⚠️ As of July 2019, the most up-to-date documentation is available in the Storybook addon-docs README. Please refer there & file github issues if you have questions or suggestions.

Storybook Docs: Technical Preview

Welcome to the technical preview of Storybook Docs. We're excited to give you a taste of the future of Storybook.

This short guide covers:

🚨🚨This document tracked the latest release of Storybook Docs up until *5.2.0-beta.26*. Docs has changed in a few ways since then. Please refer to the official documentation in the addon-docs README. 🚨🚨

⚠️ Disclaimers

This is a beta release of SB Docs. We're making it available early to maximize community participation and feedback. There are known bugs in the prerelease, but we've labeled it *beta* because we think it's more than usable and we aim to avoid significant breaking changes between now and the 5.2 final release.

You should use this if:

- You want to contribute to Storybook Docs
- You want to give feedback on Storybook Docs to make sure it suits your needs
- You are curious about what's next and want to stay on the bleeding edge

You should NOT use this if:

- You are not willing to roll with the punches as we improve it throughout the beta

💡 Getting Started

Ok, you're ready for action! Adding docs to your project is easy:

1. Install it
2. Configure it

NOTE: All of the examples in this doc are written for `react` but Storybook Docs is compatible with all of the currently supported view layers, so unless otherwise noted you can replace `react` with `vue`/`angular`/`ember` etc. in all of the steps below.

1. Installation

This guide assumes you're adding SB Docs to an existing Storybook project. If you don't have an existing project, first create one:

```
cd my-react-project
npx -p @storybook/cli@next sb init
```

Now add docs:

```
yarn add -D @storybook/addon-docs@next
```

Now make sure that all your `@storybook/*` packages are upgraded to the same version (`5.2.0-beta.x`), by running `yarn upgrade-interactive --latest` or by hand editing your `package.json` file and running `yarn install`.

Note: eventually docs will be a standard CLI install so this will be easier

2. Configuration

Next, let's configure Docs for your project.

Docs uses Storybook presets to simplify configuration. Just add the following line to your `.storybook/presets.js` file:

```
module.exports = ["@storybook/addon-docs/react/preset"];
```

If you're using `creact-react-app`, the config is a little more involved:

```
module.exports = [
  {
    name: '@storybook/addon-docs/react/preset',
    options: {
      configureJSX: true,
    },
  },
];
```

And if you're using Typescript and get an error `"Variable '__MODULE_DEPENDENCIES__' implicitly has an 'any[]' type"`, for now you can configure the preset to not use `source-loader`. We're working on fix.

```
module.exports = [
  {
    name: '@storybook/addon-docs/react/preset',
    options: {
      sourceLoaderOptions: null,
    },
  },
];
```

Docs also uses a slightly different file story loading mechanism than previous versions of Storybook. In your `.storybook/config.js` file:

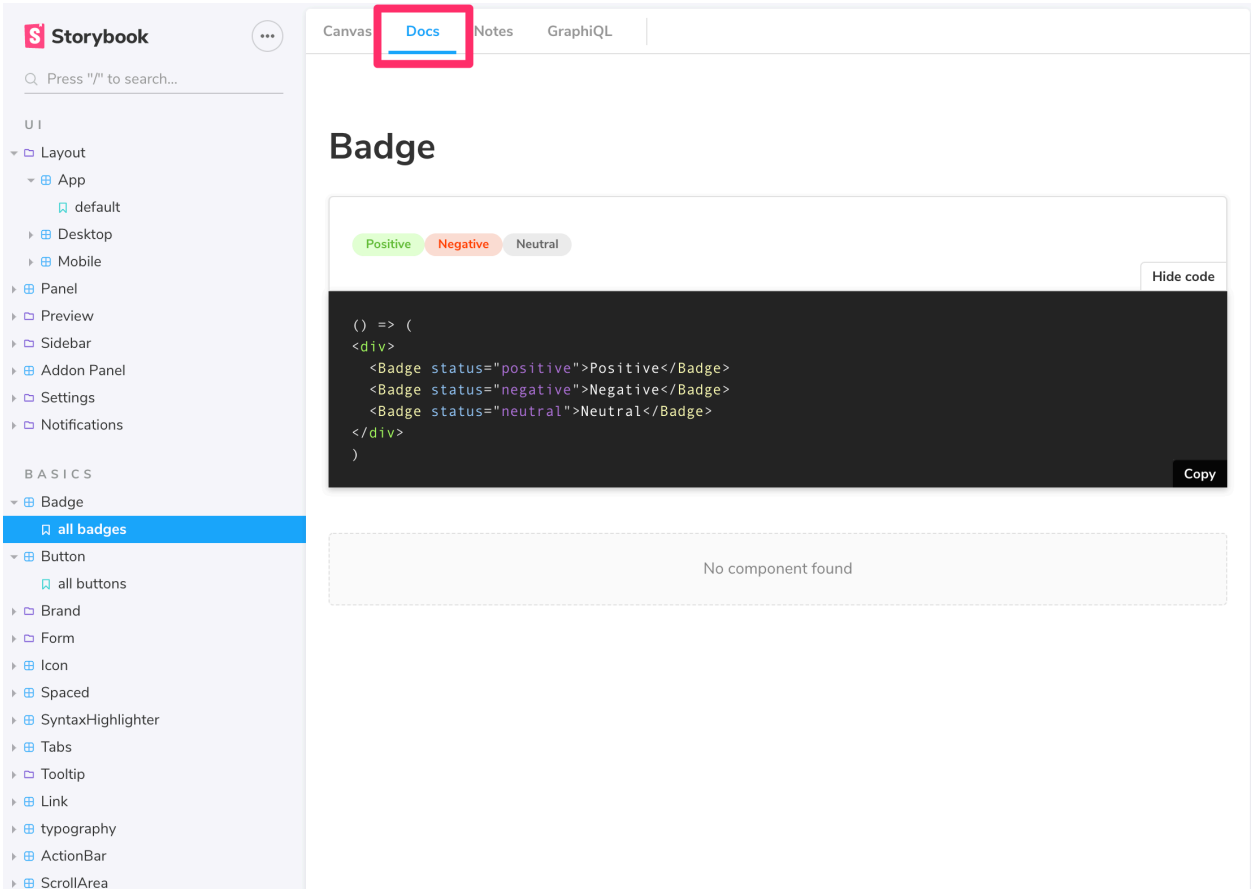
```
import { configure } from "@storybook/react";

// Contents of your existing configuration minus calls to the old-style `configure`

// wherever your story files are located
configure(require.context('../src', true, /\.stories\.(js|jsx|ts|tsx|mdx)$/),
module);
```

For manual setup instructions, see the Addon-docs README.

After you've configured Storybook, you should see a "Docs" tab in your UI, and when you click on the tab you should see an automatically generated documentation page “DocsPage” for your component.



This is a starting point, and from here you can either configure your stories and DocsPage to improve the output, or if you want to write flexible longform docs in MDX, SB Docs supports that as well.

🚀 Using docs

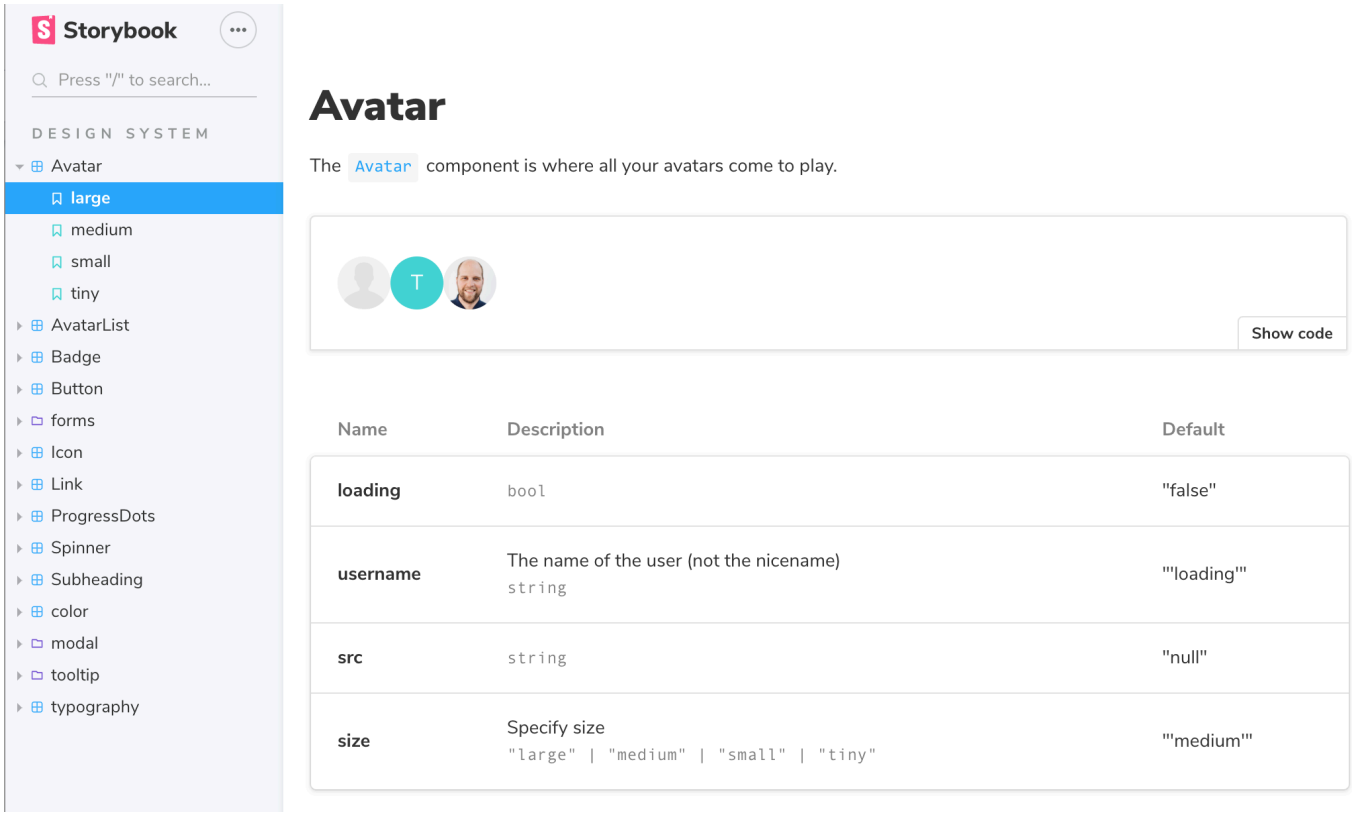
Now that you've installed SB Docs, there at least three different ways to use it based on your needs. You can:

- Improve the auto-generated DocsPage for existing stories
- Create rich documentation pages in MDX
- Embed an existing stories into MDX

1. Improve the auto-generated DocsPage for existing stories

When you set up SB Docs in the previous step, it automatically created a DocsPage for your existing stories. The easiest thing you can do to get great docs is to tweak those stories to improve the auto-generated documentation.

Here's what we're shooting for:



For example, to get the props table to show up you need to declare the “component” DocsPage should be extracting props from. Add something like the following to each of your stories:

```
storiesOf('Button', module)
  .addParameters({ component: Button })
  .add( .... )
```

The layout of the page currently includes:

1. Title / Subtitle
2. Component description
3. “Primary” story
4. Props table
5. Non-primary stories

The primary story is the first story of the list.

NOTE: aside from technical preview bugs this is currently “supported” for React, implemented for Vue, but not yet for any of the other view layers.

For all non-react view layers, you can use DocsPage for legacy stories and it will display Notes and Info props if they exist. This has been only minimally tested.

~~To display source for your existing stories, there’s currently a bit of setup required. Docs uses `source-loader` to parse the source. We’ll be streamlining this, but in the meantime install `@storybook/source-loader` as normal & then configure your `.storybook/webpack.config.js`:~~

```
const path = require('path');

module.exports = async ({ config }) => {
 config.module.rules.push({
   test: /\.stories\.[tj]sx?$/,
   loaders: [require.resolve('@storybook/source-loader')],
   include: [path.resolve(__dirname, '../src')],
   enforce: 'pre',
 });
 return config;
```

```
};
```

NOTE: As of `5.2.0-beta.8`, now `source-loader` is included as part of the addon-docs preset. If you're upgrading from an older version and you've already setup `source-loader` manually per the crossed out instructions above, you have two choices when you upgrade:

- EITHER remove the source-loader config from `webpack.config.js`
- OR disable `source-loader` in the preset by setting `sourceLoaderOptions` to `null` in the preset `options`.

NOTE: If you're using Typescript to define your component props, you also need to add `@storybook/preset-typescript` OR manually configure `react-docgen-typescript-loader` or another babel/webpack package to generate docgen for your components.

NOTE: In versions lower than alpha.38 you needed also need to specify `options: { injectParameters: true }` for source-loader. Now it's included by default.

DocsPage aims to replace **addon-info**. Right now it's roughly at parity with Info for React and Vue, and through the 5.2 release we will be refining and enhancing the functionality and building it out for Angular and potentially other view layers.

2. Create rich documentation pages in MDX

DocsPage gets you story-based component documentation quickly, but what if you want long-form documentation that integrates stories and rich text and documentation components? Storybook Docs also supports MDX for more control over your documentation.

Create a `Button.stories.mdx` in your project wherever you've configured Docs to get loaded in the previous step:

```
import { action } from '@storybook/addon-actions';
import { Button } from '@storybook/react/demo';
import { Story, Meta } from '@storybook/addon-docs/blocks';
```

```
<Meta title='MDX|Button' component={Button} />

# Hello Docs

Welcome to the future of Storybook!

<Story name="hello">
  <Button onClick={action('clicked')}>Hello button!</Button>
</Story>

<Story name="with emoji">
  <Button onClick={action('clicked')}>🤘🚀💯</Button>
</Story>
```

These stories are interchangeable with the stories you're used to in Storybook. The contents of the `<Story>` block corresponds to the return value of the `storiesOf.add(...)` function. So, the equivalent example for Vue might be:

```
<Story name="hello">
  {{
    template:
      '<my-button :handle-click="handleClick">Hello button!</my-button>',
    methods: { handleClick: action("clicked") }
  }}
</Story>
```

When you edit the MDX file, Storybook should pick up the changes just like when you edit JS stories.

3. Embedding existing storybooks in MDX

The last use case is a hybrid of the first two: suppose you have an existing Storybook, and you'd like to embed those stories inside rich MDX documentation so you have more control over the output. Well, we've got you covered.

Suppose you have an existing story and want to embed it into your docs. Here's how to show a story with ID `some--id` (check the browser URL in Storybook v5+ to see a story's ID):

```
import { Story } from "@storybook/addon-docs/blocks";
```

```
# Some header

And markdown here

<Story id="some--id" />
```

You can also use the rest of the MDX features in conjunction with this embedding. That includes source, preview, and prop tables.

4. Additional features

Docs has many more features in addition to the main use cases described above:

- Add existing stories to docs
- Add docs to existing stories
- Embed documentation components
- Story decorators and parameters
- Controlling iframe height
- Exporting documentation-only storybook
- More doc blocks.. coming soon

Add docs to existing stories, manually

What if you want to go the other way, i.e. you have an existing story and want to attach MDX docs to it? Just use the `docs` parameter. Note that if you don't use the `.stories.mdx` suffix, you can't use most doc blocks like `Story`, `Preview`, `Props`, or `Source`, you can just load MDX directly:

```
import React from 'react';
import { storiesOf } from '@storybook/react';
import MDX from './notes.mdx';

storiesOf('Classic|Welcome', module).add(
 'to Storybook',
 () => <Welcome />,
 { docs: { page: MDX } }
);
```

NOTE: both of these examples assume you're using the docs preset. If you're configuring manually YMMV.

Documentation components

You can embed arbitrary components in your documentation. For example, suppose you want to show an image carousel:

```
import Carousel from "nuka-carousel";

<Carousel>
  <img src="http://placehold.it/1000x400/ffffff/c0392b/&text=slide1" />
  <img src="http://placehold.it/1000x400/ffffff/c0392b/&text=slide2" />
  <img src="http://placehold.it/1000x400/ffffff/c0392b/&text=slide3" />
</Carousel>;
```

Story decorators and parameters

To add decorators and parameters in MDX:

```
<Meta
  title='MyComponent'
  decorators={[ ... ]}
  parameters={{ ... }}
/>
<Story name="story" decorators={[ ... ]} parameters={{ ... }} >
  ...
</Story>
```

In addition, global decorators work just like before, e.g. adding the following to your `.storybook/config.js`

```
import { addDecorator, addParameters } from '@storybook/react';
addDecorator(...);
addParameters({ ... });
```

Controlling iframe height

If you're not running your stories in inline mode, e.g. if you're using a view layer other than React, your embedded stories are displayed inside iframes within your docs. To control the height of the iframes use the **height** prop:

```
<Story name="foo" height="75px">...</Story>
<Story id="bar--baz" height="100px" />
```

Exporting documentation

The Storybook UI is a workshop for developing components in isolation. Storybook docs is a showcase for documenting your components. During component/docs development, it's useful to see both of these modes side by side. But when you export your static storybook, you might want to just export the docs to reduce clutter.

To address this, we've added a flag (WIP like everything else here) to export just the docs:

```
yarn build-storybook --docs
```

Doc Blocks

We'll be releasing a series of Doc Blocks to help you build out beautifully styled design system documentation. Check out the announcement and keep an eye on the Storybook twitter to be notified of their release.

💬 Feedback and Contributing

Problems setting up docs? Questions about using it? Missing features? Love it but think it could be better? Want to contribute? We welcome participation of all sorts!

Here are the best ways to get in touch:

- Check out Docs issues on github or file your own
- Join our discussion in the `#docs-mode` channel on the Storybook Discord

🤷 Frequently asked questions

What's missing from the technical preview?

Lots! The vision for Docs is outlined in the Docs sneak peek

We've still got a lot of work to do:

- More doc blocks like color swatches, typography, etc.
- Source and Props support for all story formats and view layers
- Freeform documentation that's not tied to a specific story
- Static site generation with `build-docs`
- Theming and documentation templates

There's probably more! For now we're just excited to get the basics into your hands as quickly as possible.

When will you release a stable version of Docs?

We don't have a release schedule yet. We're using feedback from this technical preview to figure one out. Check back here in a week and we should have a strawman.

Does Docs support framework X?

Docs does not currently support React Native. Aside from that, docs should currently support all frameworks that Storybook supports, including React, Vue, Angular, Ember, Svelte, Polymer, and others.

How does Docs interact with existing addons?

Currently we hide the addons panel when docs is visible. It's tricky because all the addons assume that there is only one story currently visible, and in docs there are potentially many. We have a proposal for "knobs v2" to address this for knobs, but nothing planned to address it in general. How we deal with it generally is open for discussion!

Does Docs coexist with my existing stories? What's a good migration strategy?

Assuming you're not doing anything too exotic, Docs should coexist seamlessly with all your existing stories.

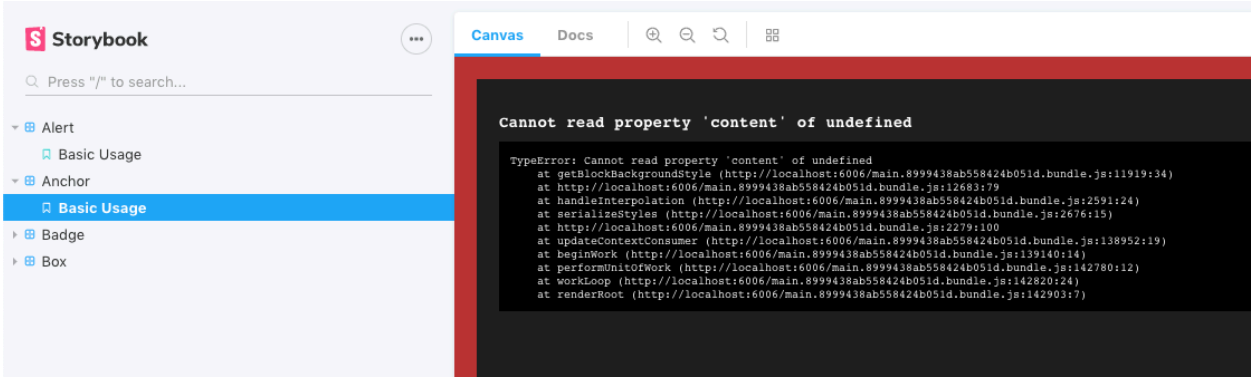
Stories defined in MDX will show up alongside your existing stories. And if you use the DocsPage component shown above, your existing stories will show up as well-formatted docs alongside your MDX.

Some storybook users have implemented their own documentation systems inside storybook stories, and have asked for help migrating to docs. If your stories contain documentation text etc and you want to show the existing stories "as is" in the docs tab, this little snippet will do the trick. Instead of setting the parameter `docs: { page: DocsPage }` in your config, do this instead:

```
import { DocsContainer, Story } from '@storybook/addon-docs/blocks';

addParameters({
 docs: {
  page: ({ context }) => <DocsContainer context={context} content={() => <Story id="." />}
/>
 }
});
```

How do I debug my MDX story?



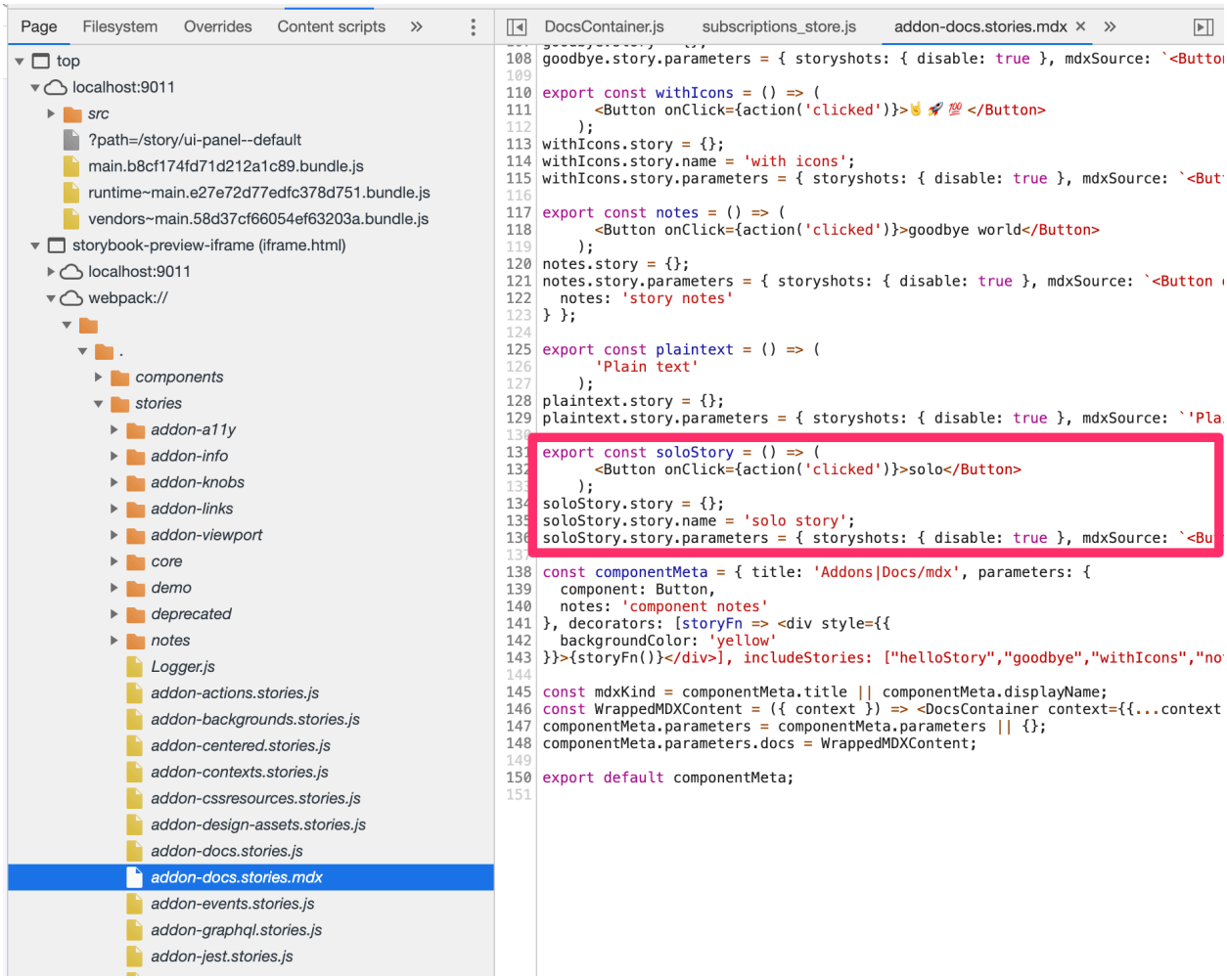
“My story renders in docs, but doesn’t show up the way I’d expect in the Canvas”

The original MDX gets compiled to Javascript, and the easiest way to debug your MDX stories in the Canvas is to inspect that Javascript. To do this, open your browser dev tools and view the source that’s being served by the webpack dev server. You may need to hunt for it a little bit under the “webpack > > . > path/to/your/stories” folder, but it’s there.

For example, the following MDX story:

```
<Story name="solo story">
  <Button onClick={action('clicked')}>solo</Button>
</Story>
```

Shows up in the dev tools as follows:




```
export const soloStory = () => (
  <Button onClick={action('clicked')}>solo</Button>
);
soloStory.story = {};
soloStory.story.name = 'solo story';
soloStory.story.parameters = { storyshots: { disable: true },
mdxSource: `<Button onClick={action('clicked')}>solo</Button>` };
```

This is useful to see because it's just the Module format which is the new standard for specifying stories. (It might look foreign right now, but it will replace the storiesOf format completely and will be well-documented for release).

Until then, this snippet is equivalent to the legacy:

```
storiesOf(...).add('solo story', () => <Button
onClick={action('clicked')}>solo</Button>, { storyshots ... })
```

Furthermore you can cut and paste this code into a new `.stories.js` file and play around with it at a lower level than MDX to fiddle with what's going wrong.