

Simplicity Builder™

Reference Manual

Version 1.0
March 11, 2024



Contents

- Overview..... 4
- Developer Benefits..... 4
- Getting Started 5
- Common Concepts..... 5
 - Core Architecture..... 6
 - Template 6
 - Template Block Properties..... 7
- Configuration Object 15
 - Configuration Object Properties 17
 - license (string)..... 17
 - devKey (string) 17
 - themeColor (string)..... 17
 - themeDarkColor (string)..... 17
 - assetsLocation (string) 18
 - helpUrl (string)..... 18
 - cleanupAttributes (array [] of strings)..... 18
 - cleanupStrings (array [] of strings) 18
 - cleanupIds (array [] of strings)..... 18
 - blockSections (array [] of block section objects{ })..... 18
 - imageSelections (array [] of image objects { }) 19
 - attributeEditors (array [] of Attribute Editor objects { }) 20
- Defining Blocks 20
 - Block Sections 20
 - icon..... 21
 - label..... 21
 - tooltipHeading..... 22
 - tooltipImage 22
 - tooltipText..... 23
 - dragImage 23
 - html 24
- Defining Attribute Editors 25
 - Control Types..... 26
 - checkboxset..... 27



- color 27
- image..... 27
- select 28
- slider..... 28
- text..... 28
- text (mode = textarea) 28
- toggle..... 28
- Defining Attribute Editor Objects..... 29
 - editorHeading..... 30
 - mode..... 30
 - callback 30
 - controlSections..... 30
- Vanilla JavaScript Example 38
 - The Easiest Way to Start..... 38
 - Starting From Scratch 38
- Vue Example..... 45
 - The Easiest Way to Start..... 45
 - Starting From Scratch 45
- React Example 53
 - The Easiest Way to Start..... 53
- Angular Example (standalone) 54
- Angular Example (module)..... 55



Overview

Simplicity Builder™ is a versatile web component that lets you add simplified visual drag and drop builder functionality into your web application. It is fully configurable and lets you turn any html into a drag and drop builder. You are in full control of how much or how little you want your users to be able to edit or add.

**TURN ANY HTML INTO A SIMPLE-TO-USE
DRAG AND DROP BUILDER**

Developer Benefits

- **Framework Flexibility:** Available in wrapped versions for major JavaScript frameworks, including Vanilla JavaScript, React, Vue.js, and Angular. The Vanilla JS version is compatible with additional frameworks, ensuring broad usability.
- **Simplified Implementation:** Jumpstart development by creating templates and configuration files. Adjust configurations dynamically to cater to different templates, streamlining the development process.
- **Rapid Development:** Shortcut the development cycle by months, leveraging Simplicity Builder™ for a wide range of applications, from page and template builders to low code/no code platforms, configurable dashboards, and widget configurators.
- **Editable Output Code:** Generate clean, maintainable code that integrates seamlessly with standard development tools, enhancing productivity and collaboration.
- **Comprehensive Editing Suite:** Includes a rich text WYSIWYG editor, text editor, and code editor components, offering a versatile editing experience for developers and end-users alike.
- **Flexible Content Blocks:** Convert any HTML into draggable blocks for easy placement within templates, enriching the user experience with dynamic, customizable content.
- **Configurable Dynamic Attributes:** Tailor each block with a dynamic attribute editor, providing fine-grained control over the properties and behavior of your content.
- **Publish Method:** The component publish method allows for saving files that clean up portions of your file that are meant for design time only.



Getting Started

The Simplicity Builder™ web component requires a trial key or a developer key to develop with. A free trial key is available with no credit card information needed. Go to:

<https://www.SimplicityWebTools>

We have four different versions of the component available. Your license key works with all versions. The four available are:

- Vanilla JavaScript (actually works anywhere)
- React Wrapped
- Vue Wrapped
- Angular Wrapped

We have separate examples of using each of the above. Each is in a public repo that you can clone or download and see a basic example working that demonstrates the concepts of the Simplicity Builder™ web component.

Please remember that you will need to obtain at least a free trial dev key (available without using any credit card info). The key can be obtained from

<https://www.SimplicityWebTools.com>.

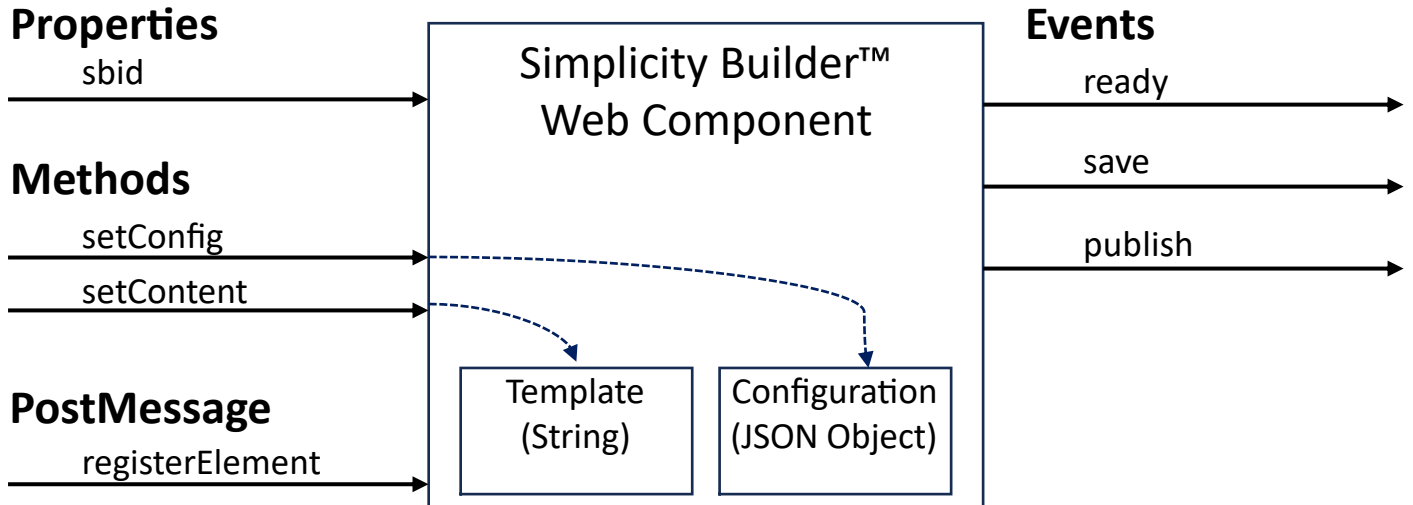
Common Concepts

Whether you use the Simplicity Builder™ web component in a Vanilla JavaScript, Angular, React or Vue project, the core concepts remain the same.

This section describes the core architecture and implementation requirements for getting the Simplicity Builder™ to work.



Core Architecture



Template

A template is a string that you load into the Simplicity Builder™ web component with the setContent method.

A template is a starting point that you load into the Simplicity Builder™ web component using the setContent method. It is a string that contains an HTML document with properties in the elements for which you want your end users to be able to edit, add content blocks, etc. The setContent method is also used to load saved editable content.

A template must also contain a special comment (<!--SB-TEMPLATE-END-->) in the line before the closing body tag. This is used as a marker for adding / removing design time injections made by the Simplicity Builder™ web component.



A template can also contain design-time only CSS and scripting that can be removed at publish time.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simplicity Builder&trade; Hello World Template</title>
  <style>
    * {
      margin: 0;
      padding:0;
      box-sizing: border-box;
    }
  </style>
</head>
<body>
  <header>
    Header Area
  </header>
  <main id="maincontent" data-sbblock="Main Content Area" data-sbzone="hi">

  </main>
  <footer>
    Footer Area
  </footer>
  <script src="##BASEADDRESS##/templates/template_helloworld/circleUtility.js"
id="helloworld_utility"></script>
  <!--SB-TEMPLATE-END-->
</body>
</html>
```

Data attributes added to an HTML element to make them editable by the Simplicity Builder™ web component.

THIS HTML COMMENT HAS TO BE IN YOUR TEMPLATE BEFORE THE CLOSING BODY TAG.

Template Block Properties

A template can be a starting page as well as a piece of a defined block that can be dragged into the editor. In the simple example above it is a basic HTML page where the only editable portion is the <main> element (i.e., in this case it is the only area in the web page that is allowed to have content added to it).

A template can contain as many or as few elements that you allow the user to modify.

The bare minimum to create an editable area of the template is to add the data-sbblock attribute.



The attributes you can add to a section and their purpose are:

data-sbblock

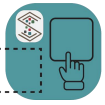
The data-sbblock attribute defines a section of the template or block that can be modified by the editor. Any element with this attribute will be registered in the editor and when a user hovers over it, it will display the text given in the attribute's value. So in the simple example above the `data-sbblock="Main Content Area"` attribute results in what is shown below when the user hovers over that area of the template.



data-sbzone

The optional data-sbzone attribute defines the areas (if any) another block could be dropped into this area. The attribute value can be any **one** of the following:

- **hi** (horizontal inside)
- **ho** (horizontal outside)
- **hb** (horizontal both)
- **vi** (vertical inside)
- **vo** (vertical outside)
- **vb** (vertical both)



As shown below the green drop zone indicator will show on the top part of the block or the bottom part (in the case of horizontal settings [hi, ho, hb]) of the block depending on where the user is dragging over.



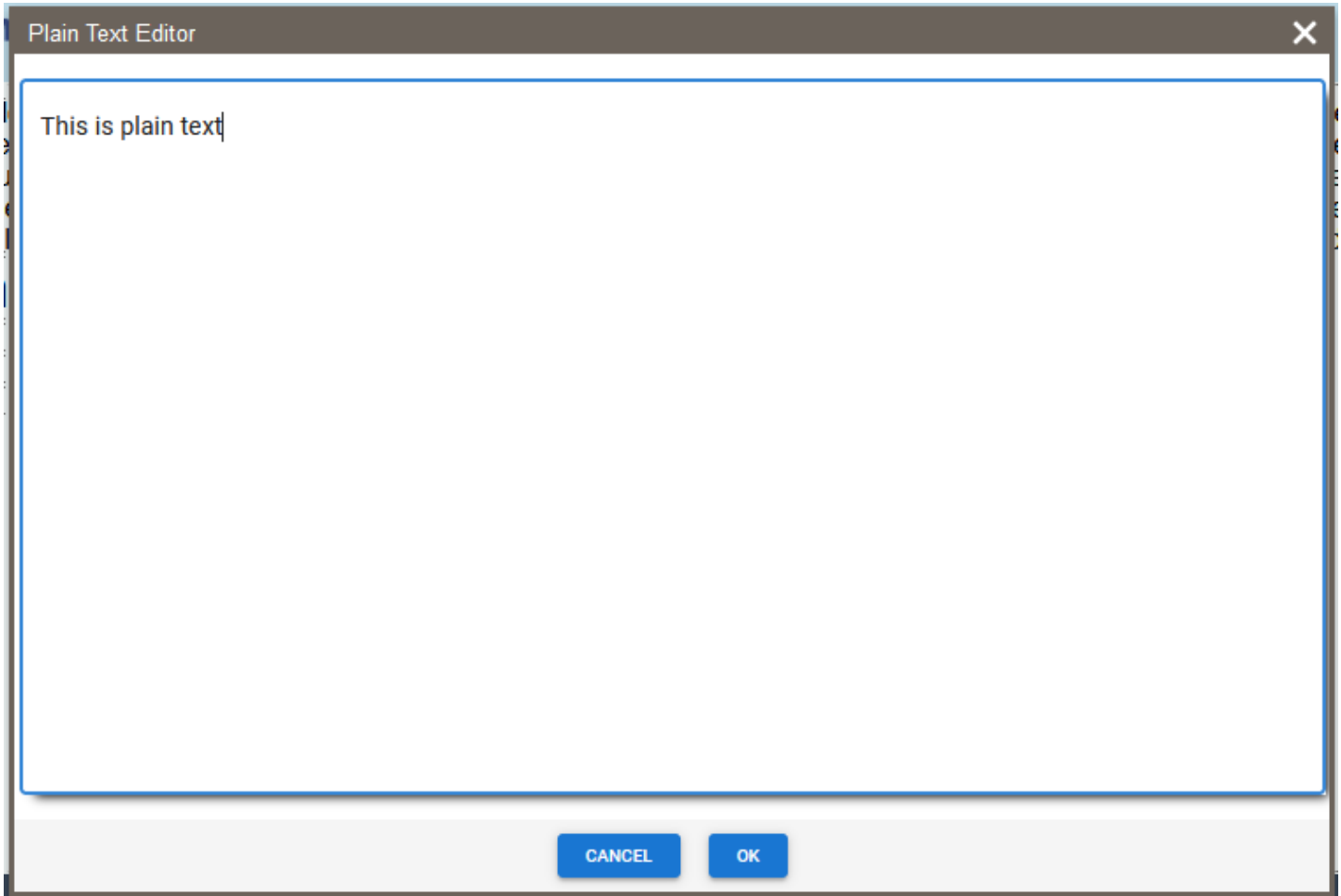
data-sbeditor

The optional `data-sbeditor` attribute defines which editor to use to edit the content (innerHTML or innerText) of this block. The value you enter for this attribute defines the editor that gets displayed when a user clicks on the edit symbol (🔍). The Simplicity Builder™ web component comes included with three different editors you can use:

- plaintext
- richtext (WYSIWYG Editor)
- code



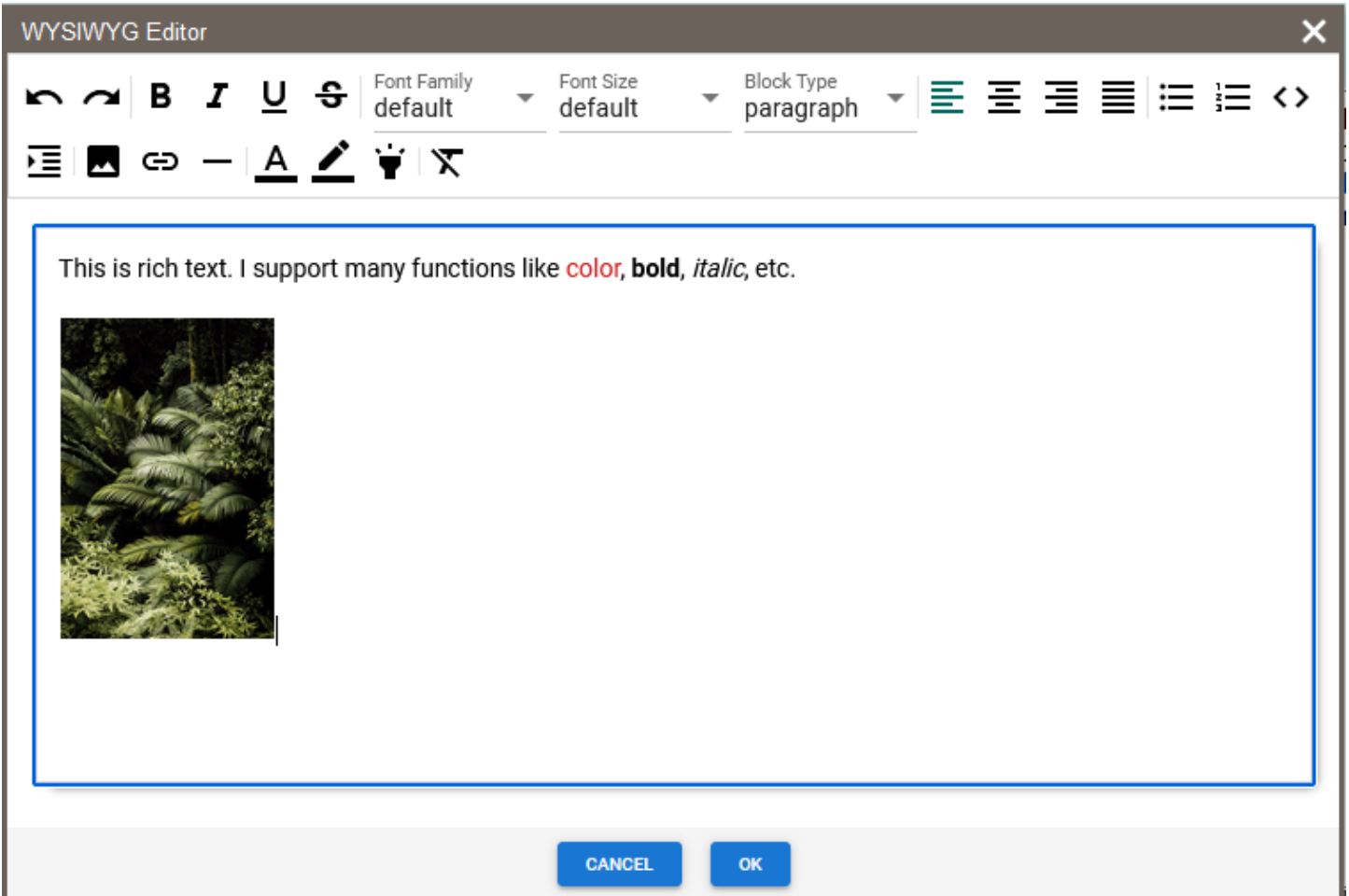
plaintext



The plain text editor lets the user edit text only of an element's content. It is a good editor choice for things like heading text where you do not want to give the user any formatting options.



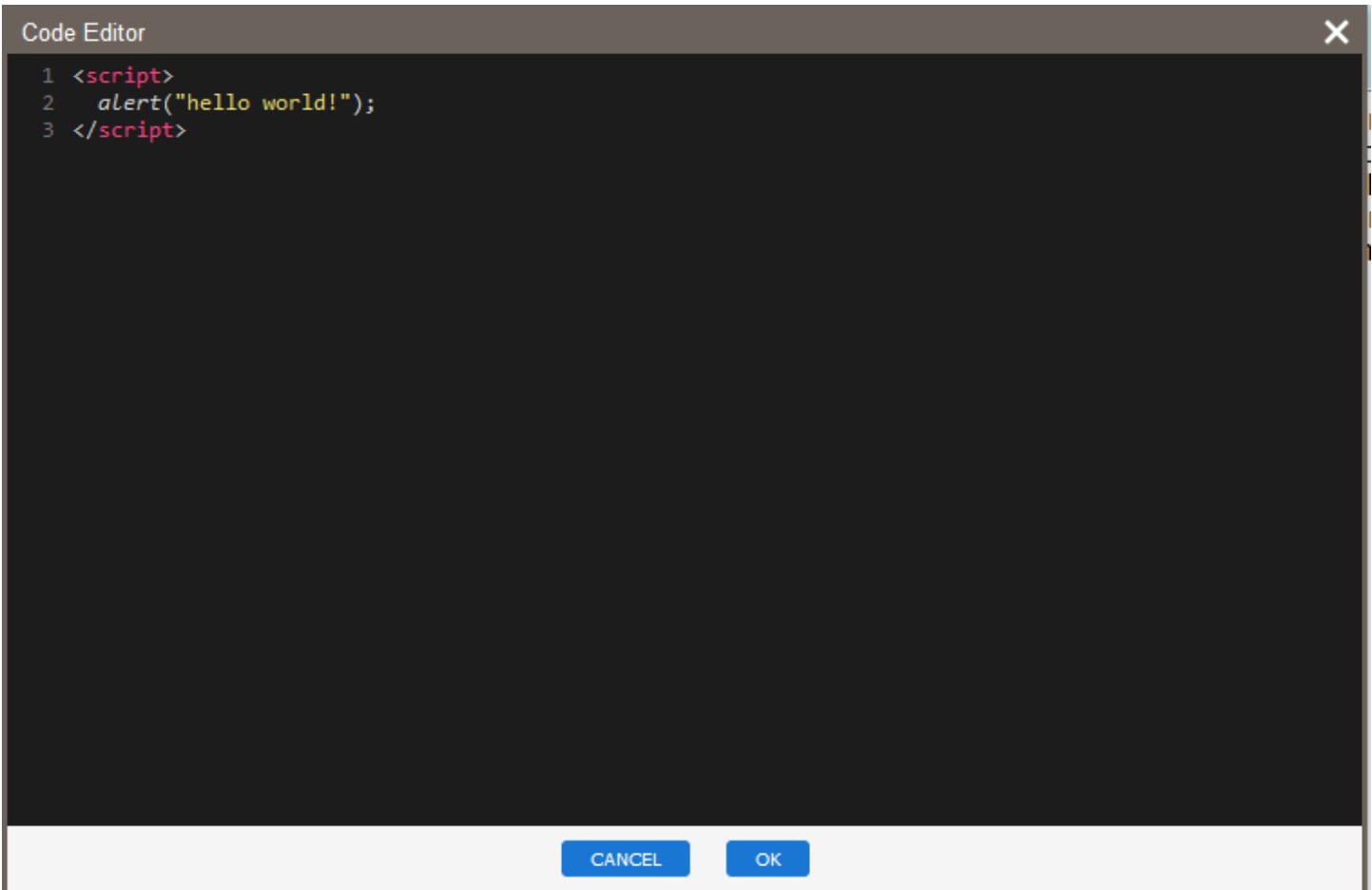
richtext



The rich text editor choice is a robust editor that lets the user format text in many ways such as color, highlight color, font selection, etc. It also allows for the insertion and formatting of images.




code




In the event that you would want to be able to let a user enter executable JavaScript code into a document, you can define a block with the code editor.




data-sbsettings

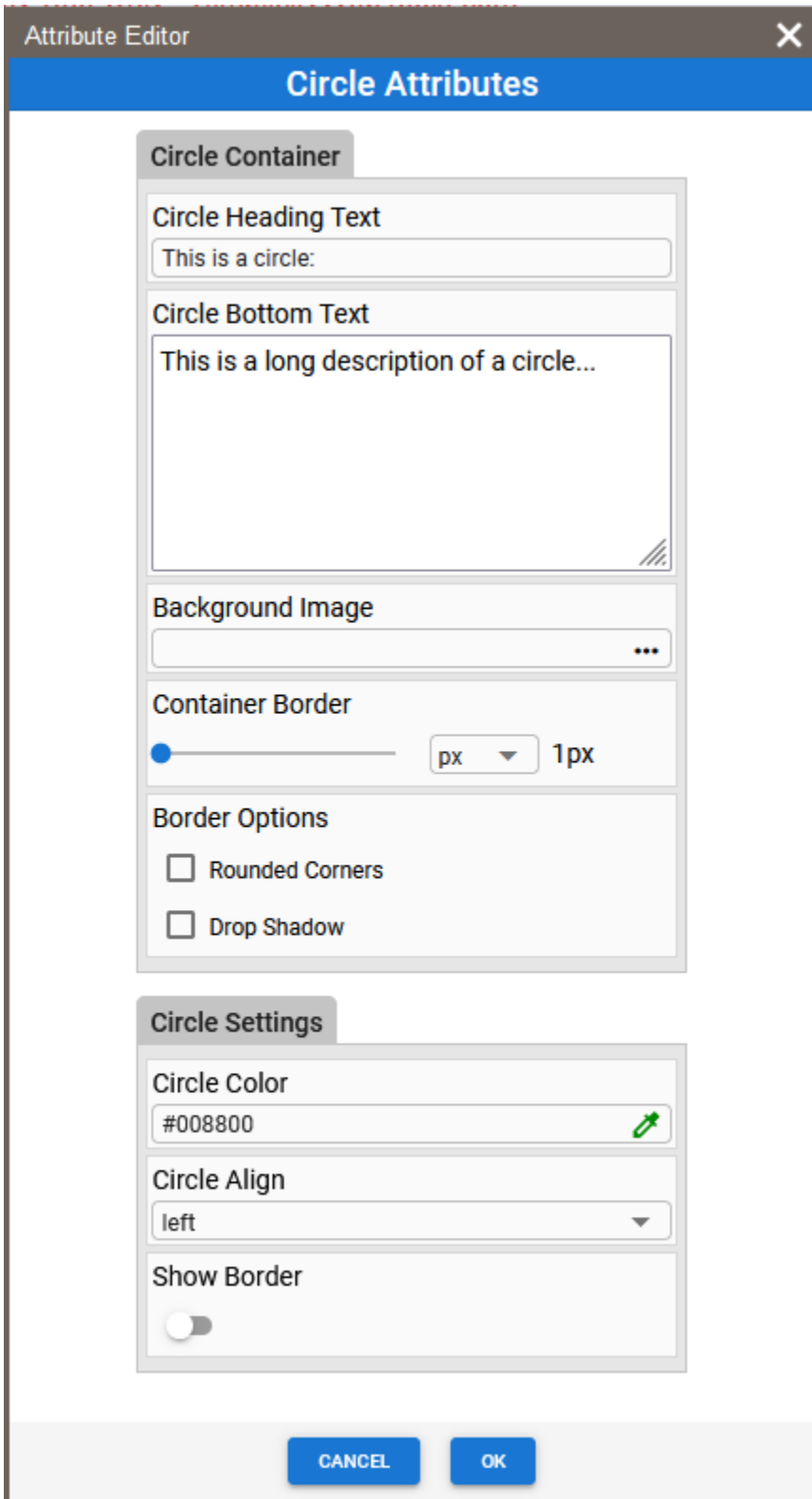
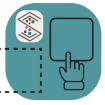
The `data-sbsettings` attribute indicates the ID of a custom attribute editor defined in the config file. In our 'Hello World' example we define a block that shows a circle in a div. A custom attribute editor is defined for this block that is designed to show all of the possible controls you can define in a custom attribute editor. If a custom attribute editor is defined for a block, the user will see a settings icon () that they can click that brings up the editor.

This is a circle:



This is a long description of a circle...





A later section describes the detail of defining an attribute editor within the config file.



Configuration Object

The Simplicity Builder™ web component requires a JSON configuration object to be loaded in order to operate. The object is loaded via the setConfig method.

If the configuration object is not loaded or is incorrectly configured, the Simplicity Builder™ will display the following:

```
SimplicityBuilder™ Waiting for Configuration...
```

IMPORTANT NOTES:

1. The setConfig method cannot be called until the Simplicity Builder™ web component has fired its ready event. You need to add an event listener to the Simplicity Builder™ web component's 'ready' event to set it. You can look at the code in the example apps done in various frameworks to see this.
2. The 'html' property of 'blocks' needs to be escaped with encodeURIComponent. This is necessary for edge cases where the HTML may contain special characters that the Simplicity Builder™ web component is expecting them to be escaped.

The JSON configuration object can get a bit unwieldy if you try to put everything in a single file so it is best to use JavaScript modules and export a method that returns an object. This allows you to break up things like the block definitions and attribute editor configurations in separate files and import them in. You will notice that our examples utilize this method.



The basic structure of the JSON configuration object is:

```
{
  "license": "XXXXXX",
  "devKey": "XXXXXX",
  "themeColor": "#7d5828",
  "themeDarkColor": "#101820",
  "assetsLocation": "https://www.mysitexyz.com/sbeditors",
  "helpUrl": "https://www.mysitexyz.com/helpDoc/index.html",
  "cleanupAttributes": ["data-attribute1", "data-attribute2"],
  "cleanupStrings": ["https://www.mysitexyz.com "],
  "cleanupIds": ["id1", "id2"],
  "blockSections": [{
    "heading": "Block Set 1",
    "blocks": [
      {
        "icon": "<img src=\"https://www.mysitexyz.com/file-text.svg\" />",
        "label": "Plain Text",
        "tooltipHeading": "Plain Text Block",
        "tooltipImage": "https://www.mysitexyz.com/plaintext-sample.png",
        "tooltipText": "Plain text.",
        "dragImage": " https://www.mysitexyz.com/plaintext-sample.png",
        "html": "HTML string that has been escaped with encodeURIComponent"
      }
    ]
  }
],
  "imageSelections": [
    {
      "url": "https://www.mysitexyz.com/editorimages/image1.jpg",
      "thumbnailUrl": "https://www.mysitexyz.com/editorimages/image1_tn.jpg"
    },
    {
      "url": "https://www.mysitexyz.com/editorimages/image2.jpg",
      "thumbnailUrl": "https://www.mysitexyz.com/editorimages/image2_tn.jpg"
    }
  ],
  "attributeEditors": [
    {
      "id": "ae1",
      "editorHeading": "Circle Attributes",
      "mode": "constant",
      "callback": "circleFunction",
      "controlSections": [
        {
          "heading": "Circle Container",
```




```
"type": "object",
  "target": "data-settingsobj",
  "controls": [
    {
      "heading": "Circle Heading Text",
      "type": "text",
      "mode": "text",
      "attribute": "displaytext",
      "value": "This is a circle:"
    },
  ],
}
]
```

Configuration Object Properties

license (string)

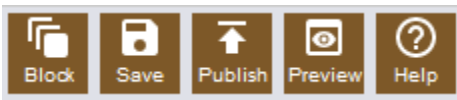
This value can be an empty string for development. The license key is only required for live deployments on domains other than localhost or 127.0.0.1. Deployment license keys can be purchased at <https://www.SimplicityWebTools.com>.

devKey (string)

This is either your free 30-day trial key or your purchased developer key. These keys can be obtained at <https://www.SimplicityWebTools.com>. Note that a free 30-day trial key is available without the need to enter any credit card information.

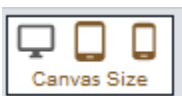
themeColor (string)

The themeColor string is any valid CSS color value (e.g., red, #FF0000, etc.). It sets the color of the buttons in the top UI:



themeDarkColor (string)

The themeDarkColor string is any valid CSS color value (e.g., red, #FF0000, etc.). It only sets the border color of the canvas size selector:





assetsLocation (string)

This is a string that contains the relative location of the Simplicity Builder™ assets folder. This is the folder that the plain text editor, rich text editor, code editor and attribute editor files are located.

The assets pack comes with an npm install or you can download it from the SimplicityWebTools.com website.

helpUrl (string)

This string contains the location of the HTML file that will provide the user with help information.

cleanupAttributes (array [] of strings)

A recommended way to communicate design-time changes between a block and the attribute editor and any design time code you include is through objects in custom data attributes. When you 'publish' the final output for live deployment you will most likely want to remove these attributes. Any attributes defined in the cleanupAttributes property will be removed from all elements in the document when the 'publish' method is executed.

cleanupStrings (array [] of strings)

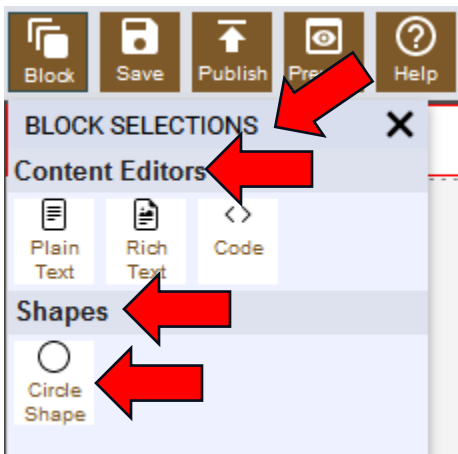
This property defines any string you want removed when a 'publish' is executed. The most common use for this is to remove the fully qualified domain from the page when published.

cleanupIds (array [] of strings)

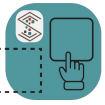
The cleanupIds property is used to list all IDs of elements that will be removed at 'publish' time. The most common use for this is to remove design-time scripts and design-time CSS.

blockSections (array [] of block section objects{ })

The block sections array and the associated nested blocks array define your predefined elements to drag and drop onto the canvas.

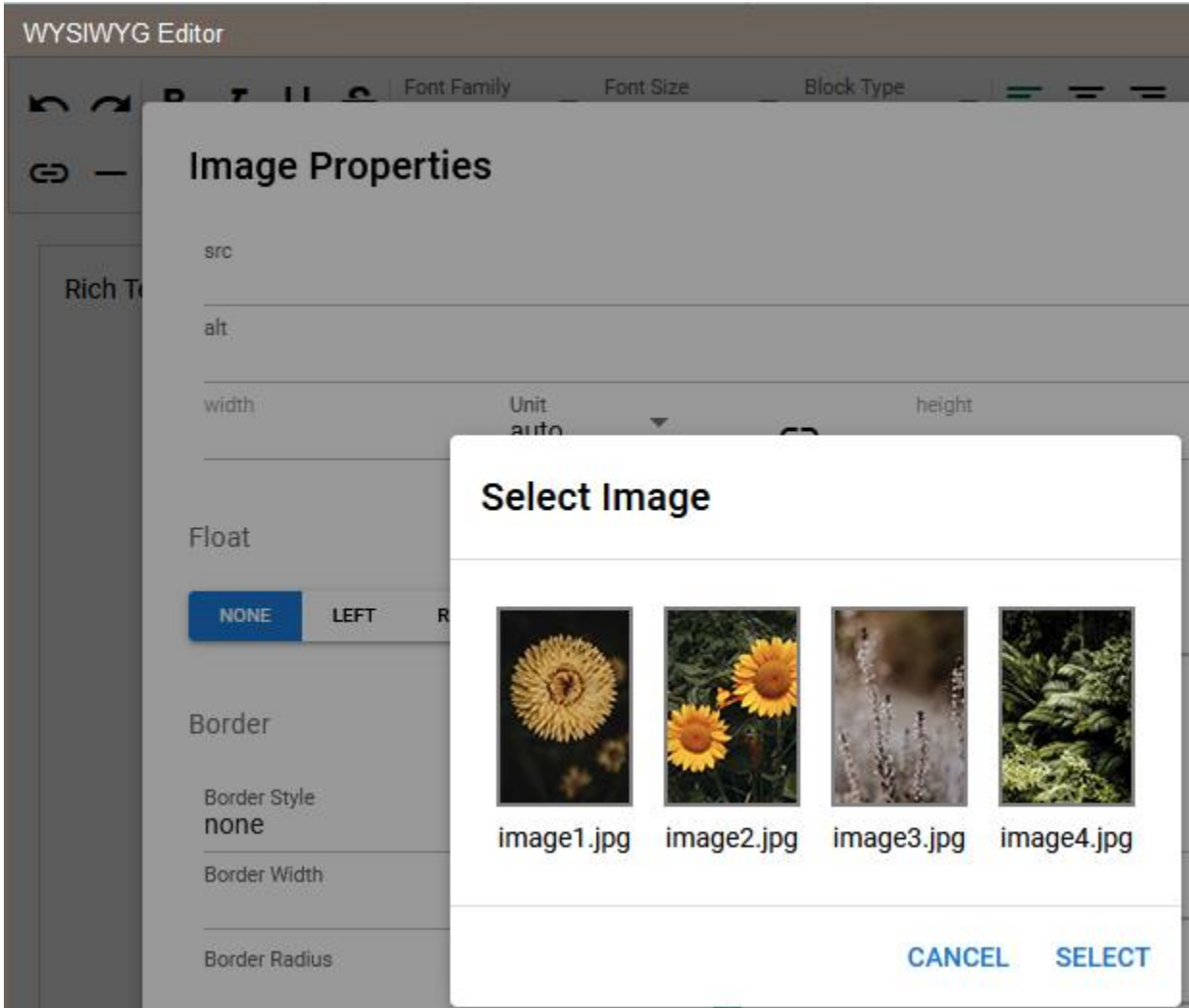


The 'Defining Blocks' section below provides the detail needed to define block sections and blocks.



imageSelections (array [] of image objects {})

The 'imageSelections' property holds a list of images intended specifically for use as the available images shown in the rich text editor.



The array is a list of objects with each object having a 'url' property that points to the full size image that will be placed in the document and a 'thumbnailUrl' property that is used for the image selection picker.

```
{  
  "url": "https://www.mysitexyz.com/editorimages/image1.jpg",  
  "thumbnailUrl": "https://www.mysitexyz.com/editorimages/image1_tn.jpg"  
},  
{
```



attributeEditors (array [] of Attribute Editor objects {})

The 'attributeEditors' property is a list of attribute editor objects that are referenced vi each attribute editor's 'id' property with the data-settings attribute in a block.

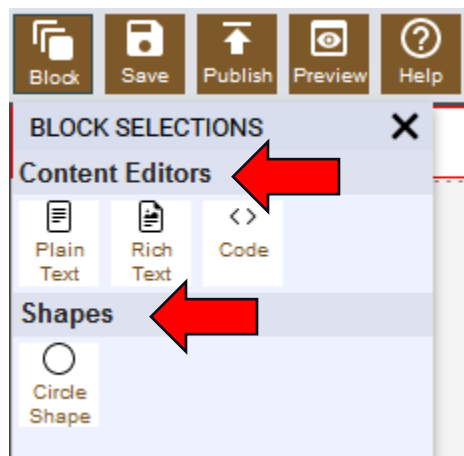
The 'Defining Attribute Editors' section below provides the detail needed to define attribute editors.

Defining Blocks

Defining blocks is at the heart of the Simplicity Builder™ web component. Blocks are predefined sections of HTML that can be drag and dropped onto the canvas.

Block Sections

Block sections are a way to group your blocks into categories that make sense to your user.



```
"blockSections": [
  {
    "heading": "Content Editors",
    "blocks": []
  },
  {
    "heading": "Shapes",
    "blocks": []
  }
],
```

The code above shows two block sections defined without any blocks defined yet in each.



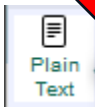
A simple sample block object is shown below. This block defines a div whose inner text can be edited with the plain text editor.

```
{
  "icon": "<img src=\"https://www.mysitexyz.com/templates/icons/file-text.svg\" />",
  "label": "Plain Text",
  "tooltipHeading": "Plain Text Block",
  "tooltipImage": "https://www.mysitexyz.com/templates/icons/plaintext-sample.png",
  "tooltipText": "Plain text editor that allows for text editing only. No HTML or
  formatting.",
  "dragImage": "https://www.mysitexyz.com/templates/icons/plaintext-sample.png",
  "html": encodeURIComponent(`<div data-sbblock="Plain Text" data-sbzone="hovo" data-
  sbeditor="plaintext" style="width: 100%;">Plain Text</div>`)
}
```

The following are the properties that define the block object:

icon

The 'icon' property is the full HTML (i.e., complete img tag or svg tag) of the icon that gets displayed for the block.

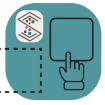


The icon should be in the range of 16px wide to approximately 40px wide.

label

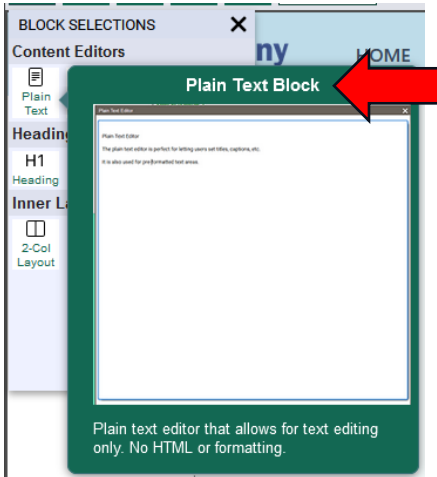
The 'label' property defines the text that is shown under the icon in the block.





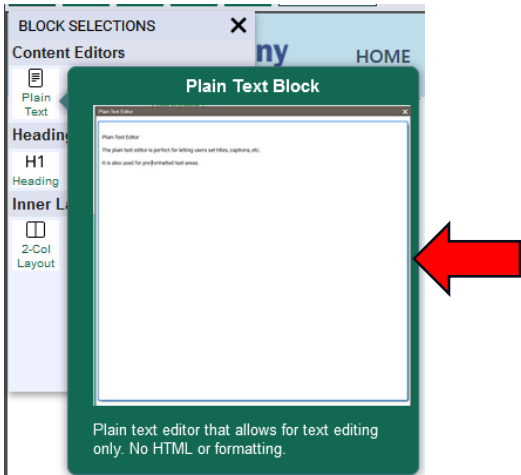
tooltipHeading

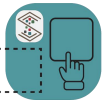
The tooltipHeading property defines the heading text that shows in the tooltip popup that shows when a user hovers over a block.



tooltipImage

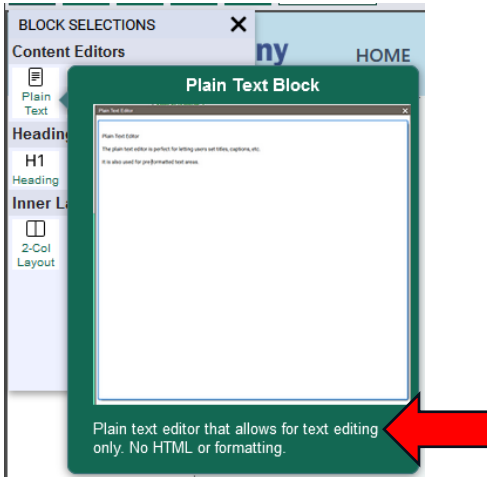
The tooltipImage property is the url pointing to the image that will show up in the tooltip popup that shows when a user hovers over a block.





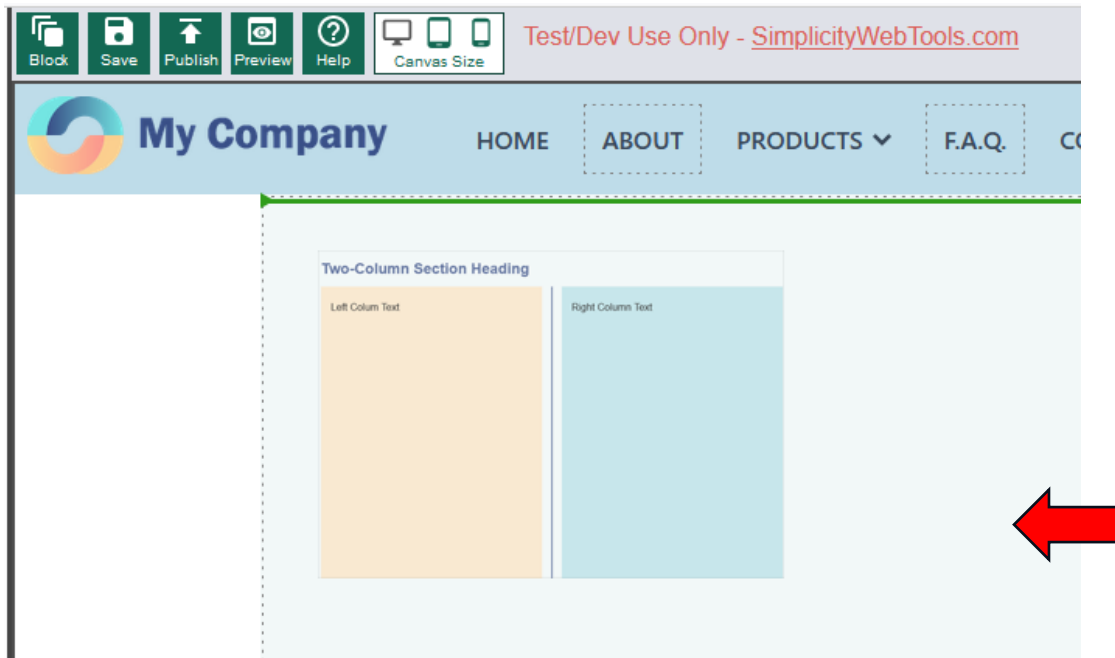
tooltipText

The tooltipText property is the text that will show up in the tooltip popup that shows when a user hovers over a block.



dragImage

The dragImage property is the url pointing to the image that will show up when the user is dragging a block onto the canvas. I typically use the same image that I show for the tooltip Image, but it can be any image you feel properly represents the block being dragged onto the canvas.





html

The html property is the html that will get inserted at the drop point on the canvas.

The string value assigned to the 'html' property must be escaped with JavaScript's encodeURIComponent function.

The html blocks can be any valid HTML. When a block gets dropped onto the canvas it will get "registered" with the Simplicity Builder™ web component so if there are any editable blocks defined they will be immediately available without any further action by you.

The following block object is the "circle" block that is part of our hello world example.

```
{
  "icon": "<img src=\"###BASEADDRESS##/sbassets/blocks/shapes_circle/circle.svg\" />",
  "label": "Circle Shape",
  "tooltipHeading": "Circle Shape",
  "tooltipImage": "###BASEADDRESS##/sbassets/blocks/shapes_circle/circle-sample.jpg",
  "tooltipText": "Simple circle shape for demonstration.",
  "dragImage": "###BASEADDRESS##/sbassets/blocks/shapes_circle/circle-sample.jpg",
  "html": encodeURIComponent(`<div data-sbblock="Circle Shape" data-sbzone="ho"
class="wrapper" data-sbsettings="ae1" data-settingsobj="{\"displaytext\":\"This is a circle:\",
\"background\":\"\", \"cborder\":\"1px\", \"bottomtext\":\"This is a long description of a circle...\",
\"options\":[]}\" data-circle="{\"align\":\"left\", \"circlecolor\":\"#008800\", \"border\":false}'>
  <h1>This is a circle:</h1>
  <div class="cwrap">
    <div class="circle borderoff"></div>
  </div>
  <p>This is a long description of a circle...</p>
</div>`)
}
```




Defining Attribute Editors

Each block defined in the Simplicity Builder™ config file can have a settings icon defined by placing the following data parameter in the HTML element: data-sbsettings="ae1"

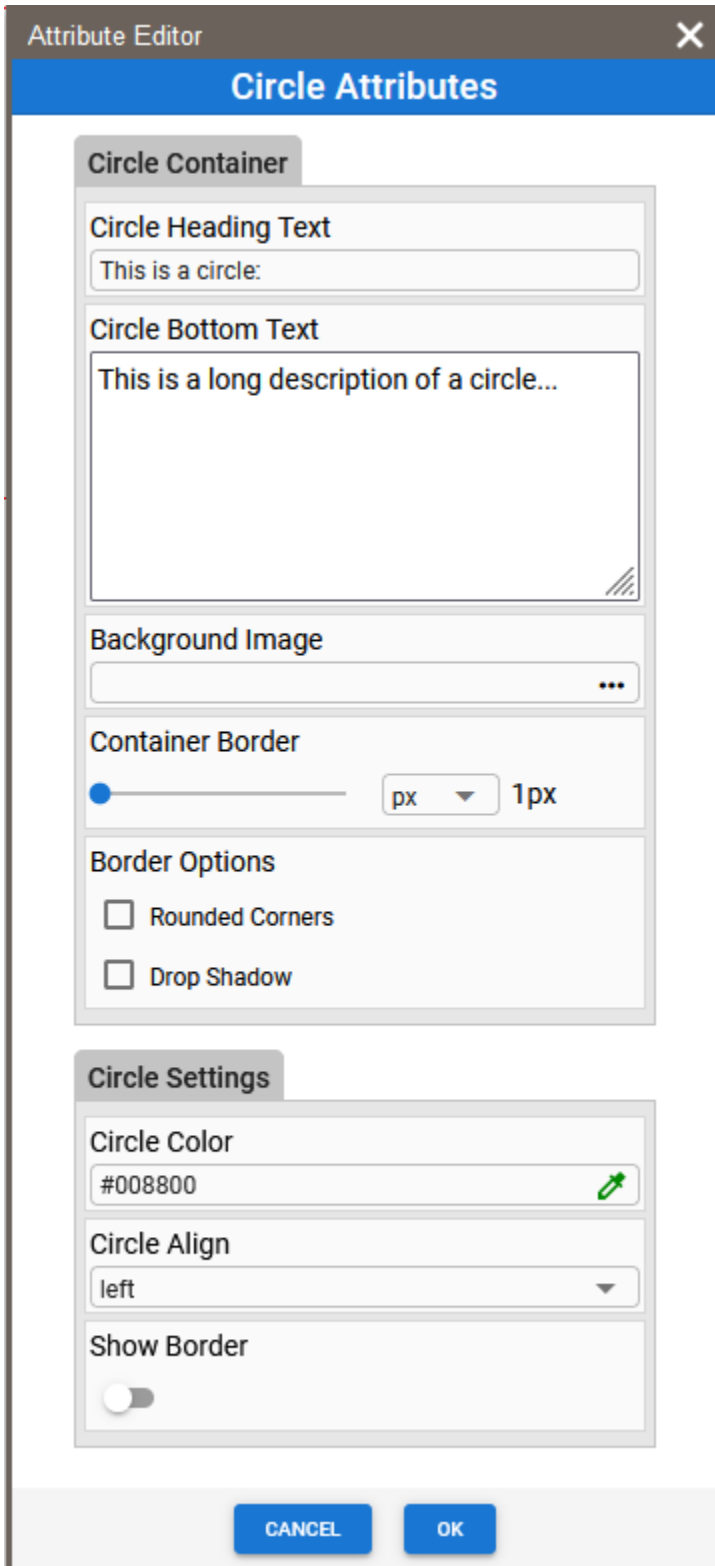
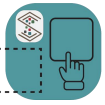
```
`<div data-sbblock="Circle Shape" data-sbzone="ho" class="wrapper" data-sbsettings="ae1" data-settingsobj="{\"displaytext\":\"This is a circle:\", \"background\": \"\", \"border\": \"1px\", \"bottomtext\": \"This is a long description of a circle...\", \"options\": [ ]}\" data-circle="{\"align\": \"left\", \"circlecolor\": \"#008800\", \"border\": false}\">  
  <h1>This is a circle:</h1>  
  <div class="cwrap">  
    <div class="circle borderoff"></div>  
  </div>  
  <p>This is a long description of a circle...</p>  
</div>`
```



In the above example the "ae1" defines the ID of an attribute editor defined in the config file. An attribute editor is a dynamic properties dialog that lets the user select a variety of settings that you define in the attribute editor setup(s).

You can use an attribute to allow your user to set / configure a block with as many or few options as you wish.

Using the circle block example again that is in our hello world example, the attribute editor defined for that is specifically designed to show off all the control types that can be used:



Control Types

The following control types can be defined in an attribute editor:



checkset

Border Options

- Rounded Corners
- Drop Shadow

color

Circle Color


#008800




image

Background Image


Select Image



CircleBackground1.jpg



CircleBackground2.jpg



CANCEL SELECT



select

Circle Align

slider

Container Border
 px 1px

text

Circle Heading Text

text (mode = textarea)

Circle Bottom Text

toggle

Show Border



Defining Attribute Editor Objects

Multiple attribute editors can be defined in a single configuration object. The `attributeEditors` property of the configuration object is an array of attribute editor objects. The sample below is a cut down version of the attribute editor defined for the circle block in our hello world example.

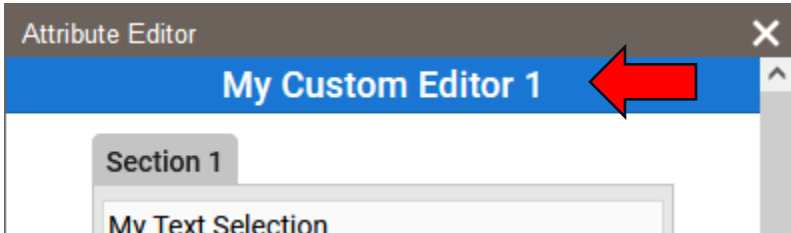
```
export default config;
"attributeEditors": [
  {
    "id": "ae1",
    "editorHeading": "Circle Attributes",
    "mode": "constant",
    "callback": "circleFunction",
    "controlSections": [
      {
        "heading": "Circle Settings",
        "type": "object",
        "target": "data-circle",
        "controls": [
          {
            "heading": "Circle Color",
            "type": "color",
            "attribute": "circlecolor",
            "value": "#008800"
          },
          {
            "heading": "Circle Align",
            "type": "select",
            "attribute": "align",
            "value": "",
            "selections": ["left", "center", "right"]
          },
          {
            "heading": "Show Border",
            "type": "toggle",
            "attribute": "border",
            "mode": "bool",
            "value": false
          }
        ]
      }
    ]
  }
]
```



id: defines the ID of the attribute editor that is referenced in the block " data-sbsettings" attribute (e.g., data-sbsettings="ae1").

editorHeading

The editorHeading property is a string that defines what shows up in the heading area of the attribute editor like below:



mode

The mode property is a string value that can be "constant" or "once". If set to constant the attribute editor will fire commands to the callback function as values change in any of the controls. If set to "once", the attribute editor will only fire the command to the callback function when the user clicks the 'OK' button.

callback

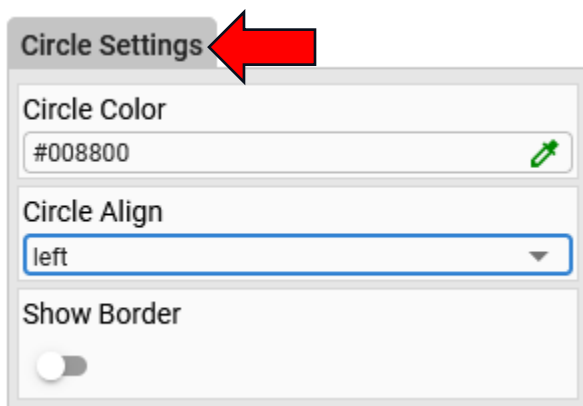
The callback property is a string that is the name of the function that gets called either as value change in any of the attribute editor controls ('constant' mode) or when the OK button is pressed ('once' mode).

controlSections

Sets of controls can be separated out into different sections. The controlSections array is where you define the section objects.

controlSections:heading

The heading property is a string that defines the text that shows up above the control section:





controlSections:type

The control section type property is a string that defines how the attribute editor interacts with the selected element / block. The value for this property can be either 'single' or 'object'.

single

If the 'type' property is set to single, the attribute defined in each control definition is a single attribute in the selected object that will have its value changed.

object

If the 'type' property is set to object, the attribute defined in each control definition is a property in the object value of the attribute named in the target property (below) that is in the selected object that will have its value changed.

In our hello world circle example, you will see the HTML for the opening div in the block is defined as follows:

```
<div
data-sbblock="Circle Shape"
data-sbzone="ho"
class="wrapper"
data-sbsettings="ae1"
data-settingsobj="{\"displaytext\":\"This is a circle:\",
\"background\":\"\",
\"cborder\":\"1px\",
\"bottomtext\":\"This is a long description of a circle...\",
\"options\":[]}"
data-circle="{\"align\":\"left\",
\"circlecolor\":\"#008800\",
\"border\":false}"
```

This has two custom data attributes that contain objects that are used as targets for attribute editor sections.

The attribute editor interacts with the target values. Your callback function needs to read these values and take appropriate actions.

In the hello world example there is a utility function that is brought into the template. Here is the code. This is the callback function that gets called as values change in the circl block attribute editor:

```
(function() {
  function init() {
    //You need to add a window message listener
    //This gets triggered by the Simplicity Builder(TM) Attribute Editor
    //The Attribute Editor sends the following object in the event data parameter
    // {
    //   command: "run",
```



```
// function: "callback Name", //this is the value you entered in the callback field
of the Attribute Editor setup in the config file
// id: abc123 //this is the id of the element that has the data-sbsettings data
attribute in it
// }
// depending on whether the attributeEditor's mode is set to "constant" or "once", this
gets triggered either whenever a parameter in the attribute editor changes or when the OK
button is clicked
window.addEventListener('message', (event) => {
  const getCommand = event.data.command;
  if (getCommand == "run") {
    switch(event.data.function) {
      case 'circleFunction':
        const main = document.getElementById(event.data.id);
        const obj = JSON.parse(main.getAttribute('data-settingsobj'));
        const obj_circle = JSON.parse(main.getAttribute('data-circle'));
        const circle_wrapper = main.querySelector(".cwrap");
        const textHolder = main.querySelector("h1");
        const circle = main.querySelector(".circle");
        const bottomText = main.querySelector("p");
        textHolder.innerText = obj["displaytext"];
        bottomText.innerText = obj["bottomtext"];
        circle_wrapper.classList.remove("left", "center", "right");
        circle_wrapper.classList.add(obj_circle["align"]);
        circle.style.backgroundColor = obj_circle["circlecolor"];
        circle.classList.remove("borderoff", "borderon");
        if (obj_circle["border"] == true) {
          circle.classList.add("borderon");
        } else {
          circle.classList.add("borderoff");
        }
        if (obj["background"] != "") {
          main.style.backgroundImage = "url('" + obj["background"] + "')";
        } else {
          main.style.backgroundImage = "unset";
        }
        main.style.borderWidth = obj["cborder"];
        main.classList.remove("rounded", "shadow");
        if (obj["options"].includes("rounded")) {
          main.classList.add("rounded");
        }
        if (obj["options"].includes("shadow")) {
          main.classList.add("shadow");
        }
        break;
    }
  }
}
```




```
    }
  });
}

if (document.readyState === "loading") {
  document.addEventListener("DOMContentLoaded", init);
} else {
  init();
}
})();
```

controlSections:target

The target is the name of the primary attribute that is targeted with values.

controlSections:controls

The controls property is an array of control objects that show up under a section heading. Each control type has some common properties to set (heading, type, attribute, value), but different control types may have some additional properties to set.

The following shows each control type and some sample code that shows how to define the control. reference the hello world example to see it in action.

checkboxset

Border Options

Rounded Corners

Drop Shadow


```
{
  "type": "checkboxset",
  "value": [],
  "selections": [
    {
      "label": "Rounded Corners",
      "value": "rounded",
      "color": "green"
    },
    {
      "label": "Drop Shadow",
      "value": "shadow",
      "color": "green"
    }
  ],
  "heading": "Border Options",
  "attribute": "options"
}
```



}

color

Circle Color


#008800 

```
{
  "heading": "Circle Color",
  "type": "color",
  "attribute": "circlecolor",
  "value": "#008800"
}
```


image

Background Image


Select Image



CircleBackground1.jpg



CircleBackground2.jpg



CANCEL SELECT

```
{
  "heading": "Background Image",
```



```

"type": "image",
"attribute": "background",
"value": "",
"selections": [
  {
    "url": "##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground1.jpg",
    "thumbnailUrl":
"##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground1_tn.jpg"
  },
  {
    "url": "##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground2.jpg",
    "thumbnailUrl":
"##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground2_tn.jpg"
  },
  {
    "url": "##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground3.jpg",
    "thumbnailUrl":
"##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground3_tn.jpg"
  },
  {
    "url": "##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground4.jpg",
    "thumbnailUrl":
"##BASEADDRESS##/sbassets/blocks/shapes_circle/CircleBackground4_tn.jpg"
  }
]
}

```

select

```

{
  "heading": "Circle Align",
  "type": "select",
  "attribute": "align",
  "value": "",
  "selections": ["left", "center", "right"]
}

```

slider



```
{  
  "heading": "Container Border",  
  "type": "slider",  
  "attribute": "cborder",  
  "value": 1,  
  "uom": "px",  
  "min": 1,  
  "max": 15,  
  "selections": [  
    "px",  
    "pt"  
  ]  
}
```

text (mode = text)

Circle Heading Text
This is a circle:

```
{  
  "heading": "Circle Heading Text",  
  "type": "text",  
  "mode": "text",  
  "attribute": "displaytext",  
  "value": "This is a circle:"  
}
```

text (mode = textarea)

Circle Bottom Text
This is a long description of a circle...

```
{  
  "heading": "Circle Bottom Text",  
  "type": "text",  
  "mode": "textarea",  
  "attribute": "bottomtext",  
  "value": "This is a long description of a circle..."  
}
```



toggle

Show Border

```
{  
  "heading": "Show Border",  
  "type": "toggle",  
  "attribute": "border",  
  "mode": "bool",  
  "value": false  
}
```



Vanilla JavaScript Example

Before starting, obtain a trial dev key (available for free without a credit card) or a dev license from <https://www.SimplicityWebTools.com>.

The Easiest Way to Start

The instructions that follow are just a guideline for "start from scratch" projects. To get familiar with Simplicity Builder™ we recommend cloning or downloading the example project and studying the code in there.

To get the Vanilla JavaScript example project go to:

<https://github.com/mfoitzik/simplicity-builder-vanillajs-example>

You can clone the repo by doing the following:

```
git clone https://github.com/mfoitzik/simplicity-builder-vanillajs-example  
open index.html with Live Server or your dev server of choice
```

Starting From Scratch

This example is intended to provide some basic guidance to using the Simplicity Builder™ web component in a Vanilla JavaScript project. We walk you through the steps needed to do a basic implementation.

If you are starting a project from scratch the Simplicity Builder™ web component can be obtained from NPM or CDN (*please see note below about asset pack*):

For vanilla JavaScript:

via NPM

```
npm install @simplicitywebtools/simplicity-builder
```

via CDN (a script and a CSS file both need to be referenced)

```
<script type="module"  
src="https://unpkg.com/@simplicitywebtools/simplicity-  
builder@1.1.3/dist/simplicity-builder/simplicity-  
builder.esm.js"></script>  
  
<link rel="stylesheet"  
href="https://unpkg.com/@simplicitywebtools/simplicity-  
builder@1.1.3/dist/simplicity-builder/simplicity-builder.css">
```

Note about asset pack: If you install via NPM, there will be a folder named 'assets' in the node_modules\@simplicitywebtools\simplicity-builder\dist\simplicity-builder folder. Copy the entire 'assets' folder into your project source area. You will need to reference the location in the configuration file.



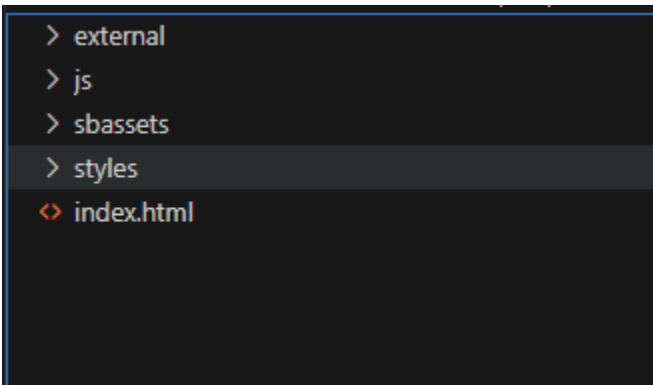
For this Vanilla JavaScript example (as well as all other examples in this document) we will be building an interface that has a few links in the left column as well as a text box for illustrating the 'save' and 'publish' outputs of the component.

We have some recommended folder structures regarding where to place assets, templates, blocks, etc., but the structure is entirely up to you.

Let's start with the most simple example that brings the editor to life, but with no blocks defined or template loaded yet. This example uses the script and CSS from the CDN.

In the editor of your choice (we use Visual Studio Code) open your project directory and create the following in the root folder:

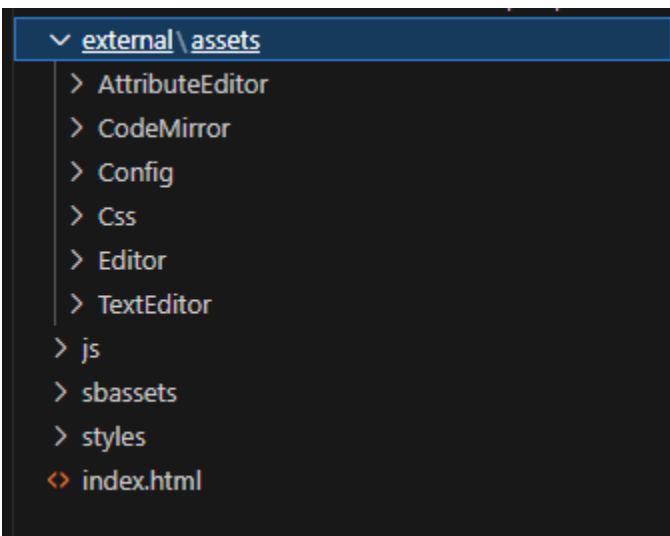
- index.html file
- a folder named external
- a folder named js
- a folder named sbassets
- a folder named styles



You will need to obtain the assets contents from our website at <https://www.simplicitytools.com/builderassets>

The assets contain specialty editors for plain text, rich text, code and custom attributes. There is also some code that gets injected into templates at design time.

Download the file, unzip it and copy the entire assets folder into the external folder so it looks like this:



In the index.html file create a basic HTML5 file. (in Visual Studio Code if you have emmet abbreviations enabled just enter `![tab]` inside the file editor. Change the title if you wish.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simplicity Builder™ Example</title>
</head>
```




```
<body>  
  
</body>  
</html>
```

Add references to js/main.js, styles/main.css and the Simplicity Builder™ JavaScript and CSS files from the content distribution network.

index.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Simplicity Builder™ Example</title>  
  <link href="styles/main.css" rel="stylesheet" type="text/css" />  
</head>  
<body>
```

```
  <script type="module" src="https://unpkg.com/@simplicitywebtools/simplicity-  
builder@1.1.3/dist/simplicity-builder/simplicity-builder.esm.js"></script>  
  <link rel="stylesheet" href="https://unpkg.com/@simplicitywebtools/simplicity-  
builder@1.1.3/dist/simplicity-builder/simplicity-builder.css">  
  <script type="module" src="js/main.js"></script>
```

```
</body>  
</html>
```

Add the HTML for the interface. We will build a simple two panel interface where the left panel has some links to load a few different templates as well as a textbox to view the output of the Simplicity Builder™ component.

index.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>
```

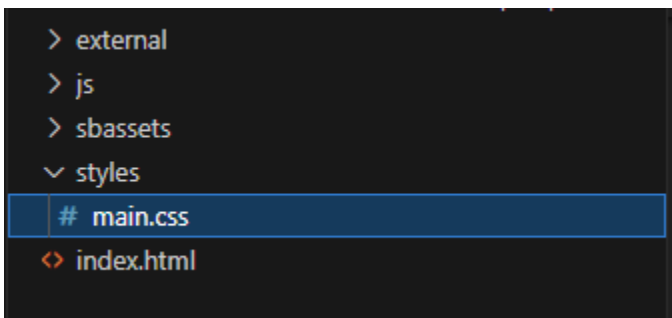


```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Simplicity Builder&trade; Example</title>
<link href="styles/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
```

```
<div class="wrapper">
  <div class="left">
    <h1>Simplicity Builder&trade; Vanilla JavaScript Example</h1>
    <p><a href="#" id="helloworldExample">Hello World Example</a></p>
    <p><a href="#" id="basicExample">Basic Example</a></p>
    <p><a href="#" id="bootstrapExample">Bootstrap Example</a></p>
    <label for="output">Save / Publish Output:</label>
    <textarea id="output"></textarea>
  </div>
  <div class="right">
    <simplicity-builder id="sbuilder"></simplicity-builder>
  </div>
</div>
```

```
<script type="module" src="https://unpkg.com/@simplicitywebtools/simplicity-
builder@1.1.3/dist/simplicity-builder/simplicity-builder.esm.js"></script>
<link rel="stylesheet" href="https://unpkg.com/@simplicitywebtools/simplicity-
builder@1.1.3/dist/simplicity-builder/simplicity-builder.css">
<script type="module" src="js/main.js"></script>
</body>
</html>
```

In the /styles folder create a file named main.css





Inside the main.css file add the CSS code to provide styling for the interface:

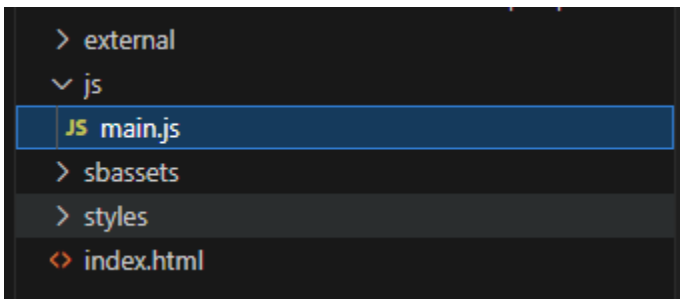
/styles/main.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body {
  width: 100vw;
}
.wrapper {
  display: grid;
  grid-template-columns: 25% 75%;
  margin: 0;
  overflow: hidden;
  height: 100vh;
}
.left {
  background-color: #fdfdfd;
  border-right: 3px solid #444444;
  grid-column-start: 1;
  grid-column-end: 2;
  padding: 10px;
  display: flex;
  flex-direction: column;
}
.right {
  grid-column-start: 2;
  grid-column-end: 3;
}
.h1, p {
  margin-bottom: 8px;
}
```



```
p {
  margin-left:15px;
}
textarea {
  width:100%;
  height:300px;
  scroll-behavior: auto;
  white-space: pre;
  flex:1;
}
label {
  font-weight:bold;
  display: inline-block;
  margin-bottom:4px;
}
```

In the /js folder create a file named main.js



Inside the /js/main.js file add the code to select and load templates, initialize the Simplicity Builder™ web component.



Vue Example

Before starting, obtain a trial dev key (available for free without a credit card) or a dev license from <https://www.SimplicityWebTools.com>.

The Easiest Way to Start

The fastest way to get to know how to use the Simplicity Builder™ component in Vuejs is to use our simplified example application for Vuejs. It can be downloaded or cloned from github:

<https://github.com/mfoitzik/simplicity-builder-vuejs-example>

Open the project and:

```
npm install  
npm run dev
```

Starting From Scratch

Start a new Vite-powered Vuejs project

```
npm create vue@latest  
Need to install the following packages:  
  create-vue@3.9.3  
Ok to proceed? (y)  
  
Vue.js - The Progressive JavaScript Framework  
√ Project name: ... simplicity-builder-vuejs-example  
√ Add TypeScript? ... No / Yes  
√ Add JSX Support? ... No / Yes  
√ Add Vue Router for Single Page Application development? ... No / Yes  
√ Add Pinia for state management? ... No / Yes  
√ Add Vitest for Unit Testing? ... No / Yes  
√ Add an End-to-End Testing Solution? » No  
√ Add ESLint for code quality? ... No / Yes  
Scaffolding project in D:\simplicity-builder-vuejs-example...  
Done. Now run:  
  cd simplicity-builder-vuejs-example  
  npm install  
  npm run dev
```

Install the Simplicity Builder™ component:

```
npm install @simplicitywebtools/simplicity-builder-vue
```



In the 'public' folder create a folder named 'sbassets'. Copy the folders and contents from 'node_modules/@simplicitywebtools/simplicity-builder/dist/simplicity-builder/assets' into the 'public/sbassets' folder you just created.

```
node_modules
├── .bin
├── @babel
├── @esbuild
├── @isaacs
├── @jridgewell
├── @one-ini
├── @pkgjs
├── @rollup
├── @simplicitywebtools
│   └── simplicity-builder
│       ├── dist
│       │   ├── cjs
│       │   ├── collection
│       │   ├── components
│       │   └── esm
│       └── simplicity-builder
│           └── assets
│               ├── AttributeEditor
│               ├── CodeMirror
│               ├── Config
│               ├── Css
│               ├── Editor
│               ├── testjs
│               └── TextEditor
```

```
public
├── sbassets
│   ├── AttributeEditor
│   ├── CodeMirror
│   ├── Config
│   ├── Css
│   ├── Editor
│   └── TextEditor
```



Copy the example templates pack (which includes block definitions, images, icons, etc.) and a sample config pack from <https://SimplicityWebTools.com/Downloads>

In the project's public folder create two subfolders: 'config' and templates. Unzip the downloaded sample files into their respective folder.

In the main.ts file import the css file (@simplicitywebtools/simplicity-builder/dist/simplicity-builder/simplicity-builder.css) needed by the Simplicity Builder™ component:

/src/main.ts

```
import './assets/main.css'
import '@simplicitywebtools/simplicity-builder/dist/simplicity-builder/simplicity-builder.css'
import { createApp } from 'vue'
import App from './App.vue'

createApp(App).mount('#app')
```

Copy the following into the main.css file:

/src/assets/main.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body {
  width: 100vw;
}
.wrapper {
  display: grid;
  grid-template-columns: 25% 75%;
  margin: 0;
  overflow: hidden;
  height: 100vh;
}
.left {
  background-color: #fdfdfd;
  border-right: 3px solid #444444;
  grid-column-start: 1;
  grid-column-end: 2;
  padding: 10px;
```



```
display: flex;
flex-direction: column;
}
.right {
  grid-column-start: 2;
  grid-column-end: 3;
}
.h1, p {
  margin-bottom: 8px;
}
p {
  margin-left: 15px;
}
textarea {
  width: 100%;
  height: 300px;
  scroll-behavior: auto;
  white-space: pre;
  flex: 1;
}
label {
  font-weight: bold;
  display: inline-block;
  margin-bottom: 4px;
}
```

Replace the contents of the `src/App.vue` file with the following:

`/src/App.vue`

```
<script setup lang="ts">
import { ref } from 'vue'
import {SimplicityBuilder} from '@simplicitywebtools/simplicity-builder-vue/lib'
// @ts-ignore
import helloWorldConfig from "../public/config/config_helloworld.js";
// @ts-ignore
import basicConfig from "../public/config/config_basic.js";
// @ts-ignore
import template_helloworld from
"../public/templates/template_helloworld/template_helloworld.js";
// @ts-ignore
import template_basic from "../public/templates/template_basic/template_basic.js";
const currentHost = window.location.protocol + "://" + window.location.host;
const sbRef = ref<HTMLSimplicityBuilderElement | null>(null);
const output = ref<string>("");
```




```
function sbReady() {
  setHelloWorld(null);
}
function setHelloWorld(event: Event | null) {
  event?.preventDefault();
  let config = JSON.stringify(helloWorldConfig);
  const re = /##BASEADDRESS##/g;
  sbRef.value?.$el.setConfig(JSON.parse(config.replace(re, currentHost)));
  sbRef.value?.$el.setContent(template_helloworld.replace(re, currentHost));
}
function setBasic(event: Event) {
  event.preventDefault();
  let config = JSON.stringify(basicConfig);
  const re = /##BASEADDRESS##/g;
  sbRef.value?.$el.setConfig(JSON.parse(config.replace(re, currentHost)));
  sbRef.value?.$el.setContent(template_basic.replace(re, currentHost));
}
function sbSave(event: CustomEvent) {
  output.value = event.detail;
}
function sbPublish(event: CustomEvent) {
  output.value = event.detail;
}
</script>

<template>
  <div class="wrapper">
    <div class="left">
      <h1>Simplicity Builder&trade; Example</h1>
      <p><a href="" @click="setHelloWorld">Hello World Example</a></p>
      <p><a href="" @click="setBasic">Basic Example</a></p>
      <label>Save / Publish Output:</label>
      <textarea v-model="output"></textarea>
    </div>
    <div class="right">
      <simplicity-builder ref="sbRef" sbid="unique1" @ready="sbReady" @save="sbSave"
@publish="sbPublish"></simplicity-builder>
    </div>
  </div>
</template>

<style scoped>

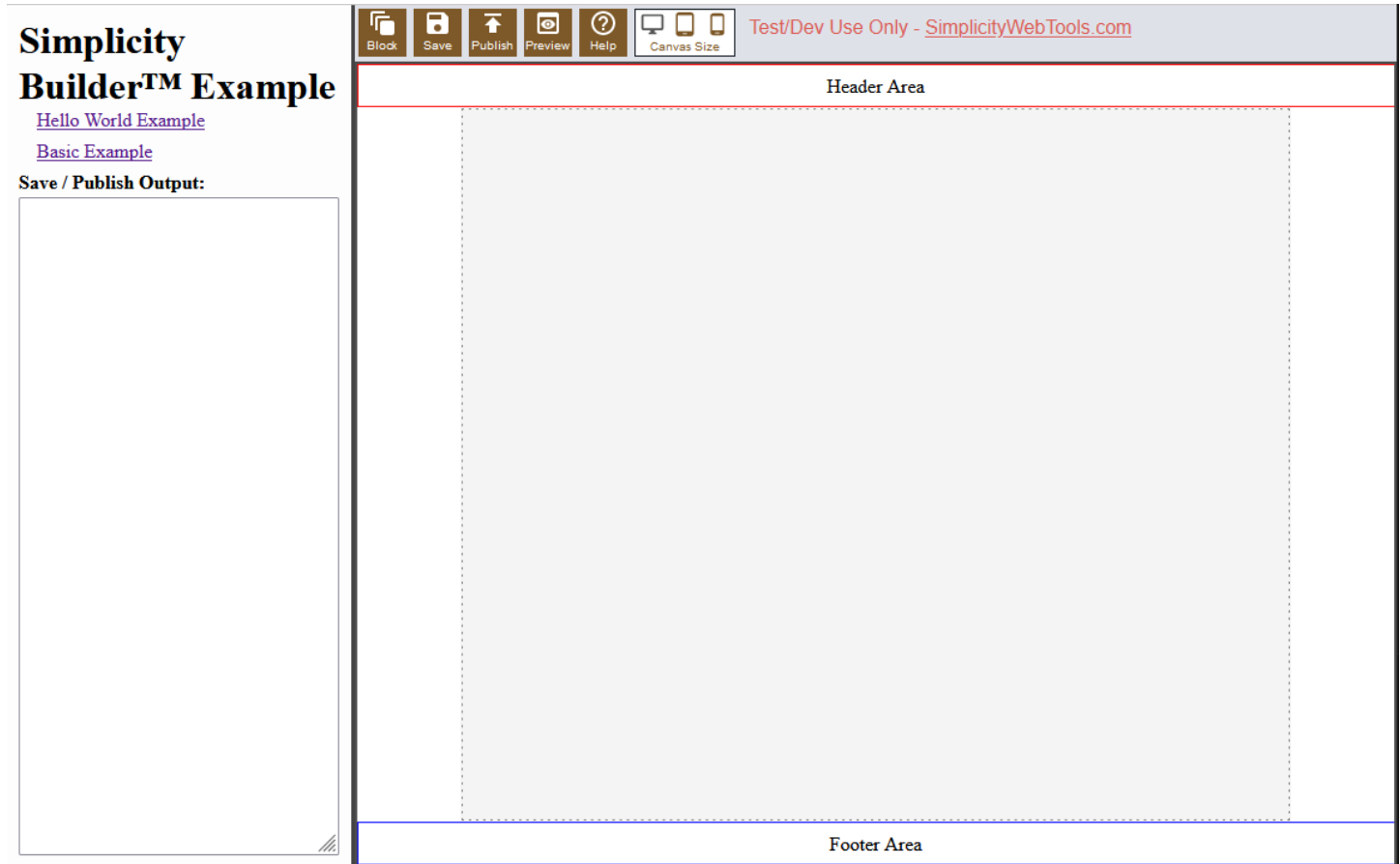
</style>
```



At this point you should be able to run:

```
npm run dev
```

You should see:



The objective of the example app is to show the basic concepts of the Simplicity Builder™ component. The "Hello World" example shows the use of the three built in content editors (plain text, rich text and code) as well as the built-in dynamic attributes editor with the "circle block".

We use a common set of JavaScript files for templates and config files that we use across all of our example apps (Vanilla JavaScript, Angular, React and Vue). That is why you will see '// @ts-ignore' above some lines.

Because different dev environments may have different port numbers assigned to their dev server, we add some code to get the current host at runtime and then do some text replaces to get the correct references in the templates and config files.



It is important to note that we have to add an event handler for the 'ready' event. This is necessary because we cannot initialize the component with a config file and content template until the component is ready.

In simple terms there are two methods to set the configuration and the content:

setConfig – accepts a JSON object with configuration information

setContent – accepts a string with a template

In the other direction there are two events for getting content out of the Simplicity Builder™ component:

save – when a user clicks the save button in the editor interface this event is triggered and the event object contains a property named 'detail' that contains the HTML including all of the attributes and design time code needed to re-open in the Simplicity Builder™ component to continue editing.

publish – when a user clicks the 'publish' button this event is triggered and the event object contains a property named 'detail' that contains the cleaned code that is meant for live deployment. It removes design time attributes as well as other items that are defined in the config file. For example, the config file can contain a list of attributes, ids and strings that will be removed as part of the published output.

If you look at the support code in the 'basic example' there is some code within the template that supports changing the header type (h1, h2, etc.). To get this to work you actually have to remove an object from the DOM and then replace it with another. There is a command you can send to the Simplicity Builder™ component that you need to run to "re-register" the component in the builder so it contains the proper menu, etc. for continued editing.

This is a way to "re-register" a DOM element:

```
window.parent.postMessage(  
{  
  command: 'registerElement',  
  content: event.data.id,  
},  
'*'  
);
```

The object you pass through a postMessage contains two properties:

command – must be 'registerElement'

content – a string that contains the id of the element you are registering.





React Example

Before starting, obtain a trial dev key (available for free without a credit card) or a dev license from <https://www.SimplicityWebTools.com>.

The Easiest Way to Start

The fastest way to get to know how to use the Simplicity Builder™ component in React is to use our simplified example application for React. It can be downloaded or cloned from github:

<https://github.com/mfoitzik/simplicity-builder-react-example>

In the public/config folder edit both config_basic.js and config_helloworld.js and replace the devkey value with the trial key or dev key you obtained from SimplicityWebTools.com.

```
const config = {  
  "helpUrl": "##BASEADDRESS##/helpDoc/index.html",  
  "assetsLocation": "##BASEADDRESS##/sbeditors",  
  "license": "",  
  "devKey": "XXXXXX",
```

Open the project and:

```
npm install  
npm run dev
```



Angular Example (standalone)

Before starting, obtain a trial dev key (available for free without a credit card) or a dev license from <https://www.SimplicityWebTools.com>.

The Angular examples require the Angular cli to be installed on your system:

```
npm install -g @angular/cli
```

Clone the repo:

```
git clone https://github.com/mfoitzik/simplicity-builder-angular-standalone-example.git
```

In the src/assets/config folder edit both config_basic.js and config_helloworld.js and replace the devkey value with the trial key or dev key you obtained from SimplicityWebTools.com.

```
const config = {  
  "helpUrl": "##BASEADDRESS##/helpDoc/index.html",  
  "assetsLocation": "##BASEADDRESS##/sbeditors",  
  "license": "",  
  "devKey": "XXXXXX",
```

```
npm install
```

```
ng serve --open
```



Angular Example (module)

Before starting, obtain a trial dev key (available for free without a credit card) or a dev license from <https://www.SimplicityWebTools.com>.

The Angular examples require the Angular cli to be installed on your system:

```
npm install -g @angular/cli
```

Clone the repo:

```
git clone https://github.com/mfoitzik/simplicity-builder-angular-module-example.git
```

In the src/assets/config folder edit both config_basic.js and config_helloworld.js and replace the devkey value with the trial key or dev key you obtained from SimplicityWebTools.com.

```
const config = {  
  "helpUrl": "##BASEADDRESS##/helpDoc/index.html",  
  "assetsLocation": "##BASEADDRESS##/sbeditors",  
  "license": "",  
  "devKey": "XXXXXX",
```

```
npm install
```

```
ng serve --open
```