

Block Development Workshop



Josh Pollock

SATURDAY RIVE



About

Josh Pollock

- VP Engineering Experience: Saturday Drive
 - Ninja Forms, Caldera Forms, SendWP, Ninja Shop & more!
- Formerly Co-Owner & Lead Developer: CalderaWP
- WordPress Core Contributor
- Wrote a lot of tutorials about WordPress development



Workshop Overview

- Structuring a block plugin
- Basic Static Block
 - Block anatomy
- Editable Block
 - Block attributes
 - Using WordPress components
- Whatever y'all want to do next
 - Shortcode replacement, plugin sidebars, state management, testing, dynamic blocks, isomorphic blocks,

Cut & Paste From The README Like A Professional

github.com/shelob9/block-worskshop/

How do I get started? Where does everything go?

Hi Roy Block

The Anatomy of Using a Block

The image shows a screenshot of the WordPress Gutenberg editor interface. The main content area displays a heading "Say Hello to the New Editor" and a paragraph of text. Below the text is a row of four dark square blocks, each with a white icon and a short description. The third block, featuring a smartphone icon, is highlighted with a blue border. To the right of the main content is a sidebar with settings for the selected block, titled "Gallery". The sidebar includes sections for "Gallery Settings" and "Advanced".

Toolbar →

Inspector Controls →

Block Content →

Say Hello to the New Editor

It's a whole new way to use WordPress. Try it right here!

We call the new editor Gutenberg. The entire editing experience has been rebuilt for media rich pages and posts. Experience the flexibility that blocks will bring, whether you are building your first site, or write code for a

Do more with fewer plugins.

Create modern, multimedia-heavy layouts.

Work across all screen sizes and devices.

Trust that your editor looks like your website.

Gallery
Display multiple images in an elegantly organized tiled layout.

Gallery Settings

Columns
4

Crop Images
On

Thumbnails are cropped to align.

Link To
None

Advanced

The Anatomy of Coding a Block

registerBlockType()

registerBlockType()
Settings

```
21  registerBlockType('caldera-learn-basic-blocks/call-to-action', {  
22  
23      title: __('Call To Action'),  
24  
25  
26      category: 'widgets',  
27  
28      supports: {  
29          html: false,  
30      },  
31  
32  
33      edit: function (props) {  
34          return content();  
35      },  
36  
37      save: function () {  
38          return content();  
39      }  
40  });
```

```

131     edit({attributes, setAttributes, isSelected}) {
132         const {
133             text,
134             linkText,
135             link,
136             align,
137             textColor,
138             backgroundColor,
139             linkColor
140         } = attributes;
141
142         const onChangeText = (text) => setAttributes({text});
143         const onChangeLinkText = (linkText) => setAttributes({linkText});
144         const onChangeLink = (link) => setAttributes({link});
145         const onChangeAlign = (align) => setAttributes({align});
146         const onChangeTextColor = (textColor) => setAttributes({textColor});
147         const onChangeBackgroundColor = (backgroundColor) => setAttributes({backgroundColor});
148         const onChangeLinkColor = (linkColor) => setAttributes({linkColor});
149
150         if (isSelected) {
151             return (
152                 <div>
153                     <InspectorControls>
154                         <TextControl
155                             label={'Link Text'}
156                             value={linkText}
157                             onChange={onChangeLinkText}
158                         />
159                         <TextControl
160                             label={'Link Url'}
161                             value={link}
162                             onChange={onChangeLink}
163                         />
164                         <ColorControl
165                             label={'Text Color'}
166                             onChange={onChangeTextColor}
167                             value={textColor}
168                         />
169                         <ColorControl
170                             label={'Link Color'}
171                             onChange={onChangeLinkColor}
172                             value={linkColor}
173                         />
174                     </InspectorControls>
175                 </div>
176             );
177         }
178     }

```

It Can
Get Complex

```

135         align,
136         textColor,
137         backgroundColor,
138         linkColor
139     } = attributes;
140
141     const onChangeText = (text) => setAttributes({text});
142     const onChangeLinkText = (linkText) => setAttributes({linkText});
143     const onChangeLink = (link) => setAttributes({link});
144     const onChangeAlign = (align) => setAttributes({align});
145     const onChangeTextColor = (textColor) => setAttributes({textColor});
146     const onChangeBackgroundColor = (backgroundColor) => setAttributes({backgroundColor});
147     const onChangeLinkColor = (linkColor) => setAttributes({linkColor});
148
149     if (isSelected) {
150         return (
151             <div>
152                 <InspectorControls>
153                     <TextControl
154                         label={'Link Text'}
155                         value={linkText}
156                         onChange={onChangeLinkText}
157                     />
158                     <TextControl
159                         label={'Link Url'}
160                         value={link}
161                         onChange={onChangeLink}
162                     />
163                     <ColorControl
164                         label={'Text Color'}
165                         onChange={onChangeTextColor}
166                         value={textColor}
167                     />
168                     <ColorControl
169                         label={'Link Color'}
170                         onChange={onChangeLinkColor}
171                         value={linkColor}
172                     />
173                     <ColorControl
174                         label={'Background Color'}
175                         onChange={onChangeBackgroundColor}
176                         value={backgroundColor}
177                     />
178                 </InspectorControls>
179             </div>
180         );
181     }

```


Start Simple

**Write Code
Now**

Install Some Stuff

- Create package.json:
- Install WordPress scripts
 - `npm i @wordpress/scripts`
- Update "scripts" in package.json
 - [Copy from README](#)
- Ignore directories in .gitignore

Add Files & (Don't) Configure webpack & Babel

- Create src/index.js
- Put some JavaScript in there.
- Compile it to make sure everything works.
 - `npm run build`
 - You should see built JavaScript
- Be impressed by how simple webpack and Babel are to use :)
- The one PHP file!

Write

Block Now

Static, Non-editable Block

- Open `src/index.js`
- Write Some JavaScript:
 - Import dependencies
 - `registerBlockType()`
 - Edit Callback
 - Save Callback

Transpile Is A Cool Word!

- For manual testing while developing
 - `npm run start`
- To ship for production
 - `npm run build`

How do I make controls? Where Does Data Go?

Editable Block

The Toolbar and Inspector Controls

The screenshot displays the WordPress Gutenberg editor. At the top, a heading reads "Say Hello to the New Editor" with a subtext "It's a whole new way to use WordPress. Try it right here!". Below this, a paragraph states: "We call the new editor Gutenberg. The entire editing experience has been rebuilt for media rich pages and posts. Experience the flexibility that blocks will bring, whether you are building your first site, or write code for a".

Annotations highlight key features:

- Toolbar:** A red arrow points to the horizontal toolbar located below the introductory text, which contains icons for various block types and editing tools.
- Inspector Controls:** A red arrow points to the right-hand sidebar, which contains settings for the currently selected block (a Gallery).

The main editing area shows four dark-themed blocks with icons and text:

- Block 1: Plug icon, text "Do more with fewer plugins."
- Block 2: Grid icon, text "Create modern, multimedia-heavy layouts."
- Block 3: Mobile phone icon, text "Work across all screen sizes and devices." (This block is highlighted with a blue border and a close button in the top right corner).
- Block 4: Document icon, text "Trust that your editor looks like your website."

The right sidebar (Inspector Controls) shows the "Gallery" block settings:

- Gallery:** Display multiple images in an elegantly organized tiled layout.
- Gallery Settings:**
 - Columns:** A slider set to 4.
 - Crop Images:** A toggle switch turned on.
 - Thumbnails are cropped to align.**
 - Link To:** A dropdown menu set to "None".
- Advanced:** A section for additional settings, currently collapsed.

Inspector Controls

Put components inside
of InspectorControls
component

```
1 //...
2 registerBlockType( 'namespace/name', {
3   //...
4   edit({attributes, setAttributes, className, isSelected, clientId}) {
5     const {content} = attributes;
6     const setContent = (event) => {
7       setAttributes({content:event.target.value});
8     }
9     return (
10      <div>
11        <p>{content}</p>
12        <InspectorControls>
13          <TextControl
14            value={content}
15            onChange={setContent}
16            label={'Set content'}
17          />
18        </InspectorControls>
19      </div>
20    );
21  },
22 //...
```

Where Do I Put Data?

Attributes

Types of Attributes

- Strings
- Numbers
- Booleans
- Arrays

```
2 attributes: {  
3   title: {  
4     type: "string",  
5     default: "Hello Default Text!"  
6   },  
7   fontSize: {  
8     type: "number",  
9     default: 18  
10  },  
11  highContrast: {  
12    type: "boolean",  
13    default: false  
14  },  
15  images: {  
16    type: "array"  
17  }  
18 }  
19 };
```

Attribute Storage Methods

- **Default** - Serialized in HTML comments
- **Sourced** - Stored as part of the HTML content
- **Post Meta** - Stored in database as post meta

Default Attributes Serialized in Comments

Store in:

- HTML
- HTML comments
 - For shortcode-like blocks
- Post Meta *

Contact Page

```
<!-- wp:paragraph -->
<p>Thanks for wanting to communicate with me.</p>
<!-- /wp:paragraph -->

<!-- wp:image {"id":13,"align":"center"} -->
<div class="wp-block-image"><figure class="aligncenter"><figcaption>A Photo Of Me,
Josh Pollock</figcaption></figure></div>
<!-- /wp:image -->

<!-- wp:calderaforms/cform {"formId":"CF5ba40f978adc0"} /-->

<!-- wp:paragraph -->
<p></p>
<!-- /wp:paragraph -->
```

**Write More
Code Now
Please**

Make Block Editable

- Install new dependencies:
 - `npm i @wordpress/editor @wordpress/components`
- Import `TextControl` from `@wordpress/components`
 - Use for editing message
- Import `InspectorControls` from `@wordpress/editor`

Actually Make It Editable

- Add message attribute
- Get current value of message from attributes prop passed to save and edit.
- Create update function for message using setAttributes prop passed to edit callback.
- Display current value of message in edit and save callbacks.
- Use `TextControl` to create edit interface for block.
- Wrap the control in `InspectorControls` so it goes in the inspector controls.

What else do you want to know? How much time do we have left?

Cooler Blocks



Taking Gutenberg Further

- Shortcode to block conversion
- Plugin sidebars
- Block testing
- Isomorphic blocks
- React skills
- Redux Data Module
- Other frameworks besides React

Thank You!

Josh Pollock

- Josh412
- JoshPress.net
- SaturdayDrive.io

Slides And Links:

- github.com/Shelob9/block-worskshop/