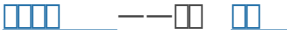




□□

- [□□□□](#)
  - [The block object](#)
- [□□□□□](#)
  - [Delimiters and parsing expression grammar](#)
- [□□□□□□□□](#)
- [□□□□□](#)



```
const value = [ block1, block2, block3 ];
```

# The block object

Each block object has an id, a set of attributes and potentially a list of child blocks.

```
const block = {  
  clientId, // unique string identifier.  
  type, // The block type (paragraph, image...)  
  attributes, // (key, value) set of attributes representing the direct properties/content of  
              the current block.
```

```
  innerBlocks, // An array of child blocks or inner blocks.
};
```

Note the attributes keys and types, the allowed inner blocks are defined by the block type. For example, the core quote block has a `cite` string attribute representing the cite content while a heading block has a numeric `level` attribute, representing the level of the heading (1 to 6).

During the lifecycle of the block in the editor, the block object can receive extra metadata:

- `isValid`: A boolean representing whether the block is valid or not;
- `originalContent`: The original HTML serialization of the block.

## Examples

```
// A simple paragraph block.
const paragraphBlock1 = {
  clientId: '51828be1-5f0d-4a6b-8099-f4c6f897e0a3',
  type: 'core/paragraph',
  attributes: {
    content: 'This is the <strong>content</strong> of the paragraph block',
    dropCap: true,
  },
};

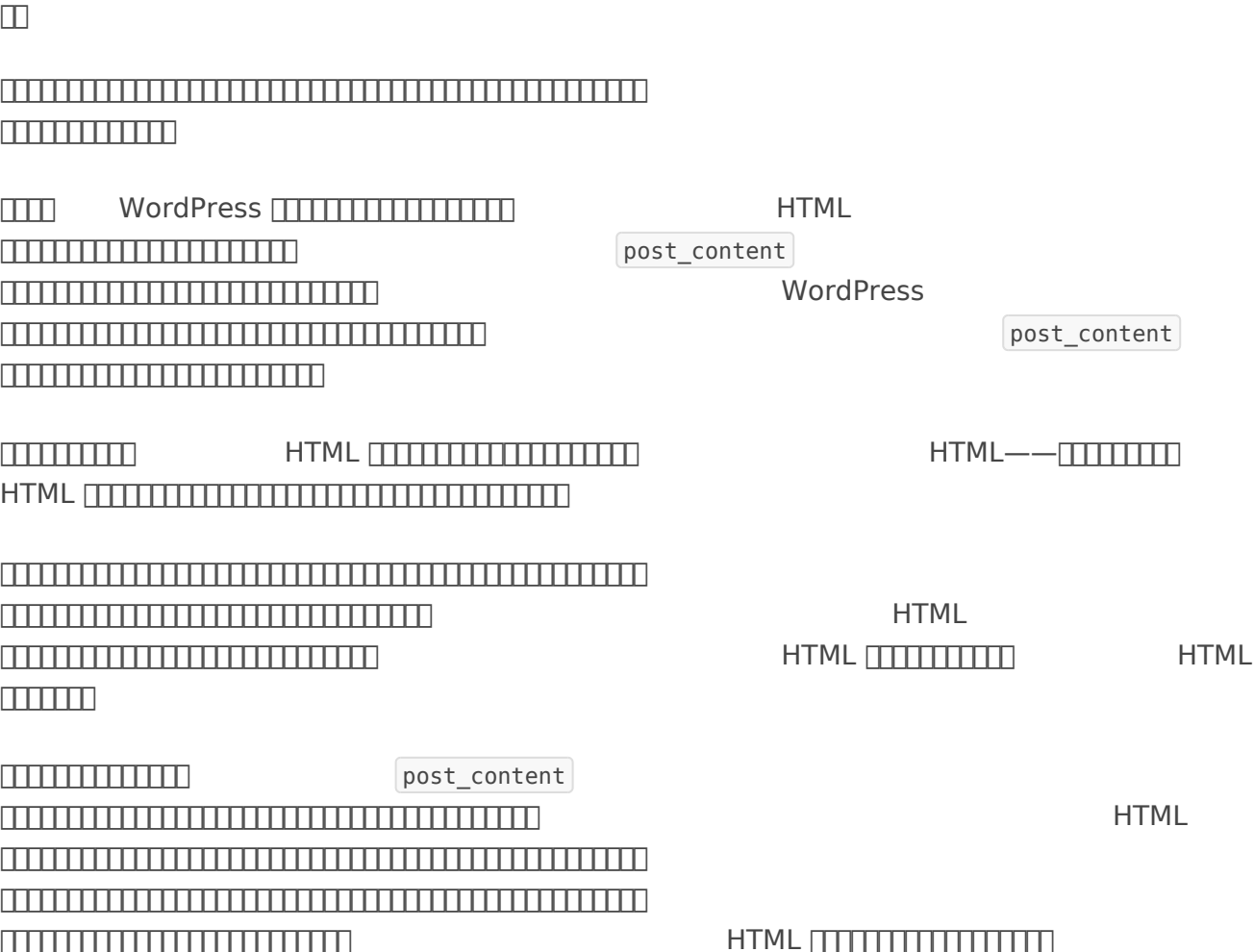
// A separator block.
const separatorBlock = {
  clientId: '51828be1-5f0d-4a6b-8099-f4c6f897e0a4',
  type: 'core/separator',
  attributes: {},
};

// A columns block with a paragraph block on each column.
const columnsBlock = {
  clientId: '51828be1-5f0d-4a6b-8099-f4c6f897e0a7',
  type: 'core/columns',
  attributes: {},
  innerBlocks: [
    {
      clientId: '51828be1-5f0d-4a6b-8099-f4c6f897e0a5',
      type: 'core/column',
      attributes: {},
    }
  ]
};
```

```

    innerBlocks: [ paragraphBlock1 ],
  },
  {
    clientId: '51828be1-5f0d-4a6b-8099-f4c6f897e0a6',
    type: 'core/column',
    attributes: {},
    innerBlocks: [ paragraphBlock2 ],
  },
],
};

```



# Delimiters and parsing expression grammar

We chose instead to try to find a way to keep the formality, explicitness, and unambiguity in the existing HTML syntax. Within the HTML there were a number of options.

Of these options, a novel approach was suggested: by storing data in HTML comments, we would know that we wouldn't break the rest of the HTML in the document, that browsers should ignore it, and that we could simplify our approach to parsing the document.

Unique to HTML comments is the fact that they cannot legitimately exist in ambiguous places, such as inside of HTML attributes like `<img alt='data-id="14"'>`. Comments are also quite permissive. Whereas HTML attributes are complicated to parse properly, comments are quite easily described by a leading `<!--` followed by anything except `--` until the first `-->`. This simplicity and permissiveness means that the parser can be implemented in several ways without needing to understand HTML properly, and we have the liberty to use more convenient syntax inside of the comment—we only need to escape double-hyphen sequences. We take advantage of this in how we store block attributes: as JSON literals inside the comment.

After running this through the parser, we're left with a simple object we can manipulate idiomatically, and we don't have to worry about escaping or unescaping the data. It's handled for us through the serialization process. Because the comments are so different from other HTML tags and because we can perform a first-pass to extract the top-level blocks, we don't actually depend on having fully valid HTML!

This has dramatic implications for how simple and performant we can make our parser. These explicit boundaries also protect damage in a single block from bleeding into other blocks or tarnishing the entire document. It also allows the system to identify unrecognized blocks before rendering them.

*N.B.:* The defining aspects of blocks are their semantics and the isolation mechanism they provide: in other words, their identity. On the other hand, where their data is stored is a more liberal aspect. Blocks support more than just static local data (via JSON literals inside the HTML comment or within the block's HTML), and more mechanisms (e.g., global blocks or otherwise resorting to storage in complementary `WP_Post` objects) are expected. See [attributes](#) for details.



```
<!-- wp:image -->
<figure class="wp-block-image"></figure>
<!-- /wp:image -->
```



```
<!-- wp:latest-posts {"postsToShow":4,"displayPostDate":true} /-->
```



```
post_content
```

[illegible]

## JSON



□□□ #2

#2



2026-03-26 17:08:43 UTC

|  |  |  |
|--|--|--|
|  |  |  |
|--|--|--|

2026-03-26 17:08:43 UTC