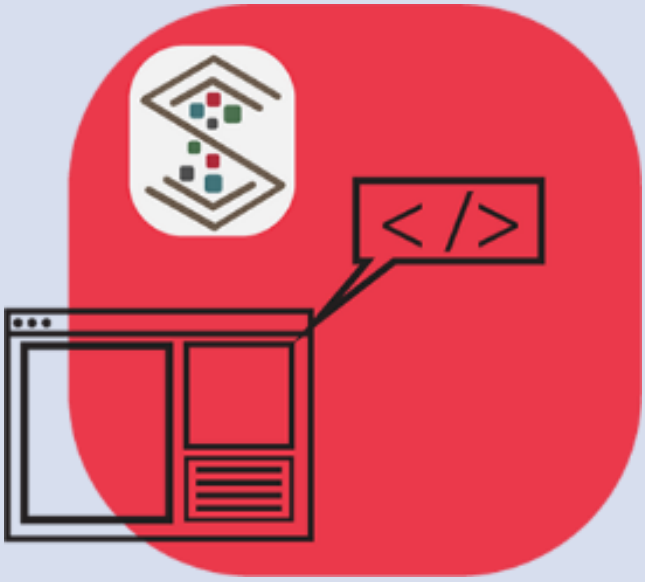
Simplicity™ Web Tools

www.SimplicityWebTools.com



Simplicity Embed™

Reference Manual

Version 1.0
March 11, 2024

Contents

Overview

Simplicity Embed™ is a robust web component that lets you easily create universally embeddable web widgets with any front-end and/or backend technology that you work with. The component solves the problem of integration, isolation, and communication.

For example, a mini application built in React can be distributed to work in any web page or even another application built in another framework like Vue or Angular.

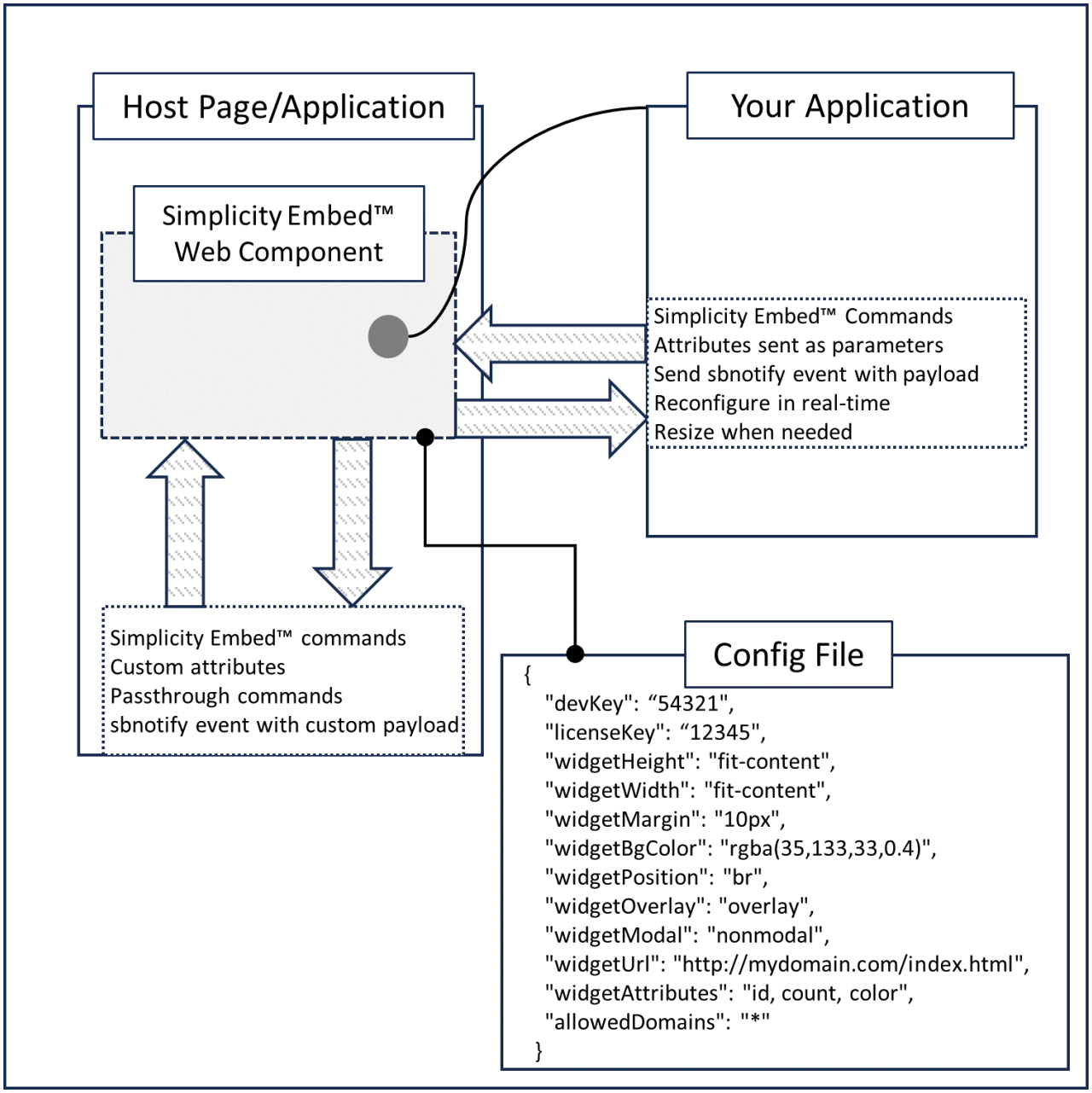
Allow others to simply integrate your web apps built with Angular, ASP.NET, Backbone, Django, Ember, Java, JQuery, Mithril, Node, PHP, React, Ruby, Svelte, Vanilla JS, Vue and more!

Any web application can be turned into an embeddable widget
Any frontend framework can be used
Any backend technology can be used

Features

- The Simplicity Embed™ solves many problems with integrating a web application into another web page or another web application built with another framework.
- Code Isolation
- CSS Isolation
- Custom Command Communications to and from the host page to your widget/app
- Resizing As Host Page Resizes
- Inline or Overlay Modes that Can be Changed at Runtime
- Modal or nonmodal overlay mode
- Custom attributes can be defined that get passed to your widget/app
- Configuration file allows for A/B testing, dynamic update of widget/app displayed

Architecture



Configuration File

The configuration file contains initial setup parameters for the component including appearance, position and the URL of the widget/application. It is referenced in the Simplicity Embed web component as a fully qualified URL. This gives you the ability to control the widget remotely if that is what you need to do (e.g., A/B testing, changing the referenced widget/app based on time/day, etc.).

Here is a sample configuration file:

```
{
    "devKey": "XXXXXXXXXXX",
    "licenseKey": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "widgetHeight": "fit-content",
    "widgetWidth": "fit-content",
    "widgetMargin": "10px",
    "widgetBgColor": "transparent",
    "widgetPosition": "br",
    "widgetOverlay": "overlay",
    "widgetModal": "nonmodal",
    "widgetUrl": "http://localhost:5500/index.html",
    "widgetAttributes": "id,data-icon,count,
                         data-color,
                         data-background,
                         data-heading,
                         data-subhead"
  }
```

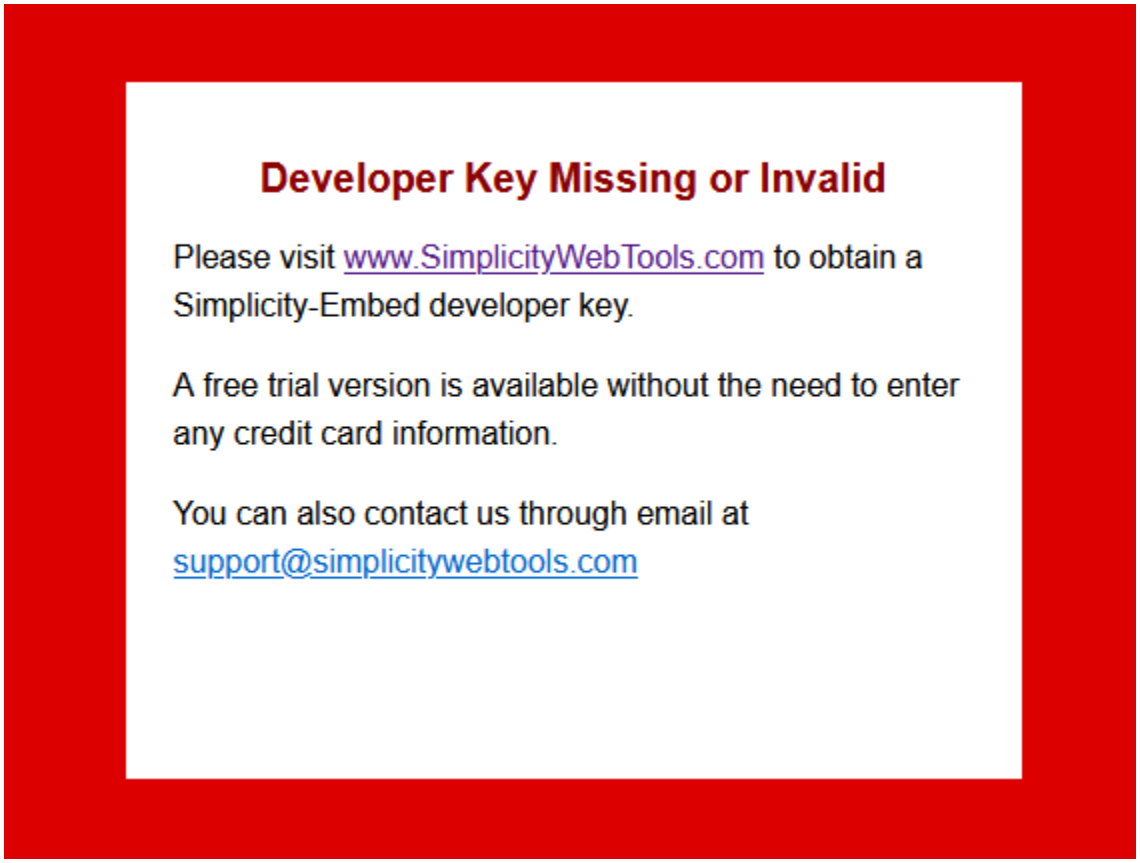
The following properties and their possible values are:

devKey

The Simplicity Embed™ web component requires a valid developer key or a valid license key to properly function. A developer key is either a free 30 day trial or a paid longer period developer's license. The developer key is only required if the widgetUrl property value is localhost or 127.0.0.1 (note: it IS NOT based on the domain of the page hosting the Simplicity Embed™ component).

A developer key can be obtained at www.SimplicityWebTools.com.

If an invalid developer key is in the devKey property field the component will display the following:



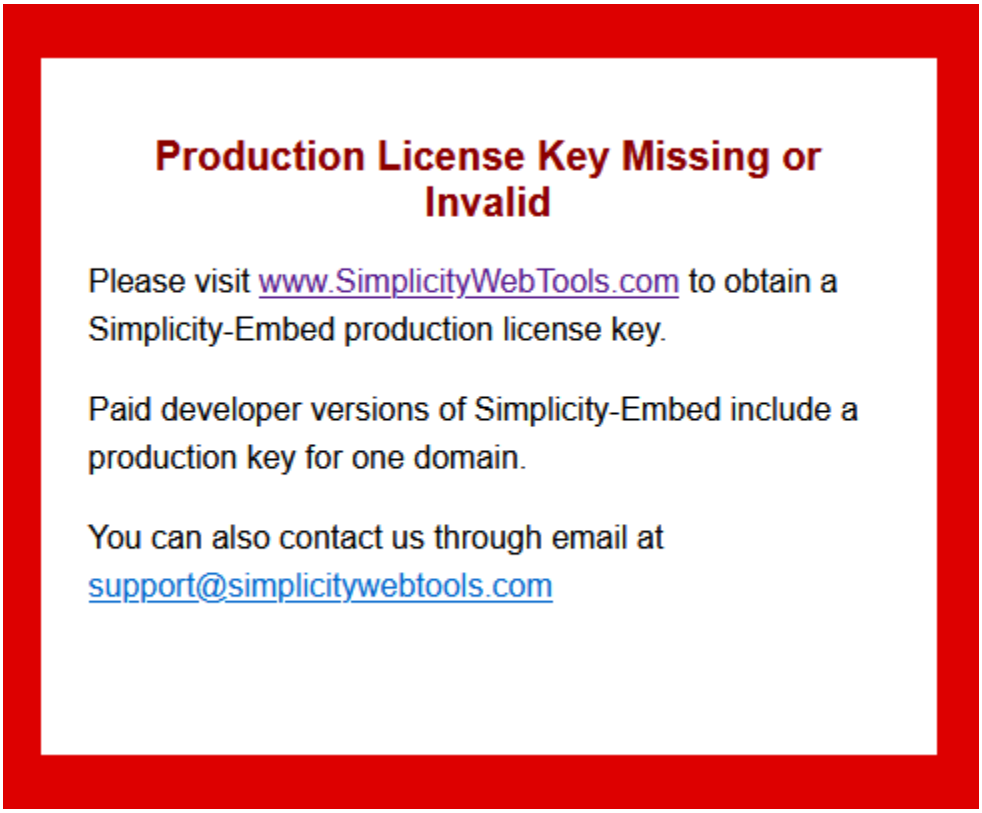
licenseKey

For production deployments (i.e., where the widget referenced by the Simplicity Embed web component is not hosted at localhost or 127.0.0.1) a valid key needs to be entered for the licenseKey property value. (note: the license key IS NOT based on the domain of the page hosting the Simplicity Embed™ component but instead the domain of where the widget is hosted).

A license key can be purchased at www.SimplicityWebTools.com.

If a license key is purchased for mydomain.com it is also valid for www.mydomain.com. A key purchased for a specific sub-domain (e.g., xyz.mydomain.com) will only work for that subdomain. Wildcard keys are also available that would work with any subdomain on a particular domain.

If an invalid developer key is in the devKey property field the component will display the following:



widgetHeight

Any valid CSS value for height will work for this parameter. It is best to match this to the specific height of your widget. In the case of utilizing inline mode you will typically use 100% so the component matches the height of the container.

Keep in mind that the component renders overlayed on the container page content. So if your widget is 200px x 200px you do not want to define a height larger than that.

The exception to the above would be if you for example have an animation of your widget. In that scenario you would define a larger area that could accommodate the path of the animation and then when the animation stops (assuming it does) send a command to fix the height to fit your widget.

widgetWidth

Any valid CSS value for width will work for this parameter. It is best to match this to the specific width of your widget. In the case of utilizing inline mode you will typically use 100% so the component matches the width of the container.

Keep in mind that the component renders overlayed on the container page content. So if your widget is 200px x 200px you do not want to define a width larger than that.

The exception to the above would be if you for example have an animation of your widget. In that scenario you would define a larger area that could accommodate the path of the animation and then when the animation stops (assuming it does) send a command to fix the width to fit your widget.

widgetMargin

Any valid CSS value for margin. This adds an outside margin around the widget.

widgetBgColor

The architecture of the Simplicity Embed™ component is that there is an overlay container injected into the host page. This parameter sets the background color of that. Typically you would be using this when the widget is acting in a modal mode.

This value can be any valid CSS value for color including (if you wish) alpha settings for transparency.

widgetPosition

This defines the position of the widget when in overlay mode. There are nine possible values for this:

tl – position the widget in the **TOP LEFT** of the view port

tm – position the widget in the **TOP MIDDLE** of the view port

tr – position the widget in the **TOP RIGHT** of the view port

ml – position the widget in the **MIDDLE LEFT** of the view port

mm – position the widget in the **MIDDLE MIDDLE** (i.e., center) of the view port

mr – position the widget in the **MIDDLE RIGHT** of the view port

bl – position the widget in the **BOTTOM LEFT** of the view port

bm – position the widget in the **BOTTOM MIDDLE** of the view port

br – position the widget in the **BOTTOM RIGHT** of the view port

widgetOverlay

There are two modes that the Simplicity Embed™ web component can operate regarding how the component integrates into the hosting page. The two modes are that are supported by the component:

- overlay
- inline

NOTE: If widgetOverlay is set to "inlne" widgetPreload MUST be present and set to "yes".

When this property is set to overlay the component and the widget container sits on top of the hosting page. This is typically what you would see for example in a chat widget.

When this property is set to inline the component and container stay contained within the parent element of where the Simplicity Embed™ web component is placed in the page.

widgetPreload

widgetPreload should generally be set to "yes". In some cases if the host page is not showing the widget immediately upon load, you may wish to delay the loading of the widget until it is displayed.

If widgetOverlay is set to "inlne" widgetPreload MUST be present and set to "yes".

widgetModal

The Simplicity Embed™ web component can operate in a modal or nonmodal mode. If it is operating in a modal mode the user cannot access page content versus nonmodal mode where the page content behind your widget can be accessed.

The two possible values for this property are:

modal – do not allow access to page content

nonmodal –allow access to page content

widgetUrl

This is the fully qualified URL of your widget/app. For production implementations the domain that the license key is generated for must match this domain.

If you have not obtained a deployment license key your widget will need to be hosted at localhost or 127.0.0.1.

widgetAttributes

You can define custom attributes that the end consumer can add to the Simplicity Embed™ web component to send configuration information to your widget. You can also include standard HTML attributes if it makes sense for that info to be passed to your widget. It is strongly recommended that one of the attributes you pass is "id". This gets used internally in the Simplicity Embed™ web component to differentiate instances when there is more than one implemented on a page.

Although it is common practice to use "data-*" for custom attributes, you can also use any unreserved string as an attribute and that will work too.

Implementing the Component

When a user hosts the Simplicity Embed™ component in their web page or app they place it just like any other HTML element:

```
<simplicity-embed
        setup="https://mifo.com/config.json"
        id="se1"
        ca1="val1"
        ca2="val2"
        ca3="val3"
    >
    </simplicity-embed>
```

The above shows the mandatory **setup** attribute which pulls in a configuration file from the fully qualified URL in the value. The id attribute is necessary if you want to allow two or more instances of the Simplicity Embed™ component on a single page.

Sample Widget App

We have made available a vanilla JavaScript widget example that is simple in concept but is designed to demonstrate all the communication interfaces between the Simplicity Embed™ web component, your widget and the hosting page.

This sample can be downloaded at:

https://www.simplicitywebtools.com/Downloads

Getting Started

The Simplicity Embed™ 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

In addition to the web component, we have a simplified widget sample that helps you learn how to develop a widget that communicates to and from a host page through the Simplicity Embed™ web component.

Each example references this sample widget. The sample widget code can be obtained at:

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

The version you use is based on the page/app that is hosting the SimplicityEmbed™ web component (<simplicity-embed></simplicity-embed>). If you are distributing a widget for general use on a web page, you would most likely select the vanilla JavaScript version. If you are using the Simplicity Embed™ to integrate a separate framework or web app into another framework, you would choose the version appropriate for where you are integrating into.

The component can be obtained from NPM or CDN:

For vanilla JavaScript:
via NPM

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

via CDN

```
<script type="module"
src="https://unpkg.com/@simplicitywebtools/simplicity-
embed/dist/simplicity-embed/simplicity-embed.esm.js"></script>
```

Vanilla JavaScript Example

The Simplicity Embed™ web component allows you to embed widgets or pages built in just about any framework into any HTML web page.

For this example we use a basic Bootstrap template example and add the Simplicity Embed™ web component to it along with some buttons and scripting to demonstrate various functionality.

The Bootstrap example we use is located at:

https://getbootstrap.com/docs/5.3/examples/product/

We have a public repo available that you can download to view this example. The repo is located at:

https://github.com/mfoitzik/simplicity-embed-vanillajs-example

You can clone the repo by doing the following:

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

In the body of the index.html file we add buttons and the Simplicity Embed™ web component:

```
<div class="col-md-5 p-lg-5 mx-auto my-5">
            <h1 class="display-4 fw-normal">Simplicity Embed&trade; Basic Example</h1>
            <p class="lead fw-normal">This is a basic Bootstrap example page that has a
Simplicity Embed&trade; web component used in it.
               Go to <a
href="https://www.simplicitywebtools.com">www.simplicitywebtools.com</a> to find out more.
            </p>
```

```
            <div>
              <button type="button" id="btnOpen">Open</button>
            </div>
            <div>
              <button type="button" id="btnClose">Close</button>
            </div>
            <div>
              <button type="button" id="btnSendCommand">Send Command</button>
            </div>
            <div>
              <button type="button" id="btnChangeConfig">Change Config</button>
            </div>
          </div>
```

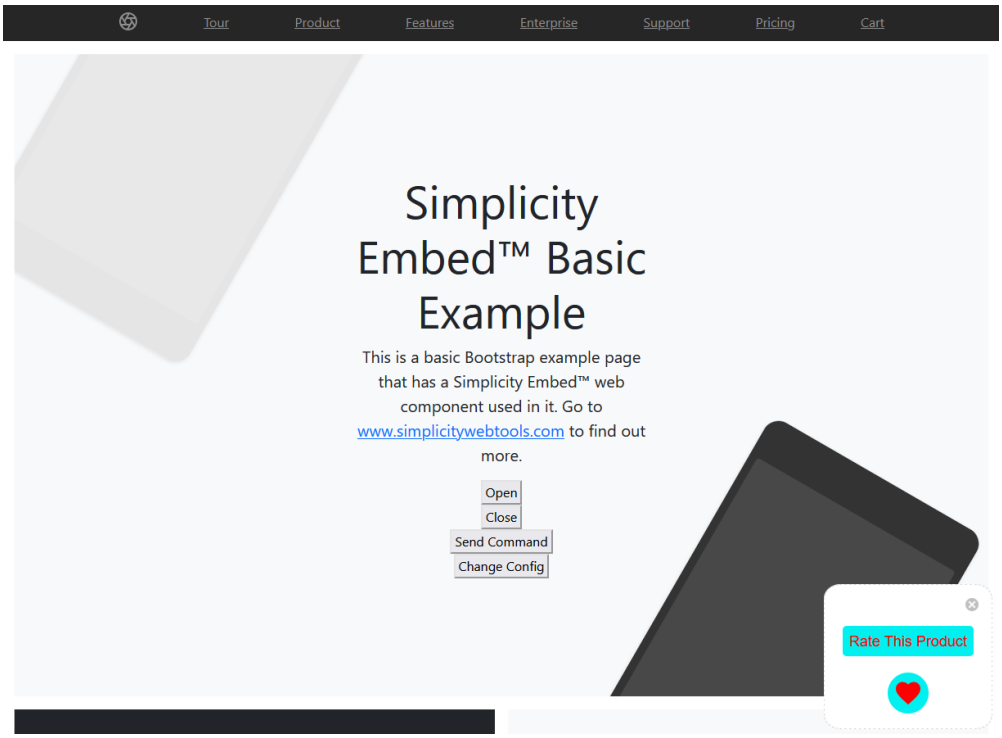
```
<simplicity-embed
      setup="http://localhost:5500/config.json"
      id="simplicity"
      data-icon="heart"
      data-count="7"
      data-color="rgb(255,0,0)"
      data-background="rgb(0,240,240)"
      data-heading="Rate This Product"
    />
```

At the bottom of the page, we pull in the Simplicity Embed™ scripting from the CDN and add scripting to handle the button events and the sbnotify event.

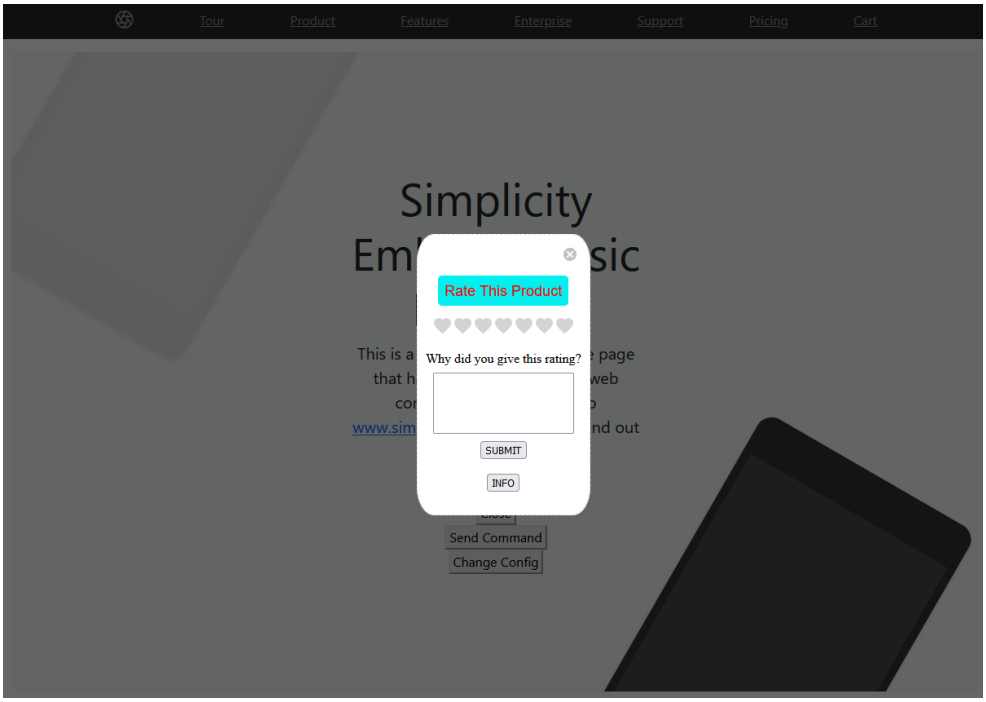
```
<script type="module" src="https://unpkg.com/@simplicitywebtools/simplicity-
embed@1.1.12/dist/simplicity-embed/simplicity-embed.esm.js"></script>
    <script>
      if (document.readyState === "loading") {
        document.addEventListener("DOMContentLoaded", init);
      } else {
        init();
      }
      function init() {
        const openBtn = document.getElementById("btnOpen");
        const closeBtn = document.getElementById("btnClose");
        const sendCommandBtn = document.getElementById("btnSendCommand");
        const changeConfigBtn = document.getElementById("btnChangeConfig");
        const seElement = document.getElementById("simplicity");
        openBtn.addEventListener("click", function() {
            seElement.open();
          });
        closeBtn.addEventListener("click", function() {
```

```
        seElement.close();
      });
      sendCommandBtn.addEventListener("click", function() {
        const sendObj = {
          "action": "spin"
        }
        seElement.sendMessage(sendObj);
      });
      changeConfigBtn.addEventListener("click", function() {
        seElement.setup = "http://localhost:5500/config2.json";
      });
      seElement.addEventListener("sbnotify", function(customEvent) {
        console.log("I received a notify event");
        console.log(customEvent.detail);
      })
    }


  </script>
```

After adding the above and running index.html in a dev server you should see:



Clicking in the body of the widget will show:

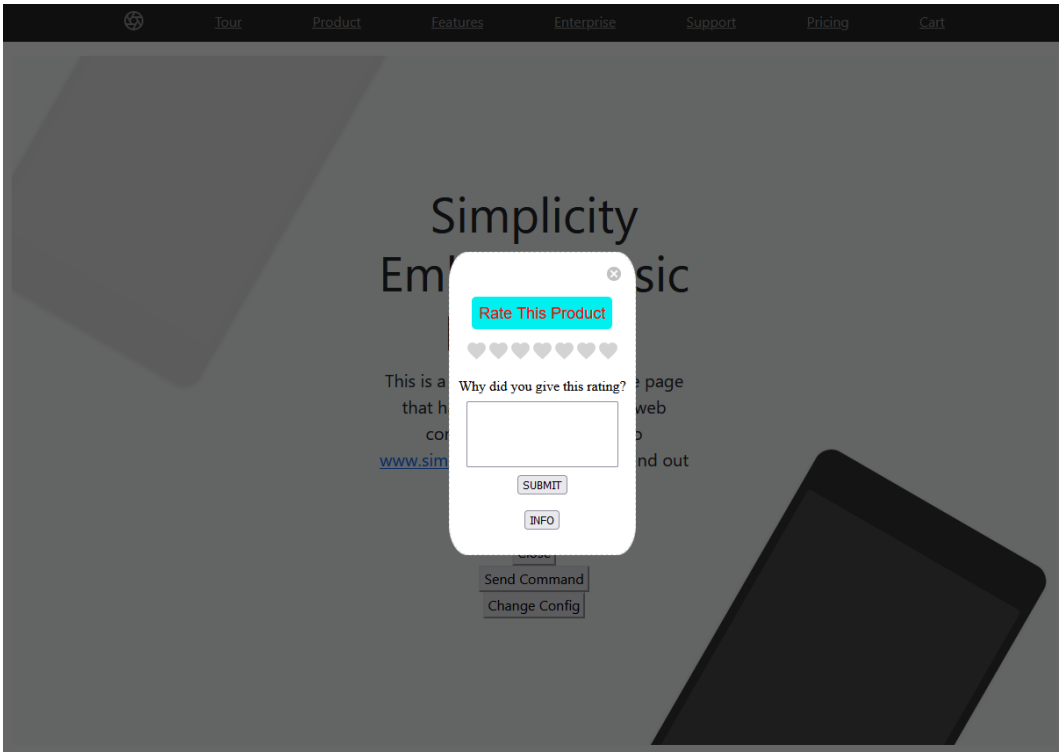


Concepts demonstrated:

There are three methods available to the Simplicity Embed™ web component's host: open, close, and sendMessage. Clicking the open or close button in the sample above simply opens or closes the component. The sendMessage method is a way for your widget to accept custom commands and information from the web component's host. In the case of this demo the widget example accepts an object with an "action" property with a value "spin". This object can be anything you want.

The "Change Config" button changes the value of the 'seSetup" variable that is bound to the "setup" property on the Simplicity Embed™ component. The component has a watcher on that attribute which will reload the new configuration file and determine any custom attributes that need to be passed to the defined widget in that configuration file.

The widget can communicate to the host page and the Simplicity Embed™ web component as well.



Inside the example vanilla JavaScript widget example the event handler for the "SUBMIT" button sends a postMessage to its parent's window. The Simplicity Embed™ web component receives this message and as long as the object sent does not have a reserved property name, it passes the object as an "sbnotify" event to the host page.

In the case of the example widget the submit button gathers the rating selected as well as the text placed in the text box and sends it with the object.

Here is the code in the widget:

```
submitBtn.addEventListener("click", function(e) {
          e.stopPropagation();
          const getMessage = document.getElementById("message");
          let outMessage = "";
          if (getMessage) {
            outMessage = getMessage.value;
          }
          console.log(chosenIcons);
          console.log(outMessage);
          let outRating = 0
          for (let x = 0; x < chosenIcons.length; x++) {
            if (chosenIcons[x] == true) {
```

```
            outRating++;
          }
        }
        let sendObject = {
            "command": "rating",
            "rating": outRating,
            "maxRating": chosenIcons.length,
            "message": outMessage,
            id: myId
          };
          window.parent.postMessage(sendObject, "*");
      });
```

Inside the host page the handler for sbnotify snippet is here:

```
handleNotify(event: Event) {
    const customEvent = event as CustomEvent<any>;
    console.log("I received a notify event");
    console.log(customEvent.detail);
  }
```

The received object (shown in the console log) will look like this:

```
{
  "command": "rating",
  "rating": 6,
  "maxRating": 7,
  "message": "Because this page is great!",
  "id": "simplicity"
}
```

Sending the info request to the Simplicity Embed™ web component requires you to do a postMessage to the widget's parent window with an object that contains the property "cmd" with a value "info" as shown here:

```
infoBtn.addEventListener("click", function(e) {
        let sendObject = {
            "cmd": "info",
            id: myId
          };
          window.parent.postMessage(sendObject, "*");
      });
```

The Simplicity Embed™ web component will respond by doing a postMessage that you can listen for that will contain an object with a variety of information about the Simplicity Embed™ web component's host such as parent dimensions, page dimensions, host, etc.

You need to have an event listener defined in your widget that listens for the message event on your window object as shown here:

```
//listen for component host messages/commands
        window.addEventListener("message", (event) => {
          if (event.data.action) {
            switch(event.data.action) {
              case "spin":
              init();
              const icon = document.getElementById("icon");
                icon.classList.remove("spin");
                const myParent = icon.closest("div");
                const inHTML = myParent.innerHTML;
                myParent.innerHTML = inHTML;
                const icon2 = document.getElementById("icon");
                icon2.classList.add("spin");
                break;
            }
          }
          if (event.data.cmd) {
            if (event.data.cmd == "info") {
              console.log("INFO RECEIVED");
              console.log(event.data);
            }
          }
        });
```

An example of a returned object from the info request is shown here:

```
{
  "cmd": "info",
  "info": {
    "tagName": "MAIN",
    "clientHeight": 496,
    "clientWidth": 944,
    "paddingTop": "",
    "paddingRight": "",
    "paddingBottom": "",
    "paddingLeft": "",
    "htmlHeight": 668,
    "htmlWidth": 960,
```

```
    "href": "http://localhost:61323/",
    "host": "localhost:61323",
    "hostname": "localhost",
    "pathname": "/"
  }
```

}

You can use this information to position your widget, animate it, limit it to specific hosts, etc.

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 Easiest Way to Start

The instructions that follow are just a guideline for "start from scratch" projects. To get familiar with Simplicity Embed™ 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-embed-angular-standalone-example

You can clone the repo by doing the following:

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

```
npm install @simplicitywebtools/simplicity-embed-angular
```

Download the sample vanilla JavaScript widget from the following location:

https://www.SimplicityWebTools.com/SimplicityEmbedSampleDownloads#samplewidget

Open the sample widget folder in Visual Studio Code and run the index.html file in Live Server (or whatever dev server you have configured in your environment). Make note of the address/port that the page launches in (typically http://localhost:5500). This widget sample contains two files that will get utilized by the project. One is a configuration file that is referenced by the Simplicity Embed™ component (config.json) and the other is the actual widget (index.html).

PLEASE NOTE: The trial key and developer license key only work with the 'localhost' domain or 127.0.0.1 IP address.

The simplicity-embed-angular component has a wrapper that can be used in an Angular standalone component as well as a module.

Standalone (default as of Angular 17)

For an example of integrating the Simplicity Embed™ web component in an Angular module component project reference the following repo:

https://github.com/mfoitzik/simplicity-embed-angular-module-example

Create a new Angular project

```
ng new angular-standalone
```

accept all defaults

cd angular-standalone

Install the simplicity-embed-angular component:

npm install @simplicitywebtools/simplicity-embed-angular

In the src/app folder modify the app.component.ts file as follows:

src/app/app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';
import { SimplicityEmbed } from '@simplicitywebtools/simplicity-embed-angular';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterOutlet, SimplicityEmbed],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent implements OnInit  {
  title = 'angular-standalone';
  seSetup = "http://localhost:5500/config.json";
  private sembed: SimplicityEmbed | null = null;

  openSimplicityEmbed() {
    this.sembed?.open();
  }
  closeSimplicityEmbed() {
    this.sembed?.close();
  }
  sendCommand() {
    const sendObj = {
      "action": "spin"
    }
    this.sembed?.sendMessage(sendObj);
  }
  changeConfig() {
    this.seSetup = "http://localhost:5500/config2.json";
  }
  handleNotify(event: Event) {
    const customEvent = event as CustomEvent<any>;
    console.log("I received a notify event");
    console.log(customEvent.detail);
  }
  ngOnInit() {
    this.sembed = document.getElementById("simplicity") as any;
  }

}
```

In the src/app/app.component.html file add the Simplicity Embed™ component as follows (it is being inserted right above the closing </main> tag.

Also, in the src/app/app.component.html file add the following buttons under the "Congratulations! Your app is running" line

src/app/app.component.html

```
  </div>
  <simplicity-embed

    [attr.setup]="seSetup"
    id="simplicity"
    data-icon="heart"
    data-count="7"
    data-color="rgb(245, 241, 5)"
    data-background="rgb(58,85,156)"
    data-heading="Rate Me!"

    (sbnotify) = "handleNotify($event)">
  </simplicity-embed>
</main>
```

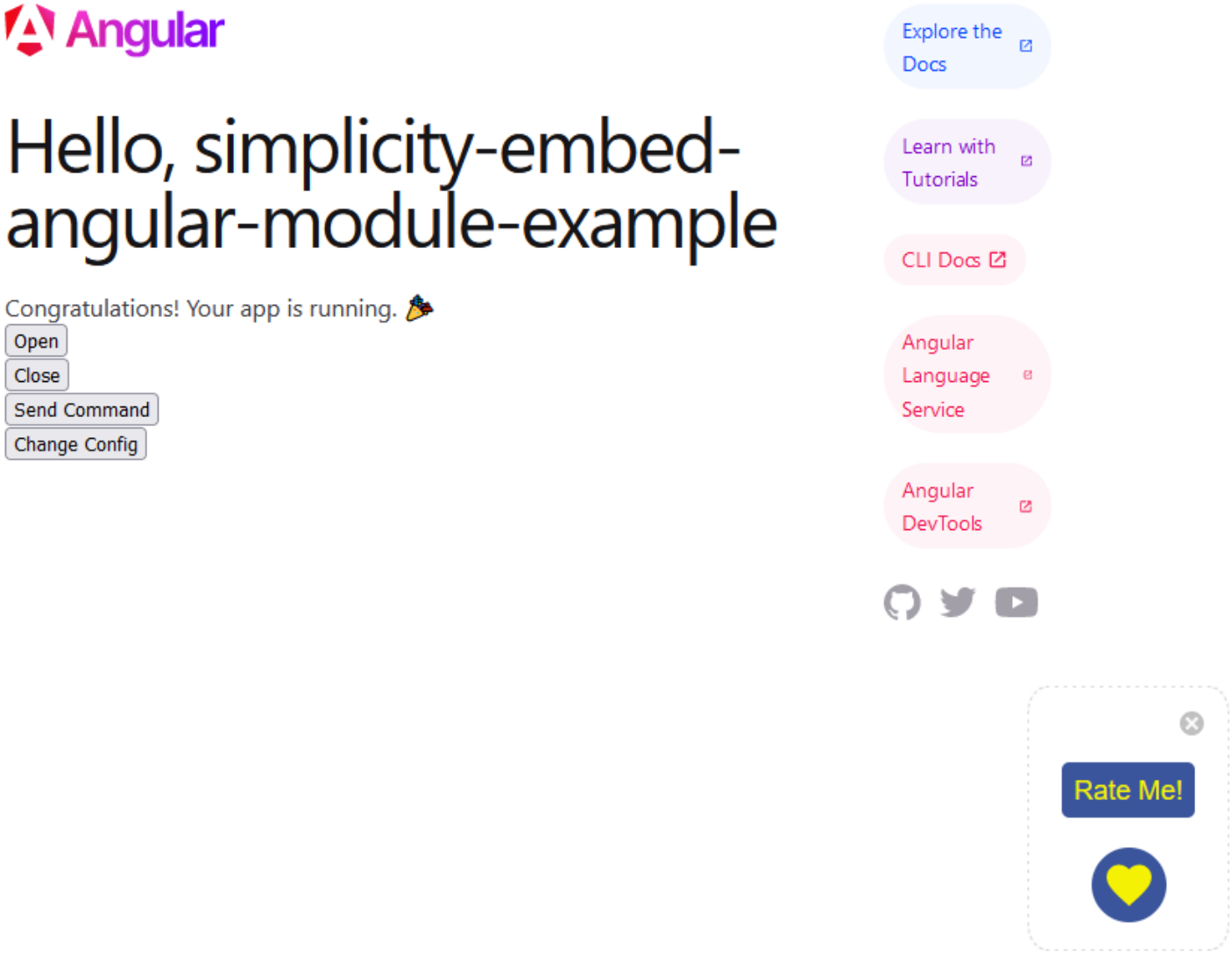
Notice that attribute binding is only applied to the setup attribute. This is because the other attributes are dynamically assigned and do not have listeners attached to them.

Also in: src/app/app.component.html

```
<p>Congratulations! Your app is running. 🎉</p>
      <div>
        <button type="button" (click)=openSimplicityEmbed()>Open</button>
      </div>
      <div>
        <button type="button" (click)=closeSimplicityEmbed()>Close</button>
      </div>
      <div>
        <button type="button" (click)=sendCommand()>Send Command</button>
      </div>
      <div>
        <button type="button" (click)=changeConfig()>Change Config</button>
      </div>
```

```
Run the project:

ng serve –open
```

You should see the following:

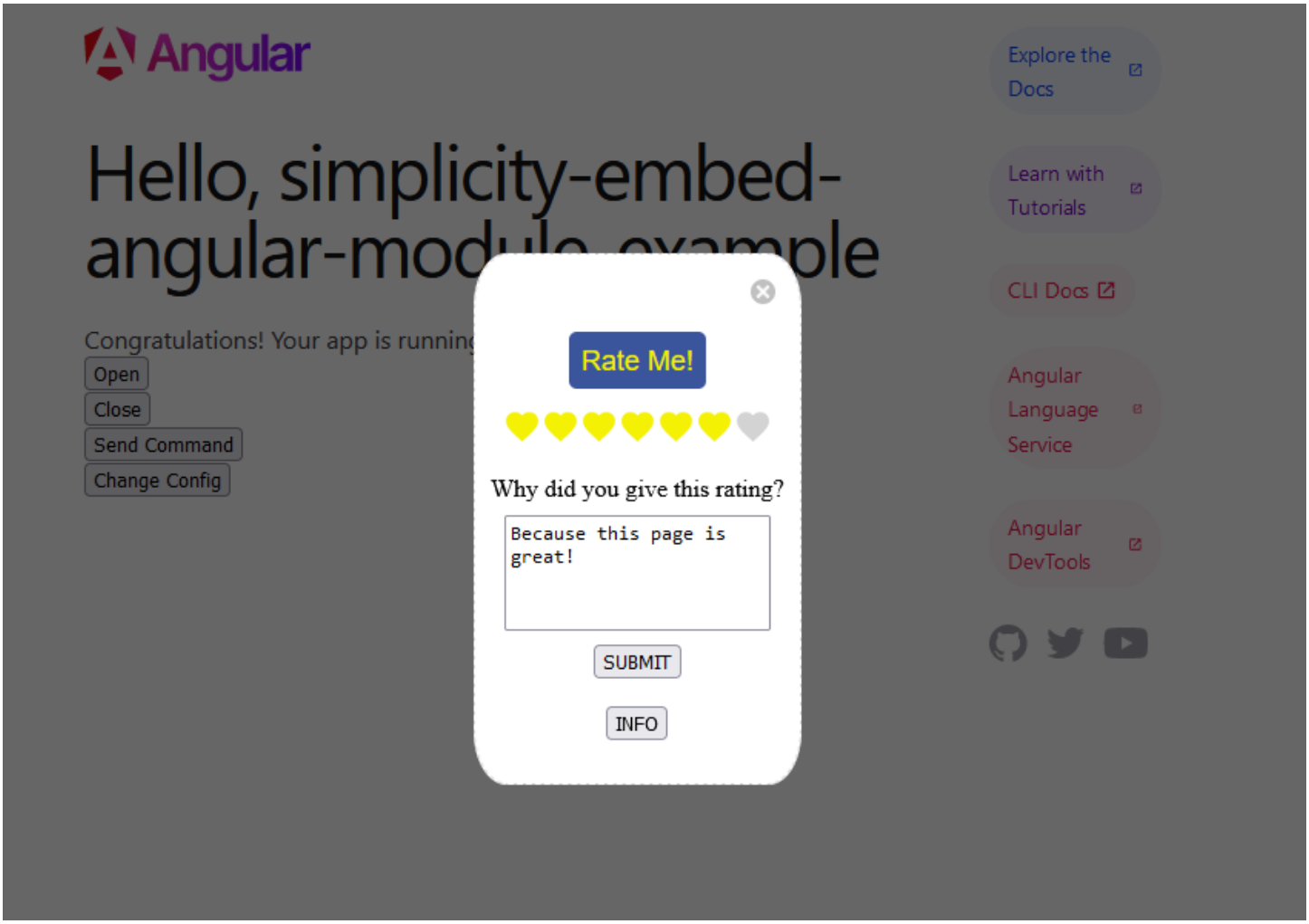


Concepts Demonstrated

There are three methods available to the Simplicity Embed™ web component's host: open, close, and sendMessage. Clicking the open or close button in the sample above simply opens or closes the component. The sendMessage method is a way for your widget to accept custom commands and information from the web component's host. In the case of this demo the widget example accepts an object with an "action" property with a value "spin". This object can be anything you want.

The "Change Config" button changes the value of the 'seSetup" variable that is bound to the "setup" property on the Simplicity Embed™ component. The component has a watcher on that attribute which will reload the new configuration file and determine any custom attributes that need to be passed to the defined widget in that configuration file.

The widget can communicate to the host page and the Simplicity Embed™ web component as well.



Inside the example vanilla JavaScript widget example the event handler for the "SUBMIT" button sends a postMessage to its parent's window. The Simplicity Embed™ web component receives this message and as long as the object sent does not have a reserved property name, it passes the object as an "sbnotify" event to the host page.

In the case of the example widget the submit button gathers the rating selected as well as the text placed in the text box and sends it with the object.

Here is the code in the widget:

```
submitBtn.addEventListener("click", function(e) {
          e.stopPropagation();
          const getMessage = document.getElementById("message");
          let outMessage = "";
          if (getMessage) {
```

```
            outMessage = getMessage.value;
          }
          console.log(chosenIcons);
          console.log(outMessage);
          let outRating = 0
          for (let x = 0; x < chosenIcons.length; x++) {
            if (chosenIcons[x] == true) {
              outRating++;
            }
          }
          let sendObject = {
              "command": "rating",
              "rating": outRating,
              "maxRating": chosenIcons.length,
              "message": outMessage,
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

Inside the host page the handler for sbnotify snippet is here:

```
handleNotify(event: Event) {
    const customEvent = event as CustomEvent<any>;
    console.log("I received a notify event");
    console.log(customEvent.detail);
  }
```

The received object (shown in the console log) will look like this:

```
{
  "command": "rating",
  "rating": 6,
  "maxRating": 7,
  "message": "Because this page is great!",
  "id": "simplicity"
}
```

Sending the info request to the Simplicity Embed™ web component requires you to do a postMessage to the widget's parent window with an object that contains the property "cmd" with a value "info" as shown here:

```
infoBtn.addEventListener("click", function(e) {
```

```
          let sendObject = {
              "cmd": "info",
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

The Simplicity Embed™ web component will respond by doing a postMessage that you can listen for that will contain an object with a variety of information about the Simplicity Embed™ web component's host such as parent dimensions, page dimensions, host, etc.

You need to have an event listener defined in your widget that listens for the message event on your window object as shown here:

```
//listen for component host messages/commands
        window.addEventListener("message", (event) => {
          if (event.data.action) {
            switch(event.data.action) {
              case "spin":
              init();
              const icon = document.getElementById("icon");
                icon.classList.remove("spin");
                const myParent = icon.closest("div");
                const inHTML = myParent.innerHTML;
                myParent.innerHTML = inHTML;
                const icon2 = document.getElementById("icon");
                icon2.classList.add("spin");
                break;
            }
          }
          if (event.data.cmd) {
            if (event.data.cmd == "info") {
              console.log("INFO RECEIVED");
              console.log(event.data);
            }
          }
        });
```

An example of a returned object from the info request is shown here:


```
{
  "cmd": "info",
  "info": {
    "tagName": "MAIN",
    "clientHeight": 496,
    "clientWidth": 944,
    "paddingTop": "",
    "paddingRight": "",
    "paddingBottom": "",
    "paddingLeft": "",
    "htmlHeight": 668,
    "htmlWidth": 960,
    "href": "http://localhost:61323/",
    "host": "localhost:61323",
    "hostname": "localhost",
    "pathname": "/"
  }
```


}

You can use this information to position your widget, animate it, limit it to specific hosts, etc.

React Example

This example uses a vanilla JavaScript widget to demonstrate embedding into a React project. This is just to simplify the demonstration. In practical cases a widget built in any framework could be integrated.

Download the sample vanilla JavaScript widget from the following location:

https://www.SimplicityWebTools.com/SimplicityEmbedSampleDownloads#samplewidget

Open the sample widget folder in Visual Studio Code and run the index.html file in Live Server (or whatever dev server you have configured in your environment). Make note of the address/port that the page launches in (typically http://localhost:5500). This widget sample contains two files that will get utilized by the project. One is a configuration file that is referenced by the Simplicity Embed™ component (config.json) and the other is the actual widget (index.html).

Start a new React Project using vitejs

```
npm create vite@latest
```

For this example, we chose the following from the vitejs prompts:

```
Project name: ... simplicity-embed-react-example
Select a framework: » React
Select a variant: » TypeScript
```

Change to the project directory, npm install and run:

```
cd simplicity-embed-react-example
npm install
npm run dev
```

Vite + React

count is 0

Edit src/App.tsx and save to test HMR

Click on the Vite and React logos to learn more

Install the Simplicity Embed™ web component for React:

For React

```
npm install @simplicitywebtools/simplicity-embed-react
```

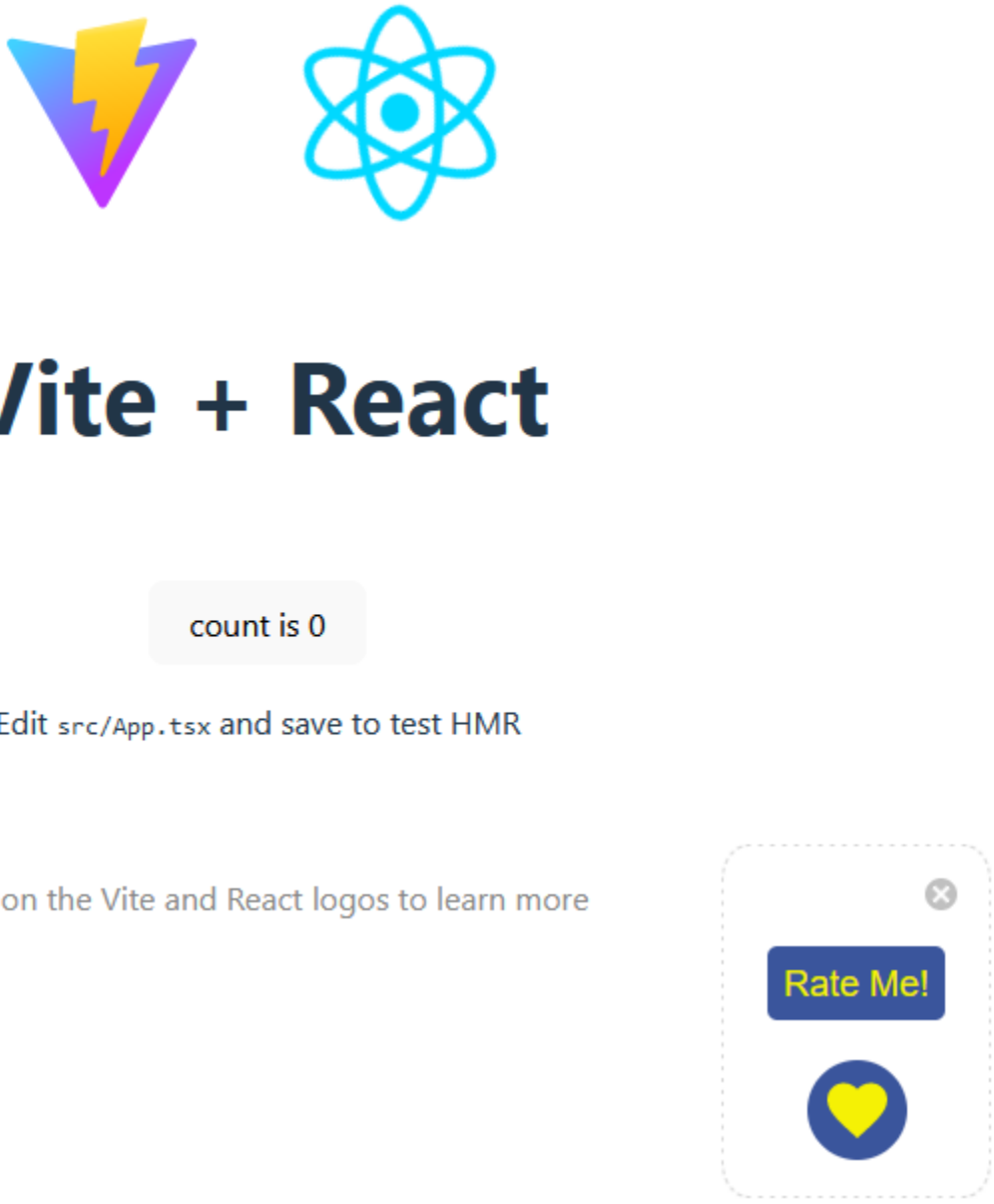
Modify the App.tsx file to import the Simplicity Embed™ web component, initialize it and use the component in the page as follows:
(NOTE: Some buttons and handlers are added to demonstrate various communication paths between the host page, Simplicity Embed™ web component and the widget.

src/App.tsx

```
import { useState, useRef } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import  { SimplicityEmbed, defineCustomElements } from '@simplicitywebtools/simplicity-embed-
react'
```

```
defineCustomElements();
function App() {
  const [count, setCount] = useState(0);
  const [seSetup, setSeSetup] = useState("http://localhost:5500/config.json");
  const seRef = useRef<HTMLSimplicityEmbedElement | null>(null);
  function openSimplicityEmbed() {
    if (seRef.current) {

      seRef.current.open();
    }
  }
  function closeSimplicityEmbed() {
    seRef.current?.close();
  }
  function sendCommand() {
    const sendObj = {
      "action": "spin"
    }
    seRef.current?.sendMessage(sendObj);
  }
  function changeConfig() {
    setSeSetup("http://localhost:5500/config2.json");
  }
  function handleSbNotify(event: CustomEvent) {
    console.log("I received a notify event");
    console.log(event.detail);
  }

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <div>
        <button type="button" onClick={openSimplicityEmbed}>Open</button>
      </div>
      <div>
```

```
        <button type="button" onClick={closeSimplicityEmbed}>Close</button>
      </div>
      <div>
        <button type="button" onClick={sendCommand}>Send Command</button>
      </div>
      <div>
        <button type="button" onClick={changeConfig}>Change Config</button>
      </div>
        <p>
          Edit <code>src/App.tsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
      <SimplicityEmbed
    setup={seSetup}
    id="simplicity"
    ref={seRef}
    data-icon="heart"
    data-count="7"
    data-color="rgb(245, 241, 5)"
    data-background="rgb(58,85,156)"
    data-heading="Rate Me!"
    onSbnotify={handleSbNotify}>
  </SimplicityEmbed>
    </>
  )
}

export default App
```

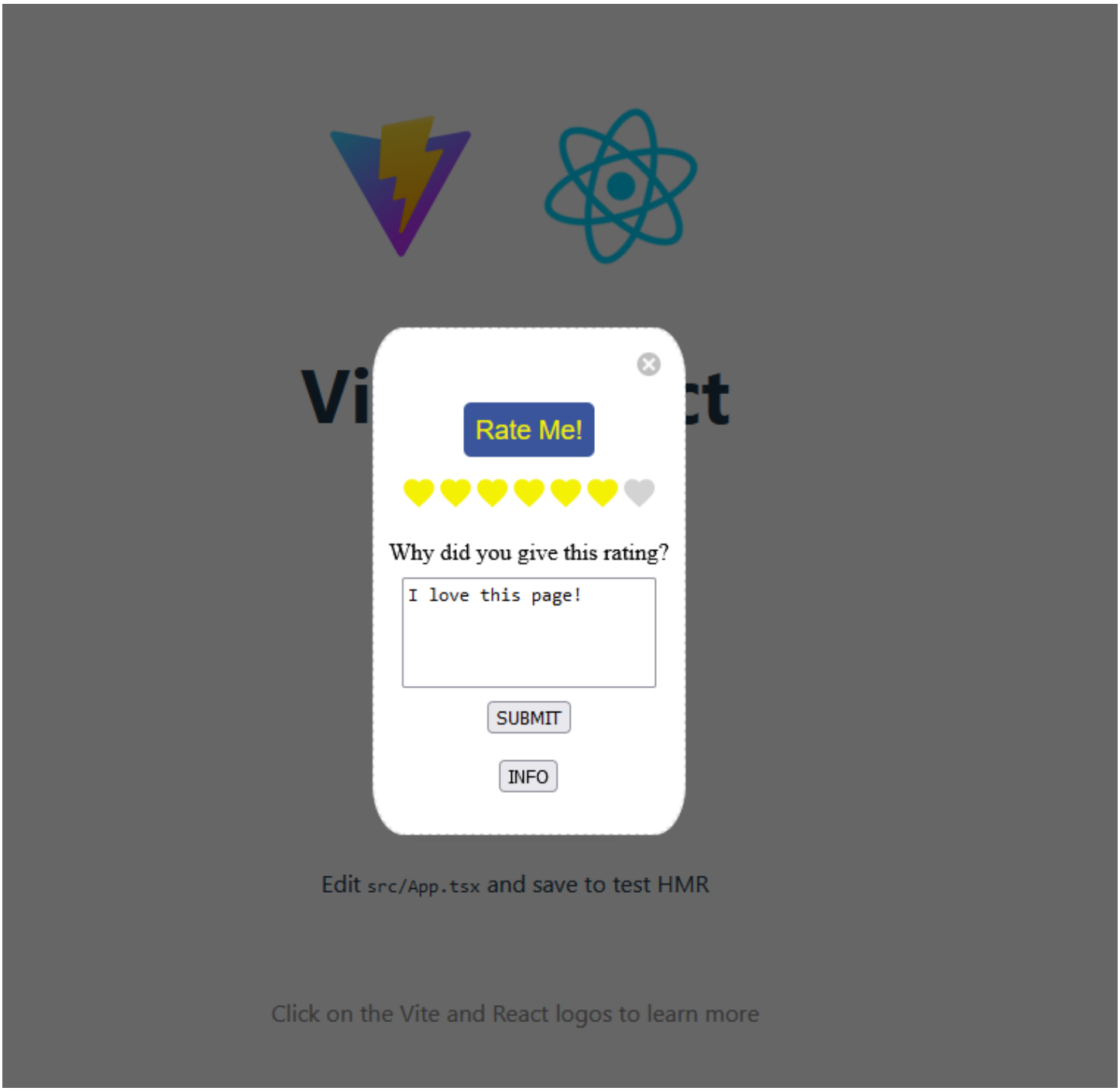


Concepts Demonstrated

There are three methods available to the Simplicity Embed™ web component's host: open, close, and sendMessage. Clicking the open or close button in the sample above simply opens or closes the component. The sendMessage method is a way for your widget to accept custom commands and information from the web component's host. In the case of this demo the widget example accepts an object with an "action" property with a value "spin". This object can be anything you want.

The "Change Config" button changes the value of the 'seSetup" variable that is bound to the "setup" property on the Simplicity Embed™ component. The component has a watcher on that attribute which will reload the new configuration file and determine any custom attributes that need to be passed to the defined widget in that configuration file.

The widget can communicate to the host page and the Simplicity Embed™ web component as well.



Inside the example vanilla JavaScript widget example the event handler for the "SUBMIT" button sends a postMessage to its parent's window. The Simplicity Embed™ web component receives this message and as long as the object sent does not have a reserved property name, it passes the object as an "sbnotify" event to the host page.

In the case of the example widget the submit button gathers the rating selected as well as the text placed in the text box and sends it with the object.

Here is the code in the widget:

```
submitBtn.addEventListener("click", function(e) {
          e.stopPropagation();
          const getMessage = document.getElementById("message");
          let outMessage = "";
          if (getMessage) {
            outMessage = getMessage.value;
          }
          console.log(chosenIcons);
          console.log(outMessage);
          let outRating = 0
          for (let x = 0; x < chosenIcons.length; x++) {
            if (chosenIcons[x] == true) {
              outRating++;
            }
          }
          let sendObject = {
              "command": "rating",
              "rating": outRating,
              "maxRating": chosenIcons.length,
              "message": outMessage,
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

Inside the host page the handler for sbnotify snippet is here:

```
function handleSbNotify(event: CustomEvent) {
    console.log("I received a notify event");
    console.log(event.detail);
  }
```

The received object (shown in the console log) will look like this:

```
{
  "command": "rating",
  "rating": 6,
  "maxRating": 7,
  "message": "Because this page is great!",
  "id": "simplicity"
}
```

Sending the info request to the Simplicity Embed™ web component requires you to do a postMessage to the widget's parent window with an object that contains the property "cmd" with a value "info" as shown here:

```
infoBtn.addEventListener("click", function(e) {
          let sendObject = {
              "cmd": "info",
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

The Simplicity Embed™ web component will respond by doing a postMessage that you can listen for that will contain an object with a variety of information about the Simplicity Embed™ web component's host such as parent dimensions, page dimensions, host, etc.

You need to have an event listener defined in your widget that listens for the message event on your window object as shown here:

```
//listen for component host messages/commands
        window.addEventListener("message", (event) => {
          if (event.data.action) {
            switch(event.data.action) {
              case "spin":
              init();
              const icon = document.getElementById("icon");
                icon.classList.remove("spin");
                const myParent = icon.closest("div");
                const inHTML = myParent.innerHTML;
                myParent.innerHTML = inHTML;
                const icon2 = document.getElementById("icon");
                icon2.classList.add("spin");
                break;
            }
          }
          if (event.data.cmd) {
            if (event.data.cmd == "info") {
```

```
            console.log("INFO RECEIVED");
            console.log(event.data);
          }
        }
      });
```

An example of a returned object from the info request is shown here:

```
{
  "cmd": "info",
  "info": {
    "tagName": "MAIN",
    "clientHeight": 496,
    "clientWidth": 944,
    "paddingTop": "",
    "paddingRight": "",
    "paddingBottom": "",
    "paddingLeft": "",
    "htmlHeight": 668,
    "htmlWidth": 960,
    "href": "http://localhost:61323/",
    "host": "localhost:61323",
    "hostname": "localhost",
    "pathname": "/"
  }
```

}

You can use this information to position your widget, animate it, limit it to specific hosts, etc.

Vue Example

Start a new Vite-powered Vuejs project

```
npm create vue@latest
Need to install the following packages:
  create-vue@3.9.1
Ok to proceed? (y)


Vue.js - The Progressive JavaScript Framework
√ Project name: ... simplicity-embed-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-embed-vuejs-example...

Done. Now run:

  cd simplicity-embed-vuejs-example
  npm install
  npm run dev
```

Install the Simplicity Embed™ component:

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

Modify the App.vue file as follows:

```
<script setup lang="ts">
import HelloWorld from './components/HelloWorld.vue'
import TheWelcome from './components/TheWelcome.vue'
import { ref } from 'vue'
import {SimplicityEmbed} from '@simplicitywebtools/simplicity-embed-vue/lib'
const seSetup = ref('http://localhost:5500/config.json')
const seRef = ref<HTMLSimplicityEmbedElement | null>(null);
function openSimplicityEmbed() {
  seRef.value?.$el.open()
}
  function closeSimplicityEmbed() {
    seRef.value?.$el.close();
  }
  function sendCommand() {
```

```
    const sendObj = {
      "action": "spin"
    }
    seRef.value?.$el.sendMessage(sendObj);
  }
  function changeConfig() {
    seSetup.value = "http://localhost:5500/config2.json";
  }
  function handleSbNotify(event: CustomEvent) {
    console.log("I received a notify event");
    console.log(event.detail);
  }
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="./assets/logo.svg" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />
    </div>
  </header>

  <main>
    <TheWelcome />
    <div>
        <button type="button" @click="openSimplicityEmbed">Open</button>
      </div>
      <div>
        <button type="button" @click="closeSimplicityEmbed">Close</button>
      </div>
      <div>
        <button type="button" @click="sendCommand">Send Command</button>
      </div>
      <div>
        <button type="button" @click="changeConfig">Change Config</button>
      </div>
    <simplicity-embed
    :setup="seSetup"
    ref="seRef"
    id="simplicity"
    data-icon="heart"
    data-count="7"
    data-color="rgb(245, 241, 5)"
    data-background="rgb(58,85,156)"
    data-heading="Rate Me!"
    @sbnotify="handleSbNotify">
```

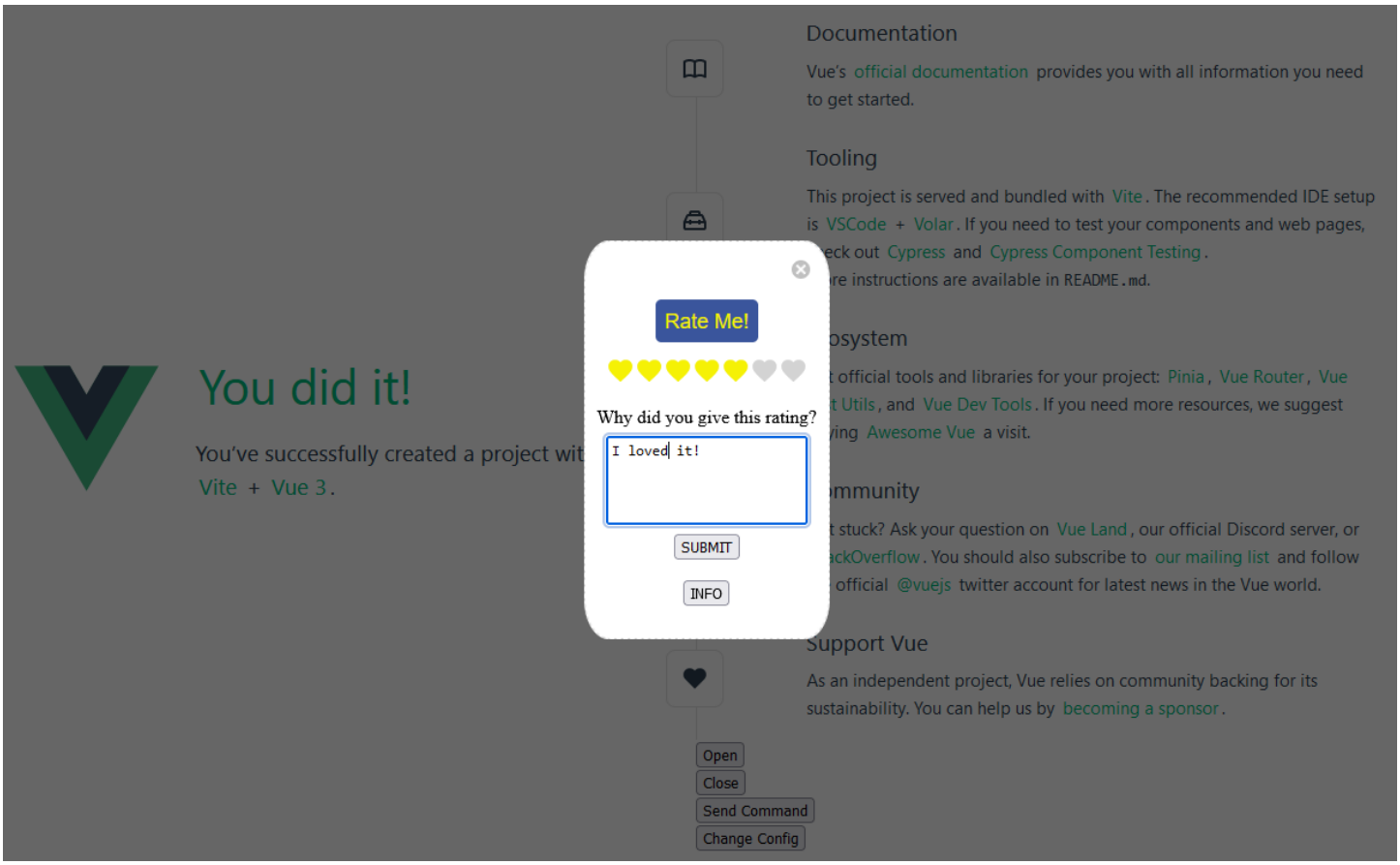
```
  </simplicity-embed>
  </main>
</template>

<style scoped>
header {
  line-height: 1.5;
}

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  .logo {
    margin: 0 2rem 0 0;
  }

  header .wrapper {
    display: flex;
    place-items: flex-start;
    flex-wrap: wrap;
  }
}
</style>
```

Concepts Demonstrated

There are three methods available to the Simplicity Embed™ web component's host: open, close, and sendMessage. Clicking the open or close button in the sample above simply opens or closes the component. The sendMessage method is a way for your widget to accept custom commands and information from the web component's host. In the case of this demo the widget example accepts an object with an "action" property with a value "spin". This object can be anything you want.

The "Change Config" button changes the value of the 'seSetup" variable that is bound to the "setup" property on the Simplicity Embed™ component. The component has a watcher on that attribute which will reload the new configuration file and determine any custom attributes that need to be passed to the defined widget in that configuration file.

The widget can communicate to the host page and the Simplicity Embed™ web component as well.



Inside the example vanilla JavaScript widget example the event handler for the "SUBMIT" button sends a postMessage to its parent's window. The Simplicity Embed™ web component receives this message and as long as the object sent does not have a reserved property name, it passes the object as an "sbnotify" event to the host page.

In the case of the example widget the submit button gathers the rating selected as well as the text placed in the text box and sends it with the object.

Here is the code in the widget:

```
submitBtn.addEventListener("click", function(e) {
          e.stopPropagation();
          const getMessage = document.getElementById("message");
```

```
          let outMessage = "";
          if (getMessage) {
            outMessage = getMessage.value;
          }
          console.log(chosenIcons);
          console.log(outMessage);
          let outRating = 0
          for (let x = 0; x < chosenIcons.length; x++) {
            if (chosenIcons[x] == true) {
              outRating++;
            }
          }
          let sendObject = {
              "command": "rating",
              "rating": outRating,
              "maxRating": chosenIcons.length,
              "message": outMessage,
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

Inside the host page the handler for sbnotify snippet is here:

```
function handleSbNotify(event: CustomEvent) {
    console.log("I received a notify event");
    console.log(event.detail);
  }
```

The received object (shown in the console log) will look like this:

```
{
  "command": "rating",
  "rating": 6,
  "maxRating": 7,
  "message": "Because this page is great!",
  "id": "simplicity"
}
```

Sending the info request to the Simplicity Embed™ web component requires you to do a postMessage to the widget's parent window with an object that contains the property "cmd" with a value "info" as shown here:

```
infoBtn.addEventListener("click", function(e) {
          let sendObject = {
              "cmd": "info",
              id: myId
            };
            window.parent.postMessage(sendObject, "*");
        });
```

The Simplicity Embed™ web component will respond by doing a postMessage that you can listen for that will contain an object with a variety of information about the Simplicity Embed™ web component's host such as parent dimensions, page dimensions, host, etc.

You need to have an event listener defined in your widget that listens for the message event on your window object as shown here:

```
//listen for component host messages/commands
        window.addEventListener("message", (event) => {
          if (event.data.action) {
            switch(event.data.action) {
              case "spin":
              init();
              const icon = document.getElementById("icon");
                icon.classList.remove("spin");
                const myParent = icon.closest("div");
                const inHTML = myParent.innerHTML;
                myParent.innerHTML = inHTML;
                const icon2 = document.getElementById("icon");
                icon2.classList.add("spin");
                break;
            }
          }
          if (event.data.cmd) {
            if (event.data.cmd == "info") {
              console.log("INFO RECEIVED");
              console.log(event.data);
            }
          }
        });
```

An example of a returned object from the info request is shown here:

```
{
  "cmd": "info",
  "info": {
    "tagName": "MAIN",
    "clientHeight": 496,
```

```
    "clientWidth": 944,
    "paddingTop": "",
    "paddingRight": "",
    "paddingBottom": "",
    "paddingLeft": "",
    "htmlHeight": 668,
    "htmlWidth": 960,
    "href": "http://localhost:61323/",
    "host": "localhost:61323",
    "hostname": "localhost",
    "pathname": "/"
  }
```


}

You can use this information to position your widget, animate it, limit it to specific hosts, etc.