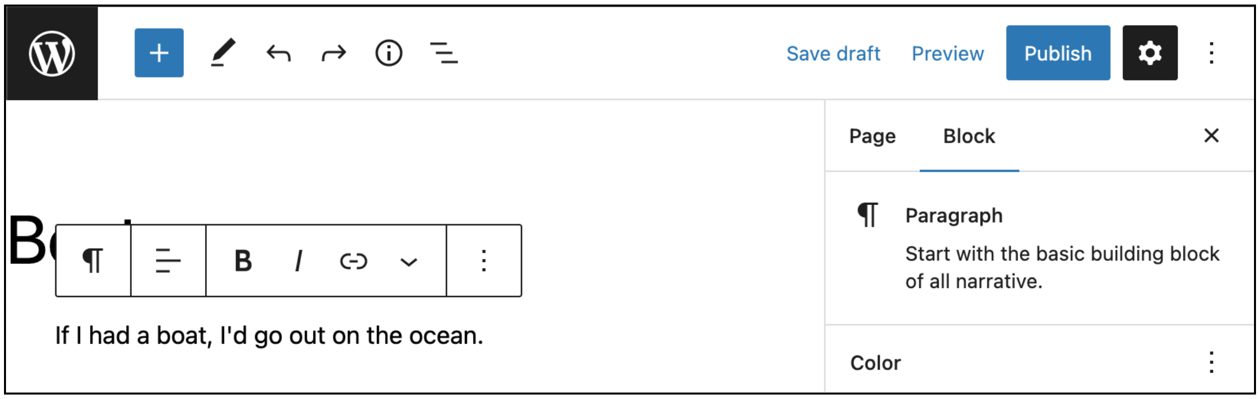
Static vs. dynamic blocks: What’s the difference?

The WordPress Block Editor (a.k.a. Gutenberg) offers two types of blocks: static and dynamic. The difference between these two types of blocks comes down to how they are rendered on the front-end.

Static blocks

A static block is a piece of content whose markup is known when the page is saved. The block saves its content and markup directly in the post content. The Paragraph block is a simple example of a static block.



If I had a boat, I'd go out on the ocean.

*Example of the **Paragraph** static block, displayed in the editor.*

The words inside a paragraph will not change unless a content editor manually changes them. Both the content and markup are known, which makes the Paragraph block a great example of a static block. Its HTML markup is directly saved to the post content.

A static block is written entirely in JavaScript. The `save()` function writes the block’s markup to the `post_content` entry for the post in the `wp_posts` database table. The entry for the example above will contain that exact markup, plus Gutenberg block indicator inline-comments:

```
<!-- wp:paragraph -->
<p>If I had a boat, I'd go out on the ocean.</p>
<!-- /wp:paragraph -->
```

The block indicators are part of the overall block grammar. WordPress uses these HTML comments to define the block and any attributes or metadata it has. WordPress parses these comments to display blocks both in the Editor and on the front-end, but the comments are never rendered in the source code. For a static block, the source code will only include the markup inside the block indicators.

A deeper dive

A static block's `save()` function and the content saved to the database are tightly related to each other. The Post Editor runs a validation check to ensure that the markup created by the `save()` function is identical to the markup that has already been saved to the database. If there are any differences, the Post Editor becomes very unhappy and declares the block to be broken. This is called a block validation error.

A content editor has the option to attempt a block recovery. This will attempt to reconcile the content in the database with what the block is expecting to save to the database, based on the block's attributes. This recovery process is sometimes, but not always, successful.

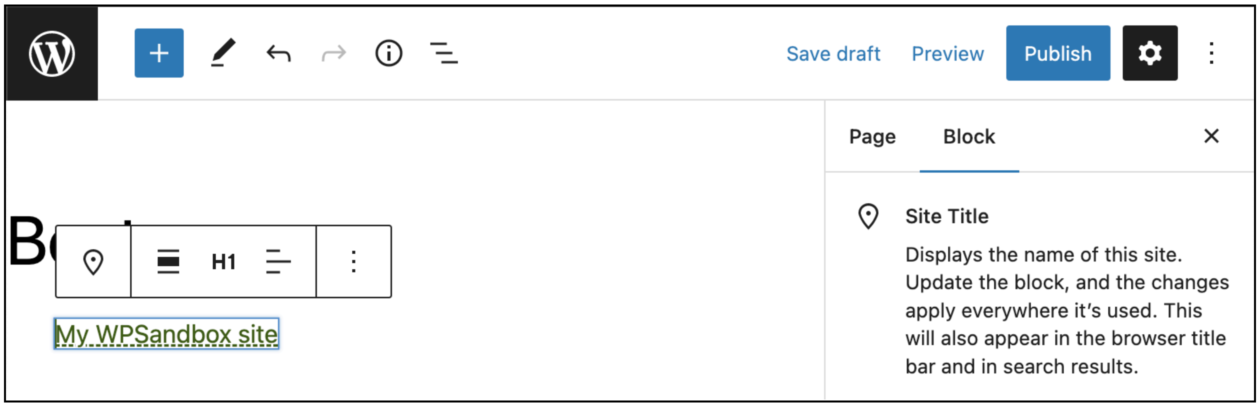
Block validation errors are most commonly caused when a block's `save()` function is updated to change the markup produced by the block. Any small change, even to a class name, can trigger a validation error in the Editor.

A block developer can mitigate these issues by adding a block deprecation to register the change in the block. The block deprecation keeps a record of previous versions of a block's markup. The Post Editor can then compare a block's current content with a previous version and (ideally) avoid a validation error.

Dynamic blocks

A dynamic block is a piece of content whose markup and exact content are not known when the page is saved. This block could contain content that is timely or dependent on changes in other parts of the site. The contents of a dynamic block are expected to change without the intervention of a content editor. As a result, the markup of a dynamic block is rendered on the server-side.

A simple example is the core Site Title block, which displays the site's name. This block must be dynamic because its content cannot be known at the time the page is saved. The site title can change at any time via the site settings.


Save draft
Preview
Publish
Page
Block
Site Title
Displays the name of this site. Update the block, and the changes apply everywhere it's used. This will also appear in the browser title bar and in search results.
H1
My WPSandbox site

Example of the ***Site Title*** *dynamic block, displayed in the WordPress post editor. This block displays the name of the site.*

A deeper dive

A dynamic block is registered in both JavaScript and PHP. While the JavaScript side handles the editor experience, the PHP side handles the server-side rendering for the front-end markup.

PHP block registration uses the `register_block_type()` function, which requires a render method to be defined. This can happen in one of two ways:

1. The registration function includes a `render_callback` argument.
2. A `render` property is added to `block.json`, whose value points to a separate PHP file.

These render methods automatically receive attribute and inner content information. They can also get additional required dynamic site information - post lists, comments, taxonomy information, etc.

Blocks are still saved as part of the `post_content` entry of the post. However, none of the block's markup is saved to the database. Instead, its attributes are written inside the block indicator comment. The render method is called to create the block's markup whenever a front-end user visits the page.

As an example, let us think about the above paragraph block as though it is a dynamic block. In this case, a serialized form of the block is saved to the database. This form uses a self-closing block indicator comment. Block attributes are saved as key-value pairs in an JSON encoded string:

```
<!— wp:paragraph { “content”: “If I had a boat, I'd go out on the ocean.”
} /—>
```

The block's render method would take the content attribute and apply markup. The front-end code would be the same as the block's static version.

Which type should you use?

If you are a block developer, one of the first decisions you will likely make is whether to write your block as a static block or a dynamic one.

Sometimes, the requirements of the block will make that decision for you. Does your block rely on information from other parts of the site, such as other posts, taxonomies, or site settings? If so, the block *must* be dynamic. The content relies on information outside of the page it lives on, so it can change.

On the other hand, if you can guarantee that the block's contents will always remain the same, you still have a choice. A static block is the obvious choice since the content is, well, static. However, there are arguments for opting for a dynamic block in this case.

Advantages of a static block

Static blocks are inherently simpler because they are written in a single language, JavaScript. Their markup is known at the time the page is saved, so, as mentioned above, all of that HTML code can be saved directly to the database.

This approach is more performant. When a visitor views the page, the content comes from the database. There is no delay in displaying the block since no server-side rendering is required.

Advantages of a dynamic block

The markup of a dynamic block is expected to change. This is why that markup is not saved to the database. As a result, that markup is not subject to the Post Editor's validation. This means that changes to a dynamic block's markup cannot throw a validation error. Block validation errors are fairly complex and are wonderfully explained in this article about Choosing Dynamic Blocks from NC State University.

This can be advantageous for block developers on more agile teams, where markup changes may be more likely. For example, a team may wish to make UX or accessibility improvements to a block. If that block is a dynamic block, changes to the markup are simply made to the PHP `render()` function. No additions to the deprecation array are required.

Conclusion

Block developers have a choice about what type of block they want to develop. Static blocks are great for content and markup that will not change. Dynamic blocks are designed for content dependent on external factors, but have advantages for static content as well. In the end, it is up to the block developer to make the right choice for their content and situation.

Props to @fabiankaegy, @greenshady, @mburridge, and @webcommsat for reviewing this post.

—

Excerpt:

The Block Editor offers two types of blocks: static and dynamic. The difference between these two types of blocks comes down to how they are rendered on the front-end. Read on to learn more about the details, advantages, and disadvantages of each.