shuffleboard2js Documentation

Amory Galili

Mar 01, 2019

Custom Widgets

shuffleboard2js further lowers the barrier of entry for teams that want to build a custom HTML/Javascript dashboard by providing a Shuffleboard like interface built on top of pynetworktables2js.

Lots of students and mentors know how to create simple web pages to display content, and there's lots of resources out there for creating dynamic content for webpages that use javascript. There is a lot of visually appealing content that others have created using web technologies – why not leverage those resources to make something cool to control your robot?

CHAPTER 1

Documentation

Documentation can be found at http://shuffleboard2js.readthedocs.org/

CHAPTER 2

Installation

Make sure to install python 3 on your computer, and on Windows you can execute:

```
py -3 -m pip install shuffleboard2js
```

On Linux/OSX you can execute:

```
pip install shuffleboard2js
```

CHAPTER 3

Usage

You can run shuffleboard2js using the following command:

```
python3 -m shuffleboard2js
```

Or on Windows:

```
py -3 -m shuffleboard2js
```

This will start a server which will serve from the current directory. You can create your custom widgets in the **shuffleboard2js/widgets** folder, which will automatically be created in the directory you ran shuffleboard2js.

You will want to also pass either the `--robot` or `--team` switch:

```
py -3 -m shuffleboard2js --robot roborio-XXXX-frc.local
py -3 -m shuffleboard2js --team XXXX
```

Dashboard mode currently doesn't work, as the underlying support in pynetworktables hasn't been implemented yet for the newer FRC Driver Station.

CHAPTER 4

Contents

4.1 Creating a Basic widget

4.1.1 Choosing a unique ID

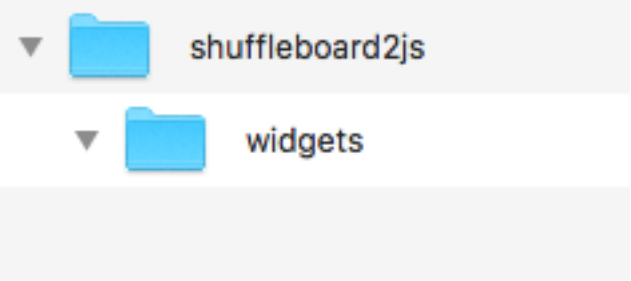
Every shuffleboard2js widget requires a unique ID. Valid IDs can contain lowercase letters, numbers and dashes. Your ID should be related to what your widget does.

Examples of valid IDs: `basicwidget`, `basic-widget2`

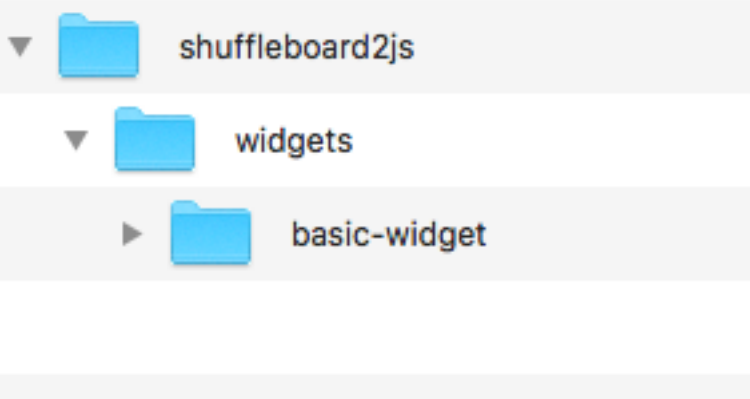
Examples of invalid IDs: `BASICWIDGET`, `basic-widget?!`

4.1.2 Folder structure

In the folder you ran *shuffleboard2js*, you should have a folder structure that looks like this:



Custom widgets need their own folder inside the *shuffleboard2js/widgets* folder. Create a folder named after the ID you chose inside the *shuffleboard2js/widgets* folder. In this example we'll use *basic-widget* as the widget ID:



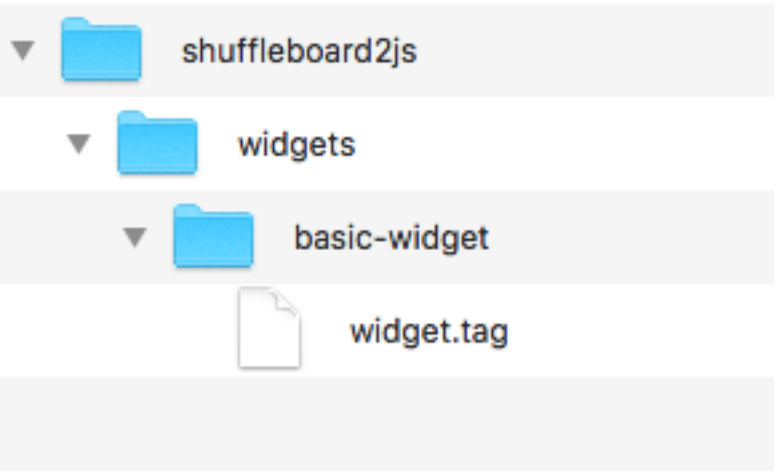
4.1.3 widget.tag

The code for your custom widget goes in a file called `widget.tag`. This file has a special HTML like syntax which you'll learn more about in the *Widget Anatomy* section.

For now create a `widget.tag` file in the *shuffleboard2js/widgets/basic-widget* folder and add the following code to it:

```
<basic-widget>
  <p>This is a basic widget!</p>
  <p>Value: {opts.table}</p>
</basic-widget>
```

Your folder structure should now look like this:



4.1.4 Registering your widget

Your widget doesn't show up automatically in the *shuffleboard2js* interface. To register your widget you need to add an `index.html` file with the following code:

```
<!-- This includes widget.tag into this file -->
<script type="riot/tag" src="widget.tag"></script>
```

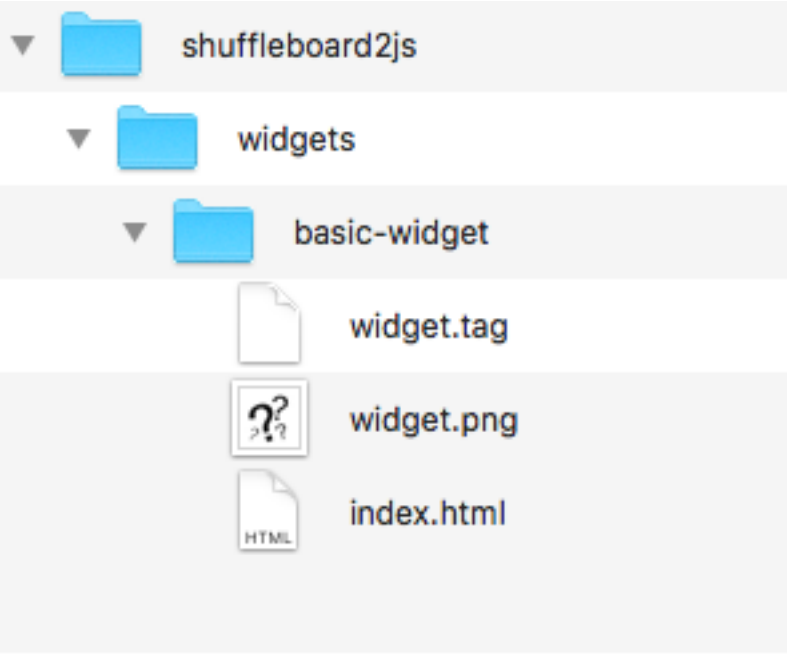
(continues on next page)

(continued from previous page)

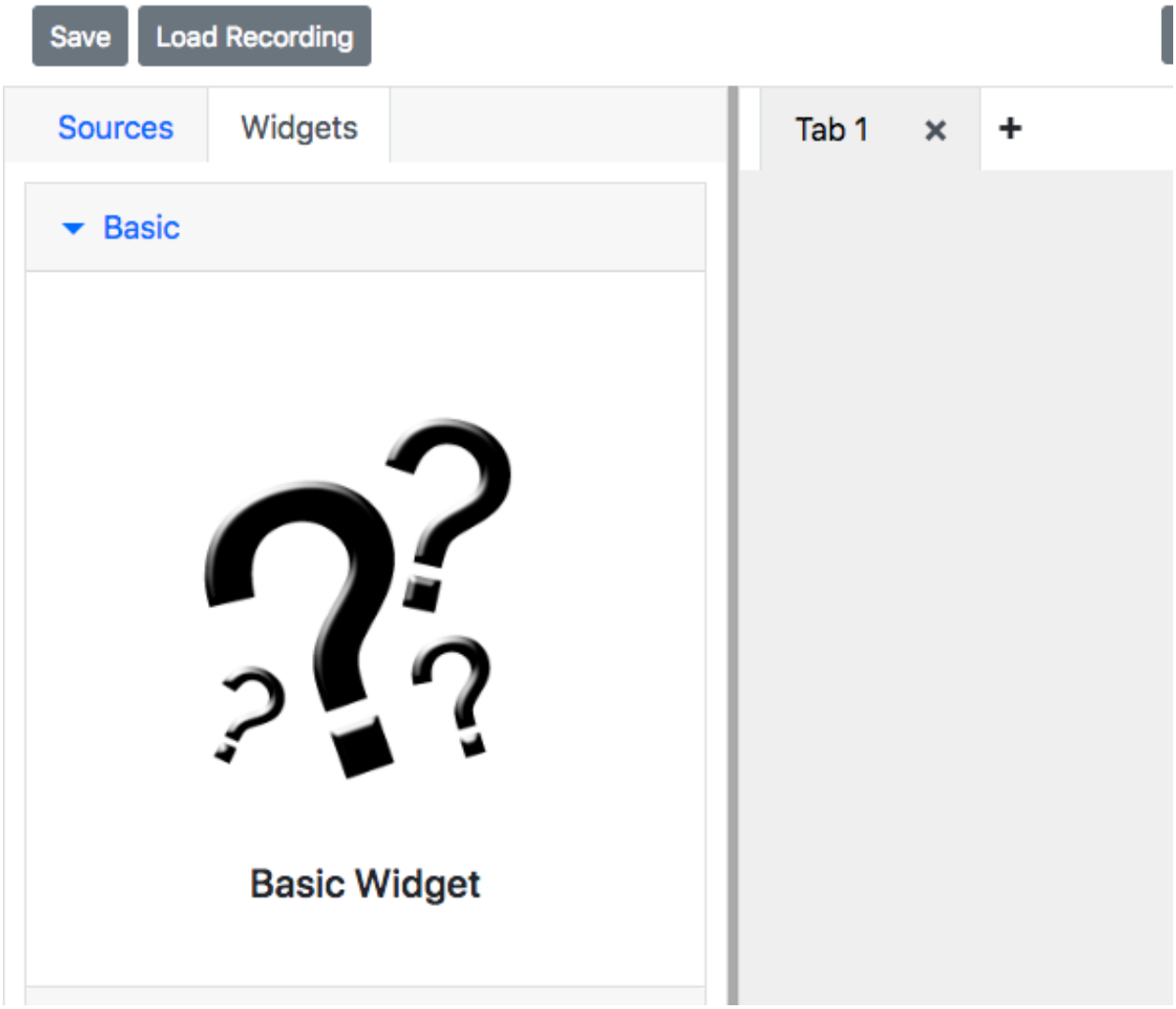
```
<script>
  /**
   * The first parameter dashboard.registerWidget takes
   * is the widget ID. The second parameter is a javascript
   * object used to configure the widget.
   */
  dashboard.registerWidget('basic-widget', {
    label: 'Basic Widget',      // This is the label the widget will take in the␣
↪widget menu
    image: 'widget.png',        // This is the image the widget will show in the␣
↪widget menu
    category: 'Basic',          // This is the category the widget will placed under␣
↪in the widget menu
    acceptedTypes: ['string'],  // These are the types of NetworkTables values you␣
↪can drag onto the widget
    minX: 3,                    // This is the minimum number of x grid spaces the␣
↪widget can take up in the interface
    minY: 3                     // This is the minimum number of y grid spaces the␣
↪widget can take up in the interface
  });
</script>
```

You can optionally put an image in your widget's folder named after whatever you passed into the *image* configuration property passed into the *dashboard.registerWidget* function.

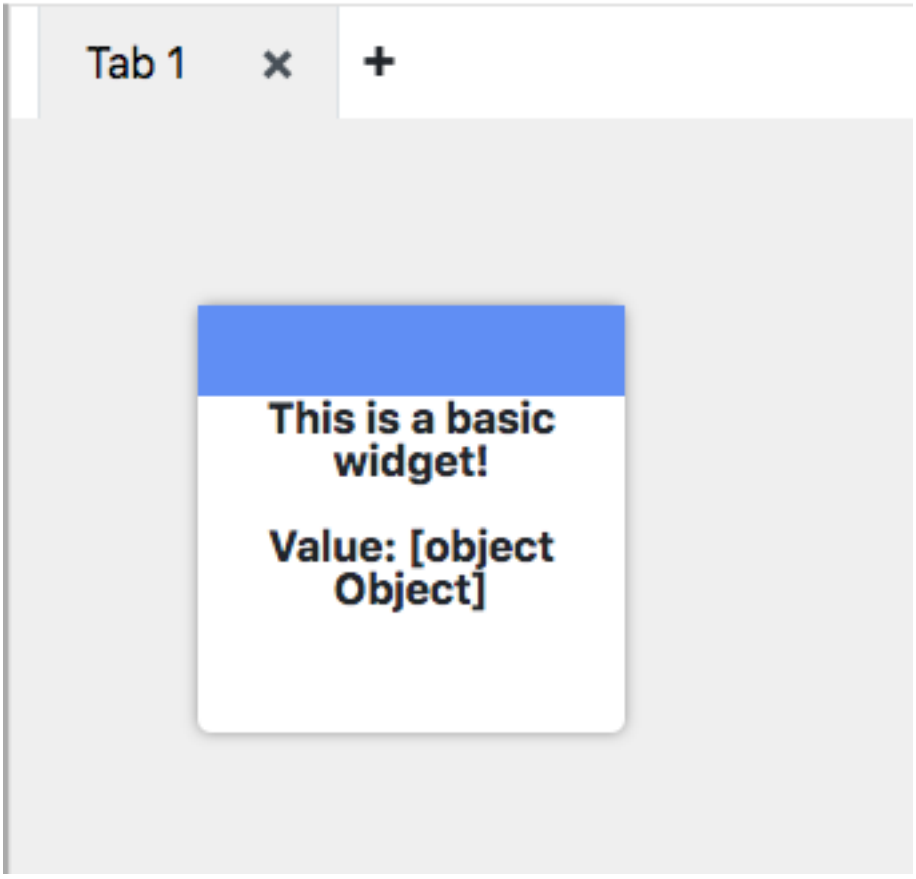
Your folder structure should now look like this:



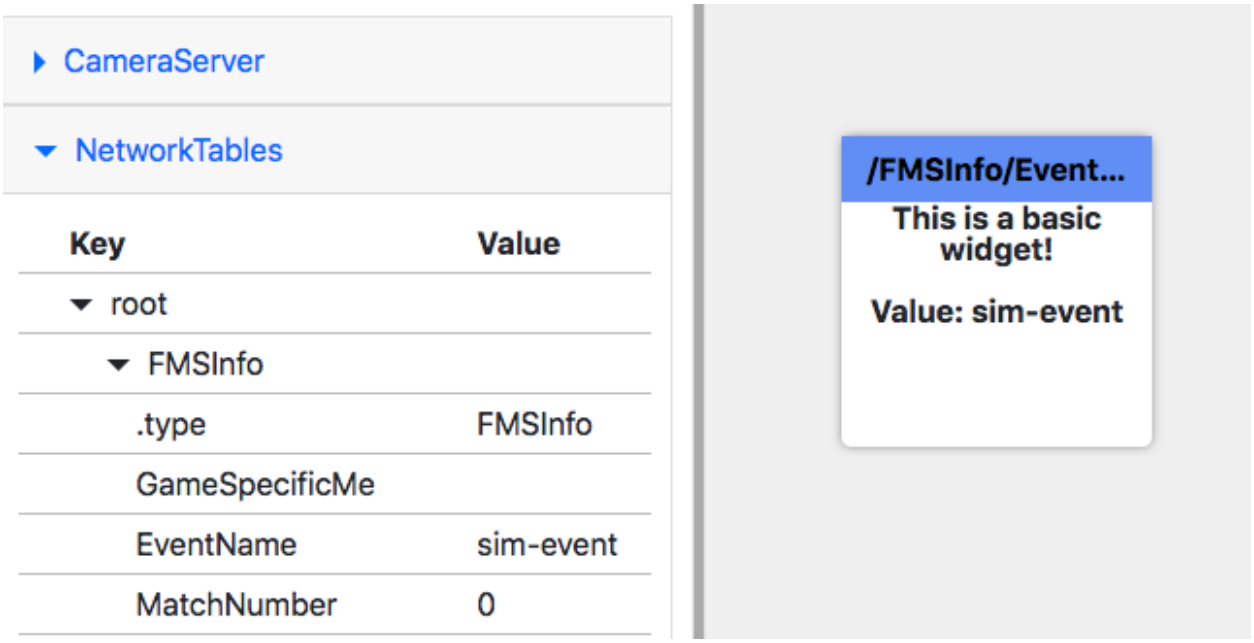
Refresh the interface and your widget should be there:



Dragging a widget onto the interface should look like this:



Drag a NetworkTables source onto the widget of type *string*. The widget should take on the NetworkTable value:



4.2 Widget Anatomy

4.2.1 RiotJS

shuffleboard2js allows you to build custom tags using the RiotJS library. To start learning about how you can create your own custom widgets it's recommended you read RiotJS's guide here.

4.2.2 Setting accepted types when registering your widget

The value of the *acceptedTypes* configuration property passed into the *dashboard.registerWidget* function determines what NetworkTables sources can be dragged onto the widget.

```
dashboard.registerWidget('widget-id', {
  acceptedTypes: ['string'],
  ...
  ...
  ...
});
```

In the code above the *acceptedTypes* property is passed an array with the element *'string'*. This means that only NetworkTables keys with value type *string* can be dragged onto the widget.

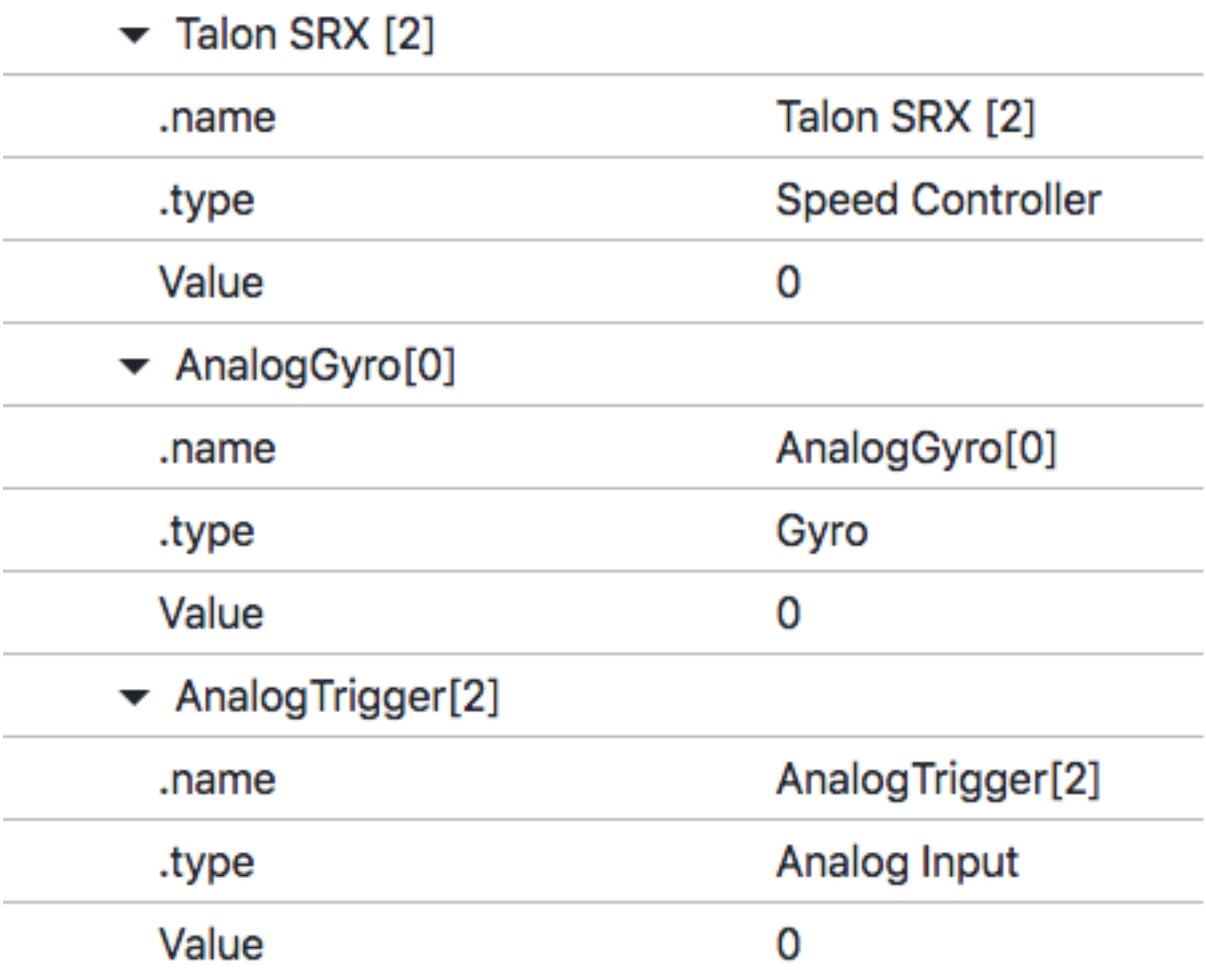
There are two kinds of widget types that can be passed into the *acceptedTypes* property: The primitive NetworkTable types that each individual NetworkTable key has, and types determined by keys that end in *.type*.

This is the list of primitive types *acceptedTypes* can be passed:

- string
- number
- boolean
- array

Here are some examples of types of the *.type* kind:

- Speed Controller
- Gyro
- Analog Input
- Digital Input
- Double Solenoid



4.2.3 Getting NetworkTables data in your widget code

NetworkTables data is accessible from your widget code through `opts.table`:

```
<your-widget-id>
  <p>NetworkTables data: {opts.table}</p>

  <script>
    this.on('update', () => {
      console.log('Networktables data updated:', this.opts.table);
    });
  </script>
</your-widget-id>
```

For example, the `basic-widget` we created in the last section, which accepts NetworkTable values of type *string*, will show whatever NetworkTables value of type *string* is dragged onto the widget:

If the widget's accepted types that are determined by the *.type* key, then `opts.table` will be an object containing all the keys in the subtable dragged onto the widget. For example, take the following widget that accepts types *Gyro* and *Speed Controller*:

```
<basic-widget>
  <p>Name: {opts.table['.name']}</p>
  <p>Type: {opts.table['.type']}</p>
  <p>Value: {opts.table['Value']}</p>
</basic-widget>
```

4.2.4 Updating the widget

Widgets are updated automatically when any NetworkTables values change. To update manually call `this.update()`.

```
<your-widget-id>

  <!-- Widget HTML goes here -->

  <script>
    this.update();
  </script>

</your-widget-id>
```

4.2.5 Styling your widget

To style your widget add a `<style></style>` tag:

```
<your-widget-id>

  <!-- Widget HTML goes here -->

  <style>
    /* CSS goes here */
  </style>

</your-widget-id>
```

4.2.6 Adding a <script></script>

Scripts can be added to your tag by adding a `<script></script>` tag:

```
<your-widget-id>

  <!-- Widget HTML goes here -->

  <script>
    // Your code goes here
  </script>

</your-widget-id>
```

4.2.7 Adding properties

Getting properties

On properties update

If you want to receive updates when the widget's properties are updated, use the `propertiesUpdate` event:

```
<your-widget-id>

  <!-- Widget HTML goes here -->

  <script>

    this.on('update', () => {
      // update event is fired when properties change
    });

    this.on('propertiesUpdate', () => {
      // propertiesUpdate event is also fired, if you want to run code specifically
      // when the the widget's properties change
  });
  </script>

</your-widget-id>
```

Properties modal

4.2.8 Other Events

If you want to receive updates when the widget is resized, use the `resize` event:

```
<your-widget-id>

  <!-- Widget HTML goes here -->

  <script>
    this.on('resize', () => {
      // code goes here
    });
  </script>

</your-widget-id>
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search