Tutorial: Build your first block

- What you're going to build
- Prerequisites
- Scaffolding the block
- Reviewing the files
- Initial setup
  - Updating block.json
    - Modifying the block identity
    - Adding block supports
    - Removing unnecessary code
    - Putting it all together
  - Updating index.js
    - Adding a custom icon
  - Updating edit.js
  - Updating render.php
  - Cleaning up
- Adding block attributes
  - Updating block.json
  - Updating edit.js
    - Adding the user interface
      - Inspector controls
      - Components and panels
      - Text control
      - Toggle control
    - Updating the block content
  - Updating render.php
- Adding static rendering

- Why add static rendering?
- Adding a save function
- Updating save.js
- Handling dynamic content in statically rendered blocks
  - Adding a new attribute
  - Setting the attribute in edit.js
  - Optimizing render.php
- Wrapping up

In this tutorial, you will build a “Copyright Date Block”—a basic yet practical block that displays the copyright symbol (©), the current year, and an optional starting year. This type of content is commonly used in website footers.

The tutorial will guide you through the complete process, from scaffolding the block plugin using the `create-block` package to modifying each file. While previous WordPress development experience is beneficial, it’s not a prerequisite for this tutorial.

By the end of this guide, you will have a clear understanding of block development fundamentals and the necessary skills to create your own WordPress blocks.

What you’re going to build

Here’s a quick look at what you’re going to build.

What you're going to build

You can also interact with the finished project in WordPress Playground or use the Quick Start Guide to install the complete block plugin in your local WordPress environment.

Prerequisites

To complete this tutorial, you will need:

1. Code editor
2. Node.js development tools
3. Local WordPress environment

If you don’t have one or more of these items, the Block Development Environment documentation will help you get started. Come back here once you are all set up.

> This tutorial uses `wp-env` to create a local WordPress development environment. However, feel free to use any development environment that meets the abovementioned prerequisites.

Scaffolding the block

The first step in creating the Copyright Date Block is to scaffold the initial block structure using the `@wordpress/create-block` package.

> Review the Get started with create-block documentation for an introduction to using this package.

You can use `create-block` from just about any directory (folder) on your computer and then use `wp-env` to create a local WordPress development environment with your new block plugin installed and activated.

Therefore, choose a directory to place the block plugin or optionally create a new folder called “Block Tutorial”. Open your terminal and `cd` to this directory. Then run the following command.

> If you are not using `wp-env`, instead, navigate to the `plugins/` folder in your local WordPress installation using the terminal and run the following command.

```
npx @wordpress/create-block@latest copyright-date-block --variant=dynamic
cd copyright-date-block
```

After executing this command, you’ll find a new directory named `copyright-date-block` in the plugins folder. This directory contains all the initial files needed to start customizing your block.

This command also sets up the basic structure of your block, with `copyright-date-block` as its slug. This slug uniquely identifies your block within WordPress.

> You might have noticed that the command uses the `--variant=dynamic` flag. This tells `create-block` you want to scaffold a dynamically rendered block. Later in this tutorial, you

> will learn about dynamic and static rendering and add static rendering to this block.

Navigate to the Plugins page in the WordPress admin and confirm that the plugin is active. Then, create a new page or post and ensure you can insert the Copyright Date Block. It should look like this once inserted.

The scaffolded block in the Editor

Reviewing the files

Before we begin modifying the scaffolded block, it's important to review the plugin's file structure. Open the plugin folder in your code editor.

The files that make up the block plugin

Next, look at the File structure of a block documentation for a thorough overview of what each file does. Don't worry if this is overwhelming right now. You will learn how to use each file throughout this tutorial.

> Since you scaffolded a dynamic block, you will not see a `save.js` file. Later in the tutorial, you will add this file to the plugin to enable static rendering, so stay tuned.

Initial setup

Let's start by creating the simplest Copyright Date Block possible, which will be a dynamically rendered block that simply displays the copyright symbol (©) and the current year. We'll also add a few controls allowing the user to modify font size and text color.

Before proceeding to the following steps, run `npm run start` in the terminal from within the plugin directory. This command will watch each file in the `/src` folder for changes. The block's build files will be updated each time you save a file.

Check out the Working with JavaScript for the Block Editor documentation to learn more.

Updating block.json

Open the `block.json` file in the `/src` folder.

```
{
    "$schema": "https://schemas.wp.org/trunk/block.json",
    "apiVersion": 3,
    "name": "create-block/copyright-date-block",
    "version": "0.1.0",
    "title": "Copyright Date Block",
    "category": "widgets",
    "icon": "smiley",
    "description": "Example block scaffolded with Create Block tool.",
    "example": {},
    "supports": {
        "html": false
    },
    "textdomain": "copyright-date-block",
    "editorScript": "file:./index.js",
    "editorStyle": "file:./index.css",
    "style": "file:./style-index.css",
    "render": "file:./render.php",
    "viewScript": "file:./view.js"
}
```

> Review the block.json documentation for an introduction to this file.

Since this scaffolding process created this file, it requires some updating to suit the needs of the Copyright Date Block.

Modifying the block identity

Begin by removing the icon and adding a more appropriate description. You will add a custom icon later.

1. Remove the line for `icon`
2. Update the description to “Display your site’s copyright date.”
3. Save the file

After you refresh the Editor, you should now see that the block no longer has the smiley face icon, and its description has been updated.

The block in the Editor with updated information

Adding block supports

Next, let's add a few block supports so that the user can control the font size and text color of the block.

> You should always try to use native block supports before building custom functionality. This approach provides users with a consistent editing experience across blocks, and your block benefits from Core functionality with only a few lines of code.

Update the `supports` section of the `block.json` file to look like this.

```
"supports": {
    "color": {
        "background": false,
        "text": true
    },
    "html": false,
    "typography": {
        "fontSize": true
    }
},
```

Note that when you enable text color support with `"text": true`, the background color is also enabled by default. You are welcome to keep it enabled, but it's not required for this tutorial, so you can manually set `"background": false`.

Save the file and select the block in the Editor. You will now see both Color and Typography panels in the Settings Panel. Try modifying the settings and see what happens.

The block in the Editor with block supports

Removing unnecessary code

For simplicity, the styling for the Copyright Date Block will be controlled entirely by the color and typography block supports. This block also does not have any front-end JavaScript. Therefore, you don't need to specify stylesheets or a `viewScript` in the `block.json` file.

1. Remove the line for `editorStyle`
2. Remove the line for `style`
3. Remove the line for `viewScript`
4. Save the file

Refresh the Editor, and you will see that the block styling now matches your current theme.

The block in the Editor without default styling

Putting it all together

Your final `block.json` file should look like this:

```
{
    "$schema": "https://schemas.wp.org/trunk/block.json",
    "apiVersion": 3,
    "name": "create-block/copyright-date-block",
    "version": "0.1.0",
    "title": "Copyright Date Block",
    "category": "widgets",
    "description": "Display your site's copyright date.",
    "example": {},
    "supports": {
        "color": {
            "background": false,
            "text": true
        },
        "html": false,
        "typography": {
            "fontSize": true
        }
    },
    "textdomain": "copyright-date-block",
    "editorScript": "file:./index.js",
    "render": "file:./render.php"
}
```

Updating index.js

Before you start building the functionality of the block itself, let's do a bit more cleanup and add a custom icon to the block.

Open the `index.js` file. This is the main JavaScript file of the block and is used to register it on the client. You can learn more about client-side and server-side registration in the Registration of a block documentation.

Start by looking at the `registerBlockType` function. This function accepts the name of the block, which we are getting from the imported `block.json` file, and the block configuration object.

```
import Edit from './edit';
import metadata from './block.json';

registerBlockType( metadata.name, {
    edit: Edit,
} );
```

By default, the object just includes the `edit` property, but you can add many more, including `icon`. While most of these properties are already defined in `block.json`, you need to specify the icon here to use a custom SVG.

Adding a custom icon

Using the calendar icon from the Gutenberg Storybook, add the SVG to the function like so:

```
const calendarIcon = (
    <svg
        viewBox="0 0 24 24"
        xmlns="http://www.w3.org/2000/svg"
        aria-hidden="true"
        focusable="false"
    >
        <path d="M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2V5c0-1.1-.9-2-2-2zm.5 16c0 .3-.2.5-.5.5H5c-.3 0-.5-.2-.5-.5V7h15v12zM9 10H7v2h2v-2zm0 4H7v2h2v-2zm4-4h-2v2h2v-2zm0 4h-2v2h2v-2zm4-4h-2v2h2v-2zm0 4h-2v2h2v-2z"></path>
    </svg>
);

registerBlockType( metadata.name, {
    icon: calendarIcon,
    edit: Edit
} );
```

> All block icons should be 24 pixels square. Note the `viewBox` parameter above.

Save the `index.js` file and refresh the Editor. You will now see the calendar icon instead of the default.

The block in the Editor a custom icon

At this point, the block's icon and description are correct, and block supports allow you to change the font size and text color. Now, it's time to move on to the actual functionality of the block.

Updating edit.js

The `edit.js` file controls how the block functions and appears in the Editor. Right now, the user sees the message "Copyright Date Block – hello from the editor!". Let's change that.

Open the file and see that the `Edit()` function returns a paragraph tag with the default message.

```
export default function Edit() {
    return (
        <p { ...useBlockProps() }>
            { __(
                'Copyright Date Block – hello from the editor!',
                'copyright-date-block-demo'
            ) }
        </p>
    );
}
```

It looks a bit more complicated than it is.

- `useBlockProps()` outputs all the necessary CSS classes and styles in the block's wrapper needed by the Editor, which includes the style provided by the block supports you added earlier
- `__()` is used for the internationalization of text strings

> Review the block wrapper documentation for an introductory guide on how to ensure the block's markup wrapper has the proper attributes.

As a reminder, the main purpose of this block is to display the copyright symbol (©) and the current year. So, you first need to get the current year in string form, which can be done with the following code.

```
const currentYear = new Date().getFullYear().toString();
```

Next, update the function to display the correct information.

```
export default function Edit() {
    const currentYear = new Date().getFullYear().toString();

    return (
        <p { ...useBlockProps() }>© { currentYear }</p>
    );
}
```

Save the `edit.js` file and refresh the Editor. You will now see the copyright symbol (©) followed by the current year.

The block in the Editor displays the correct content

Updating render.php

While the block is working properly in the Editor, the default block message is still being displayed on the front end. To fix this, open the `render.php` file, and you will see the following.

```
<?php
...
?>
<p <?php echo get_block_wrapper_attributes(); ?>>
    <?php esc_html_e( 'Copyright Date Block – hello from a dynamic block!', 'copyright-date-block' ); ?>
</p>
```

Similar to the `useBlockProps()` function in the Editor, `get_block_wrapper_attributes()` outputs all the necessary CSS classes and styles in the block's wrapper. Only the content needs to be updated.

You can use `date( "Y" )` to get the current year in PHP, and your `render.php` should look like this.

```
<?php
...
?>
<p <?php echo get_block_wrapper_attributes(); ?>>© <?php echo date( "Y" ); ?></p>
```

Save the file and confirm that the block appears correctly in the Editor and on the front end.

Cleaning up

When you use the `create-block` package to scaffold a block, it might include files that you don't need. In the case of this tutorial, the block doesn't use stylesheets or front end JavaScript. Clean up the plugin's `src/` folder with the following actions.

1. In the `edit.js` file, remove the lines that import `editor.scss`
2. In the `index.js` file, remove the lines that import `style.scss`
3. Delete the editor.scss, style.scss, and view.js files

Finally, make sure that there are no unsaved changes and then terminate the `npm run start` command. Run `npm run build` to optimize your code and make it production-ready.

You have built a fully functional WordPress block, but let's not stop here. In the next sections, we'll add functionality and enable static rendering.

Adding block attributes

The Copyright Date Block you have built shows the current year, but what if you wanted to display a starting year as well?

What you're going to build

This functionality would require users to enter their starting year somewhere on the block. They should also have the ability to toggle it on or off.

You could implement this in different ways, but all would require block attributes. Attributes allow you to store custom data for the block that can then be used to modify the block's markup.

To enable this starting year functionality, you will need one attribute to store the starting year and another that will be used to tell WordPress whether the starting year should be displayed or not.

Updating block.json

Block attributes are generally specified in the `block.json` file. So open up the file and add the following section after the `example` property.

```
"example": {},
"attributes": {
    "showStartingYear": {
        "type": "boolean"
    },
    "startingYear": {
        "type": "string"
```

```
    }
},
```

You must indicate the `type` when defining attributes. In this case, the `showStartingYear` should be true or false, so it's set to `boolean`. The `startingYear` is just a string.

Save the file, and you can now move on to the Editor.

Updating edit.js

Open the `edit.js` file. You will need to accomplish two tasks.

- Add a user interface that allows the user to enter a starting year, toggle the functionality on or off, and store these settings as attributes.
- Update the block to display the correct content depending on the defined attributes.

Adding the user interface

Earlier in this tutorial, you added block supports that automatically created Color and Typography panels in the Settings Sidebar of the block. You can create your own custom panels using the `InspectorControls` component.

Inspector controls

The `InspectorControls` belongs to the `@wordpress/block-editor` package, so you can import it into the `edit.js` file by adding the component name on line 14. The result should look like this.

```
import { InspectorControls, useBlockProps } from '@wordpress/block-editor';
```

Next, update the Edit function to return the current block content and an `InspectorControls` component that includes the text "Testing." You can wrap everything in a Fragment (`<></>`) to ensure proper JSX syntax. The result should look like this.

```
export default function Edit() {
    const currentYear = new Date().getFullYear().toString();

    return (
        <>
            <InspectorControls>
                Testing
            </InspectorControls>
            <p { ...useBlockProps() }>© { currentYear }</p>
```

```
        </>
    );
}
```

Save the file and refresh the Editor. When selecting the block, you should see the “Testing” message in the Settings Sidebar.

The Setting Sidebar now displays the message

Components and panels

Now, let’s use a few more Core components to add a custom panel and the user interface for the starting year functionality. You will want to import `PanelBody`, `TextControl`, and `ToggleControl` from the `@wordpress/components` package.

Add the following line below the other imports in the `edit.js` file.

```
import { PanelBody, TextControl, ToggleControl } from '@wordpress/components';
```

Then wrap the “Testing” message in the `PanelBody` component and set the `title` parameter to “Settings”. Refer to the component documentation for additional parameter options.

```
export default function Edit() {
    const currentYear = new Date().getFullYear().toString();

    return (
        <>
            <InspectorControls>
                <PanelBody title={ __( 'Settings', 'copyright-date-block' ) }>
                    Testing
                </PanelBody>
            </InspectorControls>
            <p { ...useBlockProps() }>© { currentYear }</p>
        </>
    );
}
```

Save the file and refresh the Editor. You should now see the new Settings panel.

The Setting Sidebar now displays a custom panel

Text control

The next step is to replace the “Testing” message with a `TextControl` component that allows the user to set the `startingYear` attribute. However, you must include two parameters in the `Edit()` function before doing so.

- `attributes` is an object that contains all the attributes for the block
- `setAttributes` is a function that allows you to update the value of an attribute

With these parameters included, you can fetch the `showStartingYear` and `startingYear` attributes.

Update the top of the `Edit()` function to look like this.

```
export default function Edit( { attributes, setAttributes } ) {
    const { showStartingYear, startingYear } = attributes;
    ...
```

> To see all the attributes associated with the Copyright Date Block, add `console.log( attributes );` at the top of the `Edit()` function. This can be useful when building and testing a custom block.

Now, you can remove the “Testing” message and add a `TextControl`. It should include:

1. A `label` property set to “Starting year”
2. A `value` property set to the attribute `startingYear`
3. An `onChange` property that updates the `startingYear` attribute whenever the value changes

Putting it all together, the `Edit()` function should look like the following.

```
export default function Edit( { attributes, setAttributes } ) {
    const { showStartingYear, startingYear } = attributes;
    const currentYear = new Date().getFullYear().toString();

    return (
        <>
            <InspectorControls>
                <PanelBody title={ __( 'Settings', 'copyright-date-block' ) }>
                    <TextControl
                        label={ __(
                            'Starting year',
                            'copyright-date-block'
                        ) }
                        value={ startingYear || '' }
```

```
                        onChange={ ( value ) =>
                            setAttributes( { startingYear: value } )
                        }
                    />
                </PanelBody>
            </InspectorControls>
            <p { ...useBlockProps() }>© { currentYear }</p>
        </>
    );
}
```

> You may have noticed that the `value` property has a value of `startingYear || ''`. The symbol `||` is called the Logical OR (logical disjunction) operator. This prevents warnings in React when the `startingYear` is empty. See Controlled and uncontrolled components for details.

Save the file and refresh the Editor. Confirm that a text field now exists in the Settings panel. Add a starting year and confirm that when you update the page, the value is saved.

A live look at editing the new Starting Year field in the Settings Sidebar

Toggle control

Next, let's add a toggle that will turn the starting year functionality on or off. You can do this with a `ToggleControl` component that sets the `showStartingYear` attribute. It should include:

1. A `label` property set to "Show starting year"
2. A `checked` property set to the attribute `showStartingYear`
3. An `onChange` property that updates the `showStartingYear` attribute whenever the toggle is checked or unchecked

You can also update the "Starting year" text input so it's only displayed when `showStartingYear` is `true`, which can be done using the `&&` logical operator.

The `Edit()` function should look like the following.

```
export default function Edit( { attributes, setAttributes } ) {
    const { showStartingYear, startingYear } = attributes;
    const currentYear = new Date().getFullYear().toString();

    return (
        <>
```

```
            <InspectorControls>
                <PanelBody title={ __( 'Settings', 'copyright-date-block' ) }>
                    <ToggleControl
                        checked={ !! showStartingYear }
                        label={ __(
                            'Show starting year',
                            'copyright-date-block'
                        ) }
                        onChange={ () =>
                            setAttributes( {
                                showStartingYear: ! showStartingYear,
                            } )
                        }
                    />
                    { showStartingYear && (
                        <TextControl
                            label={ __(
                                'Starting year',
                                'copyright-date-block'
                            ) }
                            value={ startingYear || '' }
                            onChange={ ( value ) =>
                                setAttributes( { startingYear: value } )
                            }
                        />
                    ) }
                </PanelBody>
            </InspectorControls>
            <p { ...useBlockProps() }>© { currentYear }</p>
        </>
    );
}
```

Save the file and refresh the Editor. Confirm that clicking the toggle displays the text input, and when you update the page, the toggle remains active.

A live look at editing the new Show Starting Year toggle in the Settings Sidebar

Updating the block content

So far, you have created the user interface for adding a starting year and updating the associated block attributes. Now you need to actually update the block content in the Editor.

Let's create a new variable called `displayDate`. When `showStartingYear` is `true` and the user has provided a `startingYear`, the `displayDate` should include the `startingYear` and the `currentYear` separated by an em dash. Otherwise, just display the `currentYear`.

The code should look something like this.

```
let displayDate;

if ( showStartingYear && startingYear ) {
    displayDate = startingYear + '—' + currentYear;
} else {
    displayDate = currentYear;
}
```

> When you declare a variable with `let`, it means that the variable may be reassigned later. Declaring a variable with `const` means that the variable will never change. You could rewrite this code using `const`. It's just a matter of personal preference.

Next, you just need to update the block content to use the `displayDate` instead of the `currentYear` variable.

The `Edit()` function should look like the following.

```
export default function Edit( { attributes, setAttributes } ) {
    const { showStartingYear, startingYear } = attributes;
    const currentYear = new Date().getFullYear().toString();

    let displayDate;

    if ( showStartingYear && startingYear ) {
            displayDate = startingYear + '—' + currentYear;
    } else {
        displayDate = currentYear;
    }

    return (
        <>
            <InspectorControls>
```

```
                <PanelBody title={ __( 'Settings', 'copyright-date-block' ) }>
                    <ToggleControl
                        checked={ !! showStartingYear }
                        label={ __(
                            'Show starting year',
                            'copyright-date-block'
                        ) }
                        onChange={ () =>
                            setAttributes( {
                                showStartingYear: ! showStartingYear,
                            } )
                        }
                    />
                    { showStartingYear && (
                        <TextControl
                            label={ __(
                                'Starting year',
                                'copyright-date-block'
                            ) }
                            value={ startingYear || '' }
                            onChange={ ( value ) =>
                                setAttributes( { startingYear: value } )
                            }
                        />
                    ) }
                </PanelBody>
            </InspectorControls>
            <p { ...useBlockProps() }>© { displayDate }</p>
        </>
    );
}
```

Save the file and refresh the Editor. Confirm that the block content updates correctly when you make changes in the Settings panel.

A live look at the block content being updated by the new fields in the Setting Sidebar

Updating render.php

While the Editor looks great, the starting year functionality has yet to be added to the front end. Let's fix that by updating the `render.php` file.

Start by adding a variable called `$display_date` and replicate what you did in the `Edit()` function above.

This variable should display the value of the `startingYear` attribute and the `$current_year` variable separated by an em dash, or just the `$current_year` if the `showStartingYear` attribute is `false`.

> Three variables are exposed in `render.php`, which you can use to customize the block's output:
>
> - `$attributes` (array): The block attributes.
> - `$content` (string): The block default content.
> - `$block` (WP_Block): The block instance.

The code should look something like this.

```
if ( ! empty( $attributes['startingYear'] ) && ! empty( $attributes['showStartingYear'] ) ) {
    $display_date = $attributes['startingYear'] . '–' . $current_year;
} else {
    $display_date = $current_year;
}
```

Next, you just need to update the block content to use the `$display_date` instead of the `$current_year` variable.

Your final `render.php` file should look like this.

```
<?php
$current_year = date( "Y" );

if ( ! empty( $attributes['startingYear'] ) && ! empty( $attributes['showStartingYear'] ) ) {
    $display_date = $attributes['startingYear'] . '–' . $current_year;
} else {
    $display_date = $current_year;
}
?>
<p <?php echo get_block_wrapper_attributes(); ?>>
    © <?php echo esc_html( $display_date ); ?>
</p>
```

Save the file and confirm that the correct block content is now appearing on the front end of your site.

You have now successfully built a dynamically rendered custom block that utilizes block supports, core WordPress components, and custom attributes. In many situations, this is as far as you would need to go for a block displaying the copyright date with some additional functionality.

In the next section, however, you will add static rendering to the block. This exercise will illustrate how block data is stored in WordPress and provide a fallback should this plugin ever be inadvertently disabled.

Adding static rendering

A block can utilize dynamic rendering, static rendering, or both. The block you have built so far is dynamically rendered. Its block markup and associated attributes are stored in the database, but its HTML output is not.

Statically rendered blocks will always store the block markup, attributes, and output in the database. Blocks can also store static output in the database while being further enhanced dynamically on the front end, a combination of both methods.

You will see the following if you switch to the Code editor from within the Editor.

```
<!-- wp:create-block/copyright-date-block {"showStartingYear":true,"startingYear":"2017"} /-->
```

Compare this to a statically rendered block like the Paragraph block.

```
<!-- wp:paragraph -->
<p>This is a test.</p>
<!-- /wp:paragraph -->
```

The HTML of the paragraph is stored in post content and saved in the database.

You can learn more about dynamic and static rendering in the Fundamentals documentation. While most blocks are either dynamically or statically rendered, you can build a block that utilizes both methods.

Why add static rendering?

When you add static rendering to a dynamically rendered block, the `render.php` file will still control the output on the front end, but the block's HTML content will be saved in the database. This means that the content will remain if the plugin is ever removed from the site. In the case of this

Copyright Date Block, the content will revert to a Custom HTML block that you can easily convert to a Paragraph block.

An error message in the Editor when a block type no longer exists

While not necessary in all situations, adding static rendering to a dynamically rendered block can provide a helpful fallback should the plugin ever be disabled unintentionally.

Also, consider a situation where the block markup is included in a block pattern or theme template. If a user installs that theme or uses the pattern without the Copyright Date Block installed, they will get a notice that the block is not available, but the content will still be displayed.

Adding static rendering is also a good exploration of how block content is stored and rendered in WordPress.

Adding a save function

Start by adding a new file named `save.js` to the `src/` folder. In this file, add the following.

```
import { useBlockProps } from '@wordpress/block-editor';

export default function save() {
    return (
        <p { ...useBlockProps.save() }>
            { 'Copyright Date Block – hello from the saved content!' }
        </p>
    );
}
```

This should look similar to the original `edit.js` file, and you can refer to the block wrapper documentation for additional information.

Next, in the `index.js` file, import this `save()` function and add a save property to the `registerBlockType()` function. Here's a simplified view of the updated file.

```
import save from './save';

...

registerBlockType( metadata.name, {
    icon: calendarIcon,
    edit: Edit,
    save
```

```
} );
```

> When defining properties of an object, if the property name and the variable name are the same, you can use shorthand property names. This is why the code above uses `save` instead of `save: save`.

Save both `save.js` and `index.js` files and refresh the Editor. It should look like this.

A block validation error message in the Editor

Don’t worry, the error is expected. If you open the inspector in your browser, you should see the following message.

A block validation error message in the console

This block validation error occurs because the `save()` function returns block content, but no HTML is stored in the block markup since the previously saved block was dynamic. Remember that this is what the markup currently looks like.

```
<!-- wp:create-block/copyright-date-block {"showStartingYear":true,"startingYear":"2017"} /-->
```

You will see more of these errors as you update the `save()` function in subsequent steps. Just click “Attempt Block Recovery” and update the page.

After performing block recovery, open the Code editor and you will see the markup now looks like this.

```
<!-- wp:create-block/copyright-date-block {"showStartingYear":true,"startingYear":"2017"} -->
<p class="wp-block-create-block-copyright-date-block">Copyright Date Block – hello from the saved content!</p>
<!-- /wp:create-block/copyright-date-block -->
```

You will often encounter block validation errors when building a block with static rendering, and that’s ok. The output of the `save()` function must match the HTML in the post content exactly, which may end up out of sync as you add functionality. So long as there are no validation errors when you’re completely finished building the block, you will be all set.

Updating save.js

Next, let’s update the output of the `save()` function to display the correct content. Start by copying the same approach used in `edit.js`.

1. Add the `attributes` parameter to the function
2. Define the `showStartingYear` and `startingYear` variables

3. Define a `currentYear` variable
4. Define a `displayDate` variable depending on the values of `currentYear`, `showStartingYear`, and `startingYear`

The result should look like this.

```
export default function save( { attributes } ) {
    const { showStartingYear, startingYear } = attributes;
    const currentYear = new Date().getFullYear().toString();

    let displayDate;

    if ( showStartingYear && startingYear ) {
        displayDate = startingYear + '–' + currentYear;
    } else {
        displayDate = currentYear;
    }

    return (
        <p { ...useBlockProps.save() }>© { displayDate }</p>
    );
}
```

Save the file and refresh the Editor. Click “Attempt Block Recovery” and update the page. Check the Code editor, and the block markup should now look something like this.

```
<!-- wp:create-block/copyright-date-block {"showStartingYear":true,"startingYear":"2017"} -->
<p class="wp-block-create-block-copyright-date-block">© 2017–2023</p>
<!-- /wp:create-block/copyright-date-block -->
```

At this point, it might look like you’re done. The block content is now saved as HTML in the database and the output on the front end is dynamically rendered. However, there are still a few things that need to be addressed.

Consider the situation where the user added the block to a page in 2023 and then went back to edit the page in 2024. The front end will update as expected, but in the Editor, there will be a block validation error. The `save()` function knows that it’s 2024, but the block content saved in the database still says 2023.

Let’s fix this in the next section.

Handling dynamic content in statically rendered blocks

Generally, you want to avoid dynamic content in statically rendered blocks. This is part of the reason why the term “dynamic” is used when referring to dynamic rendering.

That said, in this tutorial, you are combining both rendering methods, and you just need a bit more code to avoid any block validation errors when the year changes.

The root of the issue is that the `currentYear` variable is set dynamically in the `save()` function. Instead, this should be a static variable within the function, which can be solved with an additional attribute.

Adding a new attribute

Open the `block.json` file and add a new attribute called `fallbackCurrentYear` with the type `string`. The `attributes` section of the file should now look like this.

```
"attributes": {
    "fallbackCurrentYear": {
        "type": "string"
    },
    "showStartingYear": {
        "type": "boolean"
    },
    "startingYear": {
        "type": "string"
    }
},
```

Next, open the `save.js` file and use the new `fallbackCurrentYear` attribute in place of `currentYear`. Your updated `save()` function should look like this.

```
export default function save( { attributes } ) {
    const { fallbackCurrentYear, showStartingYear, startingYear } = attributes;

    let displayDate;

    if ( showStartingYear && startingYear ) {
        displayDate = startingYear + '–' + fallbackCurrentYear;
```

```
    } else {
        displayDate = fallbackCurrentYear;
    }

    return (
        <p { ...useBlockProps.save() }>© { displayDate }</p>
    );
}
```

Now, what happens if the `fallbackCurrentYear` is undefined?

Before the `currentYear` was defined within the function, so the `save()` function always had content to return, even if `showStartingYear` and `startingYear` were undefined.

Instead of returning just the copyright symbol, let's add a condition that if `fallbackCurrentYear` is not set, return `null`. It's generally better to save no HTML in the database than incomplete data.

The final `save()` function should look like this.

```
export default function save( { attributes } ) {
    const { fallbackCurrentYear, showStartingYear, startingYear } = attributes;

    if ( ! fallbackCurrentYear ) {
        return null;
    }

    let displayDate;

    if ( showStartingYear && startingYear ) {
        displayDate = startingYear + '–' + fallbackCurrentYear;
    } else {
        displayDate = fallbackCurrentYear;
    }

    return (
        <p { ...useBlockProps.save() }>© { displayDate }</p>
    );
}
```

Save both the `block.json` and `save.js` files; you won't need to make any more changes.

Setting the attribute in edit.js

The `save()` function now uses the new `fallbackCurrentYear`, so it needs to be set somewhere. Let's use the `Edit()` function.

Open the `edit.js` file and start by defining the `fallbackCurrentYear` variable at the top of the `Edit()` function alongside the other attributes. Next, review what's happening in the function.

When the block loads in the Editor, the `currentYear` variable is defined. The function then uses this variable to set the content of the block.

Now, let's set the `fallbackCurrentYear` attribute to the `currentYear` when the block loads if the attribute is not already set.

```
if ( currentYear !== fallbackCurrentYear ) {
    setAttributes( { fallbackCurrentYear: currentYear } );
}
```

This will work but can be improved by ensuring this code only runs once when the block is initialized. To do so, you can use the `useEffect` React hook. Refer to the React documentation for more information about how to use this hook.

First, import `useEffect` with the following code.

```
import { useEffect } from 'react';
```

Then wrap the `setAttribute()` code above in a `useEffect` and place this code after the `currentYear` definition in the `Edit()` function. The result should look like this.

```
export default function Edit( { attributes, setAttributes } ) {
    const { fallbackCurrentYear, showStartingYear, startingYear } = attributes;

    // Get the current year and make sure it's a string.
    const currentYear = new Date().getFullYear().toString();

    // When the block loads, set the fallbackCurrentYear attribute to the
    // current year if it's not already set.
    useEffect( () => {
        if ( currentYear !== fallbackCurrentYear ) {
            setAttributes( { fallbackCurrentYear: currentYear } );
        }
    }, [ currentYear, fallbackCurrentYear, setAttributes ] );

    ...
```

When the block is initialized in the Editor, the `fallbackCurrentYear` attribute will be immediately set. This value will then be available to the `save()` function, and the correct block content will be displayed without block validation errors.

The one caveat is when the year changes. If a Copyright Date Block was added to a page in 2023 and then edited in 2024, the `fallbackCurrentYear` attribute will no longer equal the `currentYear`, and the attribute will be automatically updated to `2024`. This will update the HTML returned by the `save()` function.

You will not get any block validation errors, but the Editor will detect that changes have been made to the page and prompt you to update.

Optimizing render.php

The final step is to optimize the `render.php` file. If the `currentYear` and the `fallbackCurrentYear` attribute are the same, then there is no need to dynamically create the block content. It is already saved in the database and is available in the `render.php` file via the `$content` variable.

Therefore, update the file to render the generated content if `currentYear` and `fallbackCurrentYear` do not match.

```
$current_year = date( "Y" );

// Determine which content to display.
if ( isset( $attributes['fallbackCurrentYear'] ) && $attributes['fallbackCurrentYear'] === $current_year ) {

    // The current year is the same as the fallback, so use the block content saved in the database (by the save.js function).
    $block_content = $content;
} else {

    // The current year is different from the fallback, so render the updated block content.
    if ( ! empty( $attributes['startingYear'] ) && ! empty( $attributes['showStartingYear'] ) ) {
        $display_date = $attributes['startingYear'] . '–' . $current_year;
    } else {
        $display_date = $current_year;
    }

    $block_content = '<p ' . get_block_wrapper_attributes() . '>© ' . esc_html( $display_date ) . '</p>';
}
```

```
echo wp_kses_post( $block_content );
```

That’s it! You now have a block that utilizes both dynamic and static rendering.

Wrapping up

Congratulations on completing this tutorial and building your very own Copyright Date Block. Throughout this journey, you have gained a solid foundation in WordPress block development and are now ready to start building your own blocks.

For final reference, the complete code for this tutorial is available in the Block Development Examples repository on GitHub.

Now, whether you’re now looking to refine your skills, tackle more advanced projects, or stay updated with the latest WordPress trends, the following resources will help you improve your block development skills:

- Block Development Environment
- Fundamentals of Block Development
- WordPress Developer Blog
- Block Development Examples | GitHub repository

Remember, every expert was once a beginner. Keep learning, experimenting, and, most importantly, have fun building with WordPress.

□□□ #2
□ □□□ □□□ 2026-10-31 18:26:53 UTC
□ □□□ □□□ 2026-10-31 18:26:55 UTC