto}/:file_key
```

Where:

- `design`, `proto`, `board`, `slides`, or `deck` corresponds to the type of file being embedded. For example, to embed a Figma design, you use `https://embed.figma.com/design/:file_key`.
- `:file_key` is the file key of the file or prototype that you want to embed. The file key is obtained from the URL of the Figma file. For example, in

`https://www.figma.com/board/BAZsTPbh6W1r66Bdo/Example-FigJam-board`, the file key is `BAZsTPbh6W1r66Bdo`.

- The file name (the hyphenated string that usually follows the file key in a Figma URL) can be included in the URL. If the file name changes, your embed is unaffected.

If you want to programmatically validate Figma URLs and convert them to embeds, see Determine if a URL is a Figma URL.

Query parameters for files

The following table lists the required and optional query parameters that can be used to customize the experience of an embedded Figma file.

Query parameters are added at the end of a URL. The first query parameter is preceded by a `?`, and following query parameters are preceded by `&`. For example, this URL has two query parameters:
`https://embed.figma.com/design/BAZsTPbh6W1r66Bdo?embed-host=example-product&node-id=0-3`

parameter in your embed URL.

Parameter	Value	Description	Example
`embed-host` (required)		A unique string that you use to identify the host where you're embedding the file, such as the name of your site or product.	`embed-host=example-product`
`mode` / `m`	Valid values: • `dev`	When set to `dev`, the embedded file is opened in Dev Mode.	`mode=dev`
`node-id`		The id for the node that you want the embed to display, such as a page or frame.	`node-id=0-3`
`footer`	Default value: `true`	Whether the Figma footer is displayed at	`foote`

Parameter	Value	Description	Example
	Valid values: • `true` • `false`	the bottom of the embedded file's viewport.	r=false
`viewport-controls`	Default value: `true` Valid values: • `true` • `false`	When `true`, the user can pan and zoom in the embed. For embedded Slides files, when `true`, the user can navigate to a different slide.	viewport-controls=false
`page-selector`	Default value: `true` Valid values: • `true` • `false`	When `true`, the page selector is shown and the viewer can change pages.	page-selector=false
`theme`	Default value: `light` Valid values: • `light` • `dark` • `system`	Specifies whether to use light, dark, or the viewer's system theme for the embed interface.	theme=system

Embed a Figma prototype

Prototypes are embedded much the same way as files: you create an iframe, provide the embed URL to the Figma prototype, and use query parameters to customize how the embedded prototype can be viewed and used. This section includes an example of an embedded prototype and the query parameters that are available.

Example prototype embed

Prototype not found

Either this file doesn't exist or you don't have permission to view it. Ask the file owner to verify the link and/or update permissions.

You are logged in with these emails:

nirmankalia98@gmail.com

nirman.kalia@razorpay.com

Log in to another account

```
<iframe
  src="https://embed.figma.com/proto/nrPSsILSYjesyc5UHjYYa4?node-id=5-3&starting-point-node-id=5%3A3&embed-host=figma-embed-docs"
  allowfullscreen>
</iframe>
```

The HTML in the example includes the `iframe` element, which is used to embed your Figma prototype. The iframe in the example has several attributes:

- The `src` attribute is used to specify the URL to the prototype you want to embed and includes required and optional query parameters
- The recommended `height` and `width` attributes

Embeds support more attributes and parameters than those used in the example. The general iframe and URL formats are the same for embedded prototypes as for files. For details about setting up your embed, see the following sections:

- iframe format for embeds
- URL format for embeds
- Query parameters for prototypes

Query parameters for prototypes

The following table lists the required and optional query parameters that can be used to customize the experience of an embedded Figma prototype.

Query parameters are added at the end of a URL. The first query parameter is preceded by a `?`, and following query parameters are preceded by `&`. For example, this URL has two query parameters:
`https://embed.figma.com/proto/BAZsTPbh6W1r66Bdo?embed-host=example-product&client-id=6rtt685EC`

The default values for the query parameters are provided. If you want to use the default value of an optional query parameter, you do not need to include the parameter in your embed URL.

Parameter	Value	Description	Example
`embed-host` (required)		A unique string that you use to identify the host where you're embedding the prototype, such as the name of your site or product.	`embed-host=example-product`
`client-id` (required for Embed API)		The client id of the OAuth app you set up at figma.com/developers/apps. `client-id` is required to use the Embed API with your embedded prototype. If you don't want to use the Embed API to send messages to or get events from the prototype, you can exclude `client-id`.	`client-id=6rtt685EC`
`node-id`		The id of the node to focus on when the embedded prototype loads. `node-id` and `starting-point-node-id` are both used, your embedded prototype will load displaying the node for `node-id`. If the value for `node-id` and `starting-point-node-id` are	`node-id=0-3`

Parameter	Value	Description	Example
		different, when the prototype is restarted, the flow will start from `starting-point-node-id`. If neither `node-id` or `starting-point-node-id` are used, the default prototype flow in the file is embedded.	
`starting-point-node-id`		The id of the node that starts the prototype flow that you want to embed. If `starting-point-node-id` is used, your embedded prototype will always restart from the node you specify. `node-id` and `starting-point-node-id` are both used, but have different values, the node for `node-id` will appear only when the embed loads. If the prototype is restarted, `starting-point-node-id` is used instead. If neither `node-id` or `starting-point-node-id` are used, the default prototype flow in the file is embedded.	starting-point-node-id=0-3
`footer`	Default value: `true` Valid values: • `true` • `false`	Whether the Figma footer is displayed at the bottom of the embedded file's viewport.	footer=false
`viewport-controls`	Default value: `true`	When `true`, the user can pan and zoom in	view

Parameter	Value	Description	Example
	Valid values: • `true` • `false`	the embed.	`port-controls=false`
`disable-default-keyboard-nav`	Default value: `false` Valid values: • `true` • `false`	When `true`, the default keyboard shortcuts for navigating prototypes are disabled.	`disable-default-keyboard-nav=true`
`show-proto-sidebar`	Default value: `false` Valid values: • `true` • `false`	Whether to show a left sidebar that lists all the prototype flows in the embed.	`show-proto-sidebar=true`
`hotspot-hints`	Default value: `true` Valid values: • `true` • `false`	When `true`, clickable areas of the prototype are highlighted when the user clicks in the prototype.	`hotspot-hints=false`
`device-frame`	Default value: `true` Valid values: • `true` • `false`	Whether to show the device frame for the prototype.	`device-frame=false`
`scaling`	Default value: `scale-down`	Determines the scaling mode for the prototype. These settings are equivalent to the	`scaling=contai`

Parameter	Value	Description	Example
	Valid values: • `scale-down` • `contain` • `min-zoom` • `scale-down-width` • `fit-width` • `free`	scaling options available when you play or share a prototype. Play your prototypes: Preview a prototype →	n
`content-scaling`	Default value: `fixed` Valid values: • `fixed` • `responsive`	Determines the way content is scaled in the prototype. These settings are equivalent to the content scaling options available when you play or share a prototype. Play your prototypes: Preview a prototype →	`content-scaling=responsive`

Embed API

When you embed a Figma prototype, you can use Figma's Embed API to trigger actions in the embedded prototype by sending messages, and handle events caused by user actions that are emitted by the prototype.

Get started with the Embed API

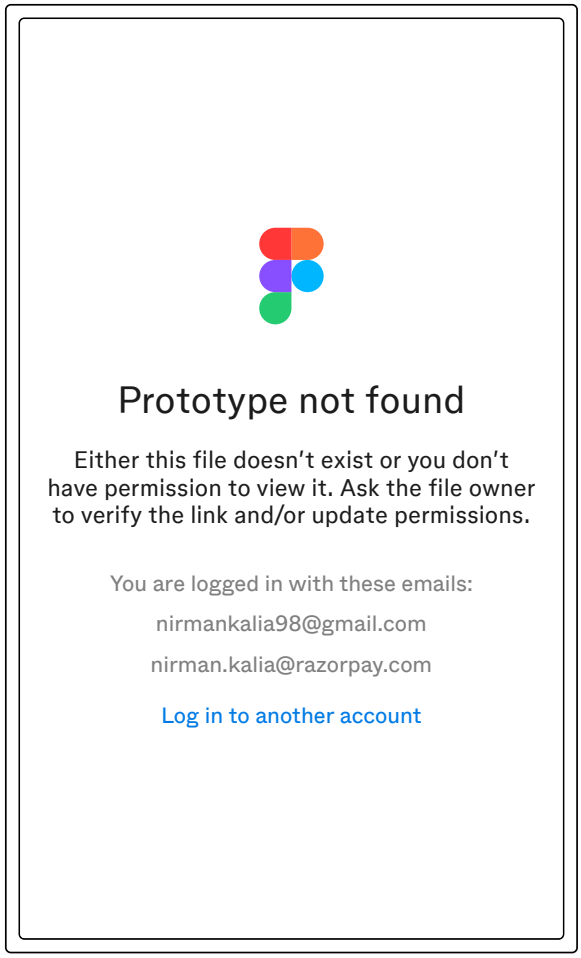
To use the Embed API, you must:

1. Create an OAuth app at figma.com/developers/apps. Provide a name, website, and logo for the OAuth app.
2. Copy the client id for your OAuth app. The client id is required to used the Embed API.
3. In the list of your OAuth apps, click the OAuth app that you created for your embeds.
4. In the modal for your app, click **Embed API**, and then **Add an embed origin**. The embedded prototype will only accept messages from and send events to the origins that you allow.

5. Add the `client-id` query parameter to the iframe URL for your embedded prototype. Set the value to your OAuth app's client id.
6. Implement code to send messages that control the prototype and handle events caused by user actions.

Example prototype using the Embed API

Suppose you want to provide buttons in your interface that a user can click to navigate to different frames or pages in the prototype. You also want to get events from the prototype so you can track how a prototype is used or have your interface react to the events.



Prototype controls:

← Restart →

Events:

Let's create a basic site that embeds a Figma prototype, has a few basic controls, and is able to send and receive messages so we can interact with the prototype and react to its state.

> **Tip:** For a version of the example site that includes CSS styling and the event viewer, see the figma/embed-kit-2.0-example GitHub repository.

Our site has two files:

- index.html, where we implement the embed and define the controls that a viewer can use with the embed
- script.js, where we handle the logic that sends our messages and reacts to events we get from the prototype

We also follow the Get started with the Embed API steps to obtain a client id and add an allowed embed origin for the domain where we're hosting our site.

Example: index.html

On our site, we want to embed a Figma prototype from one of our files. We also want to implement some basic controls: buttons that the viewer can use to navigate through the prototype and restart it.

Let's put together the HTML for our site.

```
<html>
 <head>
  <title>My Embedded Prototype</title>
  <script src="script.js" defer></script>
 </head>
 <body>
  <iframe
   id="embed-frame"
   height="600"
   width="800"
   src="https://embed.figma.com/proto/acDzTcMe9zjkq0b/Example-
embed-file
      ?node-id=5019-210
      &embed-host=example-domain
      &client-id=6rtt685EC"
   allowfullscreen
  ></iframe>
  <div id="controls">
   <div>Prototype controls:</div>
   <div id="buttons">
    <button id="prev" disabled>&larr;</button>
    <button id="restart" disabled>Restart</button>
    <button id="next" disabled>&rarr;</button>
```

```
    </div>
  </div>
 </body>
</html>
```

Let's take a quick look at the code.

In the `head`, we define the `title` for our site and link to our script.js file, where we'll handle the site logic. We `defer` the script because we'll be interacting with elements on the page and we want to be sure the DOM is ready before our script runs.

```
<head>
 <title>My Embedded Prototype</title>
 <script src="script.js" defer></script>
</head>
```

In the `body`, we first add the iframe for our prototype.

```
<iframe
 id="embed-frame"
 src="https://embed.figma.com/proto/acDzTcMe9zjkq0b/Example-
embed-file
    ?node-id=5019-210
    &embed-host=example-domain
    &client-id=6rtt685EC"
 allowfullscreen
></iframe>
```

The `src` for our `iframe` includes the `client-id` query parameter. The value for `client-id` is the client id that we obtained when we set up the OAuth app for the Embed API.

Then we add the `div` that holds the controls for our prototype. We'll start with the buttons `disabled` because we only want the controls to work after the embedded prototype has finished loading.

```
<div id="controls">
 <div>Prototype controls:</div>
 <div id="buttons">
  <button id="prev" disabled>Previous page</button>
  <button id="restart" disabled>Restart</button>
  <button id="next" disabled>Next page</button>
 </div>
</div>
```

We've assigned an `id` to each of the critical elements in our site so it'll be easier to reference the elements in our script and style them later on. With our site's structure in place, let's move on to the JavaScript we'll use for our site logic.

Example: script.js

The logic for our site needs to do a couple things:

- Send messages to the embed iframe, so we can navigate through and restart the prototype
- Handle events that are coming from the embed iframe, so we can have our site react to the state of the embedded prototype

Let's put together the JavaScript for our site.

```
// Constants for the site logic
const iframe = document.querySelector("#embed-frame");
const figmaOrigin = "https://www.figma.com";

// Messages to control the prototype
function nextPage() {
 iframe.contentWindow.postMessage(
  {
   type: "NAVIGATE_FORWARD"
  },
  figmaOrigin
 );
}

function previousPage() {
 iframe.contentWindow.postMessage(
  {
   type: "NAVIGATE_BACKWARD"
  },
  figmaOrigin
 );
}

function restartPrototype() {
 iframe.contentWindow.postMessage(
  {
   type: "RESTART"
  },
  figmaOrigin
 );
}

const restartButton = document.querySelector("#restart");
const nextButton = document.querySelector("#next");
const prevButton = document.querySelector("#prev");

restartButton.addEventListener("click", restartPrototype);
nextButton.addEventListener("click", nextPage);
prevButton.addEventListener("click", previousPage);

// Logic to handle events from the prototype
window.addEventListener("message", (event) => {
 if (event.origin === figmaOrigin) {
  if (event.data.type === "INITIAL_LOAD") {
   restartButton.removeAttribute("disabled");
   nextButton.removeAttribute("disabled");
  }
```

```
    if (event.data.type === "PRESENTED_NODE_CHANGED") {
      const nodeId = event.data.data.presentedNodeId;

      if (nodeId === "5019:210") {
        prevButton.setAttribute("disabled", "");
      } else if (prevButton.hasAttribute("disabled")) {
        prevButton.removeAttribute("disabled");
      }

      if (nodeId === "5019:72") {
        nextButton.setAttribute("disabled", "");
      } else if (nextButton.hasAttribute("disabled")) {
        nextButton.removeAttribute("disabled");
      }
    }
  } else {
    console.warn(
      "Received message from an unexpected origin:",
      event.origin
    );
  }
});
```

Let's break down the code in our script.js file.

First, we set two constants that we reuse throughout our script: `iframe` and `figmaOrigin`. `iframe` stores the reference to the iframe for our embedded prototype, so we can use it for sending messages. `figmaOrigin` contains the origin for messages from Figma, a value we need both for sending messages and handling events.

```
// Constants for the site logic
const iframe = document.querySelector("#embed-frame");
const figmaOrigin = "https://www.figma.com";
```

Next, we set up the functions that send messages to our embedded prototype.

```
// Messages to control the prototype
function nextPage() {
  iframe.contentWindow.postMessage(
    {
      type: "NAVIGATE_FORWARD"
    },
    figmaOrigin
  );
}

function previousPage() {
  iframe.contentWindow.postMessage(
    {
      type: "NAVIGATE_BACKWARD"
    },
    figmaOrigin
```

```
  );
}

function restartPrototype() {
  iframe.contentWindow.postMessage(
    {
      type: "RESTART"
    },
    figmaOrigin
  );
}
```

The `NAVIGATE_FORWARD`, `NAVIGATE_BACKWARD`, and `RESTART` messages all have a simple format. Some messages, such as `NAVIGATE_TO_FRAME_AND_CLOSE_OVERLAYS`, require a `data` property in addition to `type`.

Next, we select our `buttons` using their `ids`, and then assign event listeners to handle users clicking the buttons. The event listeners trigger the functions we defined earlier.

```
const restartButton = document.querySelector("#restart");
const nextButton = document.querySelector("#next");
const prevButton = document.querySelector("#prev");

restartButton.addEventListener("click", restartPrototype);
nextButton.addEventListener("click", nextPage);
prevButton.addEventListener("click", previousPage);
```

The events coming from our prototype, the events we want to handle, are sent to the `window`. We want to filter the messages so we're specifically handling messages from the Figma origin. To do that, we set up some initial event handling.

```
// Logic to handle events from the prototype
window.addEventListener("message", (event) => {
  if (event.origin === figmaOrigin) {

  } else {
    console.warn(
      "Received message from an unexpected origin:",
      event.origin
    );
  }
});
```

When our `window` gets a `message` event, we check if the event's origin corresponds to the Figma origin, where our prototype sends messages from. We include a simple warning in case we get messages from any unexpected origins.

Next, in our `if` statement, we start to add the logic for handling individual events. First, we want to enable the**Restart** and **Next page** buttons only after

the prototype has finished loading. To do that, we'll look for the `INITIAL_LOAD` event from our embedded prototype.

```
if (event.data.type === "INITIAL_LOAD") {
  restartButton.removeAttribute("disabled");
  nextButton.removeAttribute("disabled");
}
```

When we get the `INITIAL_LOAD` message, we remove the `disabled` attribute from the buttons.

We also want to add logic for enabling and disabling our**Previous frame** and**Next frame** buttons based on where the user is in the prototype. We want to make sure that if the user is on the first frame of the prototype that **Previous frame** is disabled (since the user can't navigate any further back), and similarly, that**Next frame** is disabled on the last frame of the prototype.

```
if (event.data.type === "PRESENTED_NODE_CHANGED") {
  const nodeId = event.data.data.presentedNodeId;

  if (nodeId === "5019:210") {
    prevButton.setAttribute("disabled", "");
  } else if (prevButton.hasAttribute("disabled")) {
    prevButton.removeAttribute("disabled");
  }

  if (nodeId === "5019:72") {
    nextButton.setAttribute("disabled", "");
  } else if (nextButton.hasAttribute("disabled")) {
    nextButton.removeAttribute("disabled");
  }
}
```

When a user navigates to a different frame in a prototype, the `PRESENTED_NODE_CHANGED` event is triggered. To know what frame the user is on, we handle `PRESENTED_NODE_CHANGED` and get the `presentedNodeId` (the new node the user is looking at). For this example, assume that `5019:210` is the node id of the first frame in our prototype flow, and that `5019:72` is the last frame.

And that's it! We've built our basic site. We followed Get started with the Embed API, set up the HTML for our site, implemented the logic to handle messages and events, and our site is ready to be used at the domain we added as an allowed embed origin.

Control a prototype with messages

The Embed API uses messages to handle cross-origin communication with the iframe. When the embedded prototype receives a supported message, it triggers the corresponding behavior.

For example, the following function sends a message that restarts the prototype from its starting node:

```
function restartPrototype(iframe) {
 iframe.contentWindow.postMessage(
  {
   type: "RESTART"
  },
   "https://www.figma.com"
 );
}
```

There `type` key is required for each message you send to an embedded prototype. Depending on the `type`, you may need to include additional keys in the message object.

The following message `types` are supported:

- `NAVIGATE_TO_FRAME_AND_CLOSE_OVERLAYS`
- `NAVIGATE_FORWARD`
- `NAVIGATE_BACKWARD`
- `CHANGE_COMPONENT_STATE`
- `RESTART`

The second argument for the `postMessage` method (the target origin) must always be `https://www.figma.com`.

The following sections describe the requirements for each `type` of message.

NAVIGATE_TO_FRAME_AND_CLOSE_OVERLAYS

This message closes any overlays that are open in the prototype and then navigates to the given node id.

```
{
 type: "NAVIGATE_TO_FRAME_AND_CLOSE_OVERLAYS",
 data: {
  nodeId: nodeId
 },
}
```

Where:

- `data.nodeId` is the node id that you want the prototype to navigate to. For example, `1:13`.

NAVIGATE_FORWARD

When the prototype has a sequence of frames, the prototype navigates to the next frame in the order.

```
{
 type: "NAVIGATE_FORWARD"
```

```
}
```

NAVIGATE_BACKWARD

When the prototype has a sequence of frames, the prototype navigates to the previous frame in the order.

```
{
 type: "NAVIGATE_BACKWARD"
}
```

CHANGE_COMPONENT_STATE

The given component in the prototype is changed to the provided state.

```
{
 type: "CHANGE_COMPONENT_STATE",
 data: {
  nodeId: string
  newVariantId: string
 }
}
```

Where:

- `data.nodeId` is the node id of the component that you want to change the state of.
- `data.newVariantId` is the id of the variant (the state) that you want to change the component to.

RESTART

This message causes the prototype to restart from its starting node.

```
{
 type: "RESTART"
}
```

Get events caused by user actions

Embedded prototypes emit events based on different user actions. Similar to the messages you send to control a prototype, events emitted by the prototype are messages sent from the iframe. You can listen for the messages, and then react to the message based on its `type` and the `data` included.

The following code is an example event listener that captures events from `www.figma.com`, the common origin of messages from the iframe:

```
// Event listener to capture messages from the iframe
window.addEventListener("message", (event) => {
 // Ensure the message is coming from the expected iframe origin
```

```
  const expectedOrigin = "https://www.figma.com";
  if (event.origin === expectedOrigin) {
    console.log("Received message from prototype iframe:", event.data);
    // Handle the event data as needed
  } else {
    console.warn("Received message from an unexpected origin:",
event.origin);
  }
});
```

The following event message `types` can be emitted by prototype:

- `MOUSE_PRESS_OR_RELEASE`
- `PRESENTED_NODE_CHANGED`
- `INITIAL_LOAD`
- `NEW_STATE`
- `REQUEST_CLOSE`
- `LOGIN_SCREEN_SHOWN`
- `PASSWORD_SCREEN_SHOWN`

The `origin` of the events emitted by the prototype is always `https://www.figma.com`.

The following sections describe the structure of each `type` of event message.

MOUSE_PRESS_OR_RELEASE

The `MOUSE_PRESS_OR_RELEASE` event provides data about user-clicks (or click-like events). This can be useful for scenarios such as generating a heat map of the user's activity in the prototype.

```
interface Position {
  x: number
  y: number
}

interface ScrollOffset {
  x: number
  y: number
}

interface MousePressOrReleaseMessage {
  type: 'MOUSE_PRESS_OR_RELEASE'
  data: {
    // If there are overlays showing, this is the node id of the
    // topmost overlay on screen. If there are no overlays showing
    // it's the id of the screen we're showing.
    presentedNodeId: string

    // Whether or not the user clicked a hotspot.
    handled: boolean

    // When this event isn't handled, this is the topmost
    // layer under the cursor.
    targetNodeId: string
```

```
    // Position relative to the top left corner of the target node;
    // this is unaffected by whether the target node is a scrolling
    // frame and has been scrolled.
    targetNodeMousePosition: Position

    // The nested-most scrolling frame enclosing the targetNode, or
    // the targetNode if it scrolls.
    nearestScrollingFrameId: string

    // Position relative to the top left corner of the scrolling frame;
    // this is unaffected by whether the target node is a scrolling
    // frame and has been scrolled.
    nearestScrollingFrameMousePosition: Position

    // The scroll offset of the above scrolling frame. If the target
    // node is a scrolling frame, you can use this and targetNodeLocation
    // to find where the click happened in the layer content bounds.
    nearestScrollingFrameOffset: ScrollOffset
  }
}
```

PRESENTED_NODE_CHANGED

The `PRESENTED_NODE_CHANGED` event describes navigation from one frame to another, or one overlay to another.

```
interface PresentedNodeChangedMessage {
  type: 'PRESENTED_NODE_CHANGED',
  data: {
    // The ID of the top-most node on screen; either an overlay node id
    // or the id of the top level frame if there are no overlays showing.
    presentedNodeId: string

    // In Figma, users can create connections that go "back" to the
previous
    // frame. To do this, Figma maintains its own history stack of frames
that
    // the user has navigated through.
    //
    // We intentionally exclude some transitions from the history stack:
ones
    // triggered via hover, mouse in and mouse out because this best
matches
    // user expectations.
    isStoredInHistory: boolean

    // A map of (string) developer friendly node IDs of instances
    // → (string) symbol IDs that they inherit from. This is useful for
    // determining the current states of interactive components on the
screen.
    // This map only contains entries for instances in the current base
screen
    // as well as any overlays being displayed.
    stateMappings: {
      [developerFriendlyNodeId: string]: string
    }
  }
```

```
}
```

INITIAL_LOAD

The `INITIAL_LOAD` event is triggered when the embedded prototype finishes loading for the first time.

```
interface InitialLoadMessage {
  type: 'INITIAL_LOAD'
  data: {}
}
```

NEW_STATE

The `NEW_STATE` event is triggered when a component instance in the prototype changes to a new variant.

```
interface NewStateMessage {
  type: 'NEW_STATE'
  data: {
    // The developer-friendly node ID of the instance that changed state.
    nodeId: string

    // The state ID (symbol ID) of the instance before the swap.
    currentVariantId: string
    // The state ID (symbol ID) of the instance after the swap.
    newVariantId: string
    // Same as PresentedNodeChangedMessage
    isStoredInHistory: boolean
    // Whether or not this was caused by an "after delay" interaction.
    // This is important b/c people use these for looping animations etc.
    isTimedChange: boolean
  }
}
```

REQUEST_CLOSE

The `REQUEST_CLOSE` event is triggered when user hits Spacebar in the inline preview as a way to request closing the viewer.

```
interface RequestCloseMessage {
  type: 'REQUEST_CLOSE',
  data: {}
}
```

LOGIN_SCREEN_SHOWN

The `LOGIN_SCREEN_SHOWN` event is triggered when a user sees a Figma login screen. Users may see a login screen if they are trying to view an embed that is private and they're not logged into Figma.

```
interface LoginScreenShownMessage {
  data: 'LOGIN_SCREEN_SHOWN',
}
```

PASSWORD_SCREEN_SHOWN

The `PASSWORD_SCREEN_SHOWN` event is triggered when a user tries to view a password-protected embed and needs to enter a password.

```
interface PasswordScreenShownMessage {
  data: 'PASSWORD_SCREEN_SHOWN',
}
```

Resources

This section contains resources for the Figma Embed Kit.

Determine if a URL is a Figma URL

Embeds are only supported for Figma URLs that link to either a Figma design file, a FigJam board, or a prototype. If you're ingesting URLs programmatically, you may want to validate that the URLs are the correct format for Figma files and prototypes.

To recognize these Figma links and render them successfully in your tool, use the following regular expression ("regex"):

```
const URL_REGEX = /https:\/\/[\w\.-]+\.?figma.com\/([\w-]+)\/([0-9a-zA-Z]{22,128})(?:\/.*)?$/
```

Any URL that matches the preceding regex is guaranteed to be handled by our embed endpoint, including any of the following URL types:

- `https://www.figma.com/board/:file_key/:file_name` → FigJam files
- `https://www.figma.com/slides/:file_key/:file_name` → Figma Slides files
- `https://www.figma.com/deck/:file_key/:file_name` → Figma Slides files in presentation view
- `https://www.figma.com/design/:file_key/:file_name` → Figma Design files, opened in design mode
- `https://www.figma.com/design/:file_key/:file_name?m=dev` → Figma Design files, opened in Dev Mode
- `https://www.figma.com/:file_type/:file_key/:file_name` → in future, we might add new file types

To turn the URLs into embeds, ensure that the `www.figma.com` or `figma.com` portion of the URL becomes `embed.figma.com`, and then apply required and optional query parameters for the file or prototype.

Find node ids

When you're working with embedded files and prototypes, you may want to reference specific nodes by id, such as when using the `node-id` query parameter.

There are a few ways you can find the id for a node.

Embed API

If you're using the Embed API with an embedded prototype, you can get node ids from the following events:

- `MOUSE_PRESS_OR_RELEASE`
- `PRESENTED_NODE_CHANGED`
- `NEW_STATE`

Figma URLs

In Figma, when you select most layers, the node id is included in the URL of the Figma file with the `node-id` query parameter (same as embed URLs). This can be a quick way to find most node ids, but won't work for certain objects, such as component variants.

For example, assume you've selected the first frame of a prototype flow in a Figma design file. In your browser, the URL for your Figma file looks like this: `https://www.figma.com/design/BAZsTPbh6W1r66Bdo/Example-file-name?node-id=5-3`. The `node-id` query parameter is `5-3`, the id of the node you've selected in this example. For use with the Embed API, you convert the hyphen to a colon (`5-3` becomes `5:3`).

Developer console

Using the developer console in the Figma desktop app or the developer tools in your browser, you can get the ids of the nodes you currently have selected on the canvas in your file. In the console, use `figma.currentPage.selection`.

`figma.currentPage.selection` returns an array, even when only one node is selected. If you only have one node selected, use `figma.currentPage.selection[0].id` to get the id of that node.

To open the console in the Figma desktop app, use the shortcut ⌥⌘I, or right-click on the canvas and go to **Plugins > Development > Show/Hide console**.

Migrate Embed Kit 1.0 embeds

To take advantage of the new features that are available, migrate your embeds from Embed Kit 1.0 to Embed Kit 2.0.

To migrate your old embeds:

1. Get the URL of your Embed Kit 1.0 embedded file or prototype. The embed URL is found in the `src` attribute of your embed iframe.
2. Remove `https://www.figma.com/embed`. This URL isn't used for Embed Kit 2.0.
3. Get the value of the `url` query parameter in the URL. If the value is URL-encoded, also decode the value. The `url` value is the URL for the file or

prototype that you embedded.

For example:
`https%3A%2F%2Fwww.figma.com%2Fdesign%2FBAZsTPbh6W1r66Bdo` becomes `https://www.figma.com/design/BAZsTPbh6W1r66Bdo`

4. Change the subdomain from `www` to `embed`.

   For example:
   `https://www.figma.com/design/BAZsTPbh6W1r66Bdo` becomes `https://embed.figma.com/design/BAZsTPbh6W1r66Bdo`

5. Fix the query parameters for the URL:

   1. Copy the `embed_host` value to the `embed-host` parameter for the new URL. This query parameter is still required for Embed Kit 2.0.
   2. If it was originally included, discard the `embed_origin` parameter. This parameter is not used with Embed Kit 2.0.

      If you want to continue to control prototypes, follow the steps to get started with the Embed API, and then add the `client-id` parameter to the new embed URL.
   3. Copy any remaining query parameters to the new URL. With the exception of `embed_origin`, Embed Kit 2.0 supports all of the parameters provided by Embed Kit 1.0.

Example of migrating the URL for a file embed:

Embed Kit 1.0:
https://www.figma.com/embed?url=https%3A%2F%2Fwww.figma.com%2Fdesign%2FBAZsTPbh6W1r66Bdo&embed_host=example-product&node_id=0-3

Embed Kit 2.0:
https://embed.figma.com/design/BAZsTPbh6W1r66Bdo?embed-host=example&node-id=0-3

Example of migrating the URL for a prototype embed:

Embed Kit 1.0:
https://www.figma.com/embed?url=https%3A%2F%2Fwww.figma.com%2Fproto%2FBAZsTPbh6W1r66Bdo&embed_host=example-product&node_id=0-3&embed_origin=https://example.com

Embed Kit 2.0:
https://embed.figma.com/proto/BAZsTPbh6W1r66Bdo?embed-host=example&node-id=0-3&client-id=6rtt685EC

Example migration function

The following example JavaScript function migrates the URL for an embed from Embed Kit 1.0 to Embed Kit 2.0.

```
function migrateEmbedURL(embedV1URL) {
  // Parse the input URL
  const oldURL = new URL(embedV1URL);
  const params = new URLSearchParams(oldURL.search);

  // Get the value of the 'url' parameter, decode it, and discard it
  let embedURL = decodeURIComponent(params.get('url'));
  params.delete('url');

  // Convert the subdomain of embedURL from 'www' to 'embed'
  let embedURLObj = new URL(embedURL);
  if (embedURLObj.hostname.startsWith('www')) {
    embedURLObj.hostname = 'embed' +
embedURLObj.hostname.slice(3);
  }

  // Merge query parameters from embedURLObj into params
  const embedURLParams = new
URLSearchParams(embedURLObj.search);
  for (const [key, value] of embedURLParams.entries()) {
    params.append(key, value);
  }

  // Check for the 'embed_origin' parameter, warn if present, and
discard it
  if (params.has('embed_origin')) {
    console.warn('embed_origin parameter is present:',
params.get('embed_origin'));
    params.delete('embed_origin');
  }

  // Construct the new URL using the value of embedURL and remaining
query parameters
  const newParams = new URLSearchParams();
  for (const [key, value] of params.entries()) {
    const newKey = key.replace(/_/g, '-');
    newParams.append(newKey, value);
  }

  // Combine the new URL and the new parameters
  embedURLObj.search = newParams.toString();
  const newURL = embedURLObj.toString();

  return newURL;
}
```

Troubleshooting

On rare occasion, viewers may have issues viewing embeds. Generally, this is related to whether the viewer has previously interacted with Figma, and whether they've correctly permitted embedded content and access to the browser's Storage Access API.

Allow embedded content

If a viewer has blocked or denied access to embedded content on a page, your embedded file or prototype won't be able to load. To solve this issue, the viewer must allow embedded content. The exact steps depend on the browser you're using.

- For Chrome, see Chrome's guide to embedded content and changing site settings.

The Storage Access API is blocked

Figma embeds require the Storage Access API. If a viewer has blocked Figma's access to the Storage Access API, or the browser has automatically blocked access, then the viewer needs to fix their browser settings to permit access to the Storage Access API. The exact steps depend on the browser you're using.

- For Chrome, the steps are the same as allowing embedded content. See Chrome's guide to embedded content and changing site settings.
- For Firefox, see the site storage settings and ensure that Figma isn't blocked.