Better IoT Product Development Using JavaScript on Microcontrollers

Andy Carle, PhD

Co-Founder, Product & Prototype Engineering

@PrototypingAndy

andy@moddable.com


Stain Treat
2nd Rinse
Auto Soak
Normal/Casual
Bulky/Bedding
Heavy Duty
Whitest Whites
Load Sensing
Wash
Rinse
Spin
42
Door Lock
Express Wash
Cold Clean
Sanitize
Delicates
Delay Start
Clean Washer
Control Lock
POWER
START/PAUSE
Extra Hot/Cold
Hot/Cold
Warm/Warm
Warm/Cold
Cold/Cold
Extra High
High
Medium
Low
No Spin
Heavy
Normal
Light
Loud
Soft
Off
Wash/Rinse
Spin Speed
Soil Level
Chime

Photo: Samsung

User expectations lie somewhere in between:

- High-quality graphics.
- Responsive touchscreen interfaces.
- Seamless cloud connectivity.
- Product-relevant functionality.

At a mass-market price point.

ESP8266

80 MHz 32-bit microcontroller
80 KB RAM
4 MB SPI Flash Storage
Wi-Fi
16x GPIO

QVGA IPS Display

240x320 QVGA
16-bit color
Capacitive multitouch
40 MHz SPI

Total cost of goods $10-$12 at volume of ~1000 units.

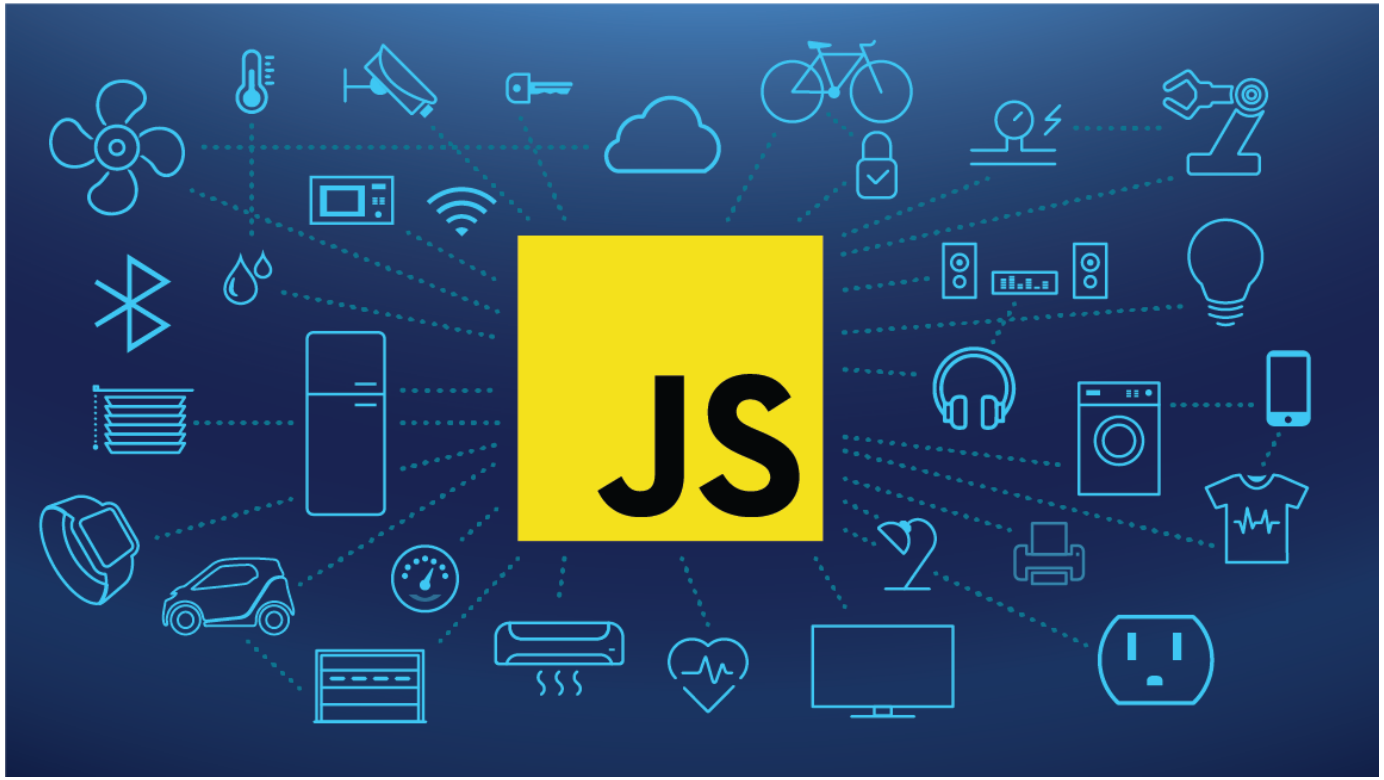
Modern microcontrollers are very capable and inexpensive, but using them comes at a high cost to your software teams.

- Silicon is a constantly moving target.
- Software SDKs from silicon providers are terrible.
- Display and input components can be difficult to source consistently and require specialized drivers.

The answer?
Don’t target individual silicon solutions.
Target a well-designed portable runtime.

- Allows your organization to separate the HMI implementation from the BSP.
- Similar approach to using Android on very expensive hardware, but for mainstream products.

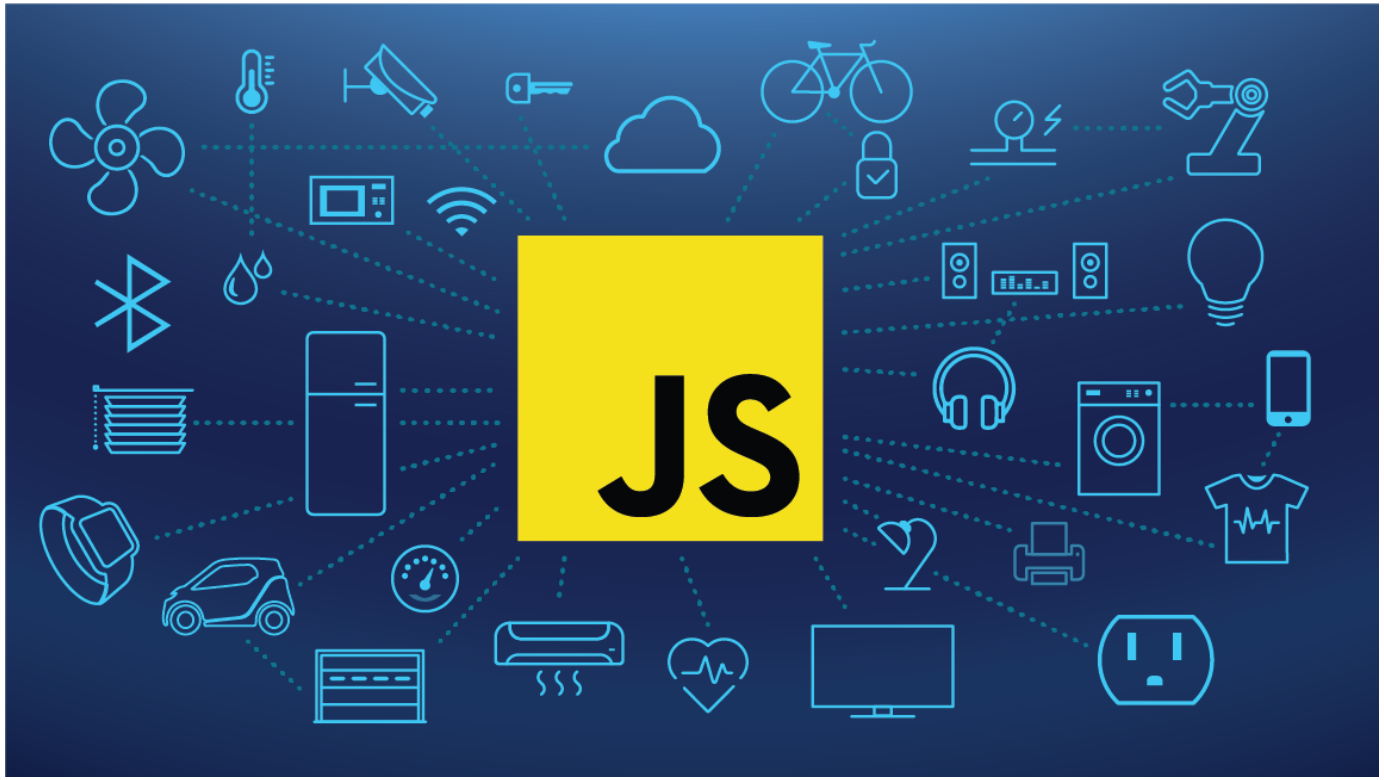
Recent JavaScript engine engineering breakthroughs have made it possible to run full, modern JavaScript on inexpensive microcontrollers.



The XS JavaScript engine is optimized to minimize memory and storage footprint.

- Runs well in as little as 32 KB of RAM.
  - Hundreds of times less RAM than other major engines.
- Minimizes flash usage with clever pre-processing of code and application assets.
- Implements the full ECMAScript 2019 standard.
  - Consistently ahead of many web browser engines.
- Open source: https://github.com/Moddable-OpenSource/moddable/tree/public/xs

Now that it is an option, JavaScript is a natural fit for embedded application development.



Concentrate complexity in the engine and runtime, not in application code.

- The engine and runtime only have to be developed well once and countless JavaScript apps and products can benefit from that work.

It is much easier to write performant code in JavaScript than in C or C++.

- Run loop tuning is taken care of in the engine.
- Common areas of embedded inefficiency are covered by highly-optimized JavaScript modules:
  - graphics libraries
  - networking stack
  - file system operations

Working in JavaScript is inherently more secure than starting from scratch in native languages.

- Security-critical native code only needs to be written once and can be inspected all in one place.
- It is impossible to write common security errors like buffer overflows and accessing uninitialized memory in JavaScript.
- Application code is more concise and less error-prone.

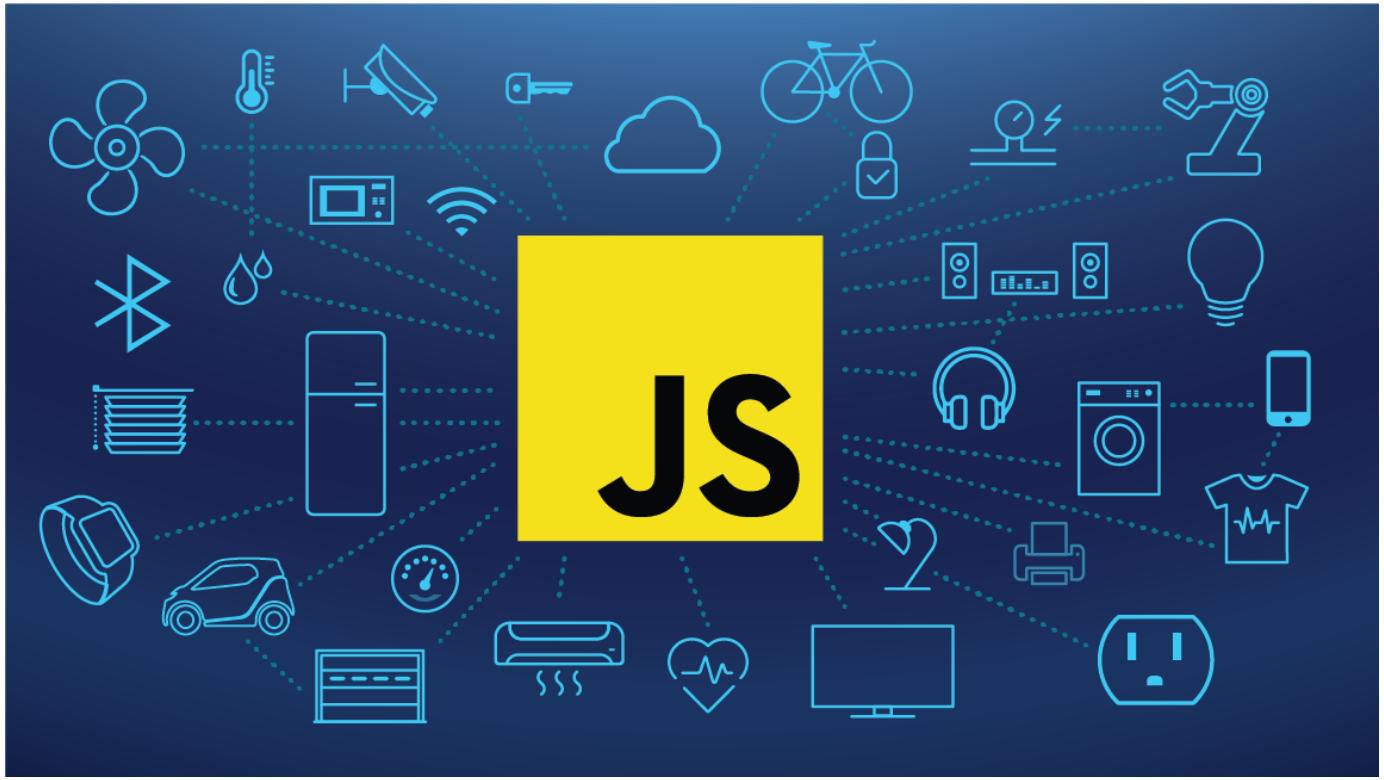
JavaScript has language features that are perfect for IoT application development.

- JavaScript has excellent support for asynchronous and event-based design patterns that fit well with sensor- and network-driven devices.
- Objected oriented languages work well for user interface construction.
- Networking is built in and parsing JSON data from cloud services is built into the language.

Building apps in JavaScript is easier for developers than working in native languages.

- Developers can stop worrying about implementation details like memory management and building fundamental runtime components.
- Instead, they can focus on making great products.

Building embedded applications with JavaScript makes entire organizations work better.



Because JavaScript is much easier to work with, it is much easier to find JavaScript developers than traditional embedded developers.

- Recruiting traditional embedded engineers is becoming harder every day.
- The massive popularity of JavaScript on the web and servers (i.e. Node.js) has created a huge pool of potential embedded JavaScript developers.

JavaScript code is portable across multiple products and platforms.

- Apps written in JavaScript for one device can easily be ported to another as business needs change.
- Target devices can be radically different without needing any changes to the application code.

Scripting enables radically different product design processes and reduces overall product development cycles.

- Ease of development allows rapid prototyping of products with production-quality implementations.
- Desktop simulation is game-changing for embedded developers and removes build/deploy cycles from day-to-day development entirely.

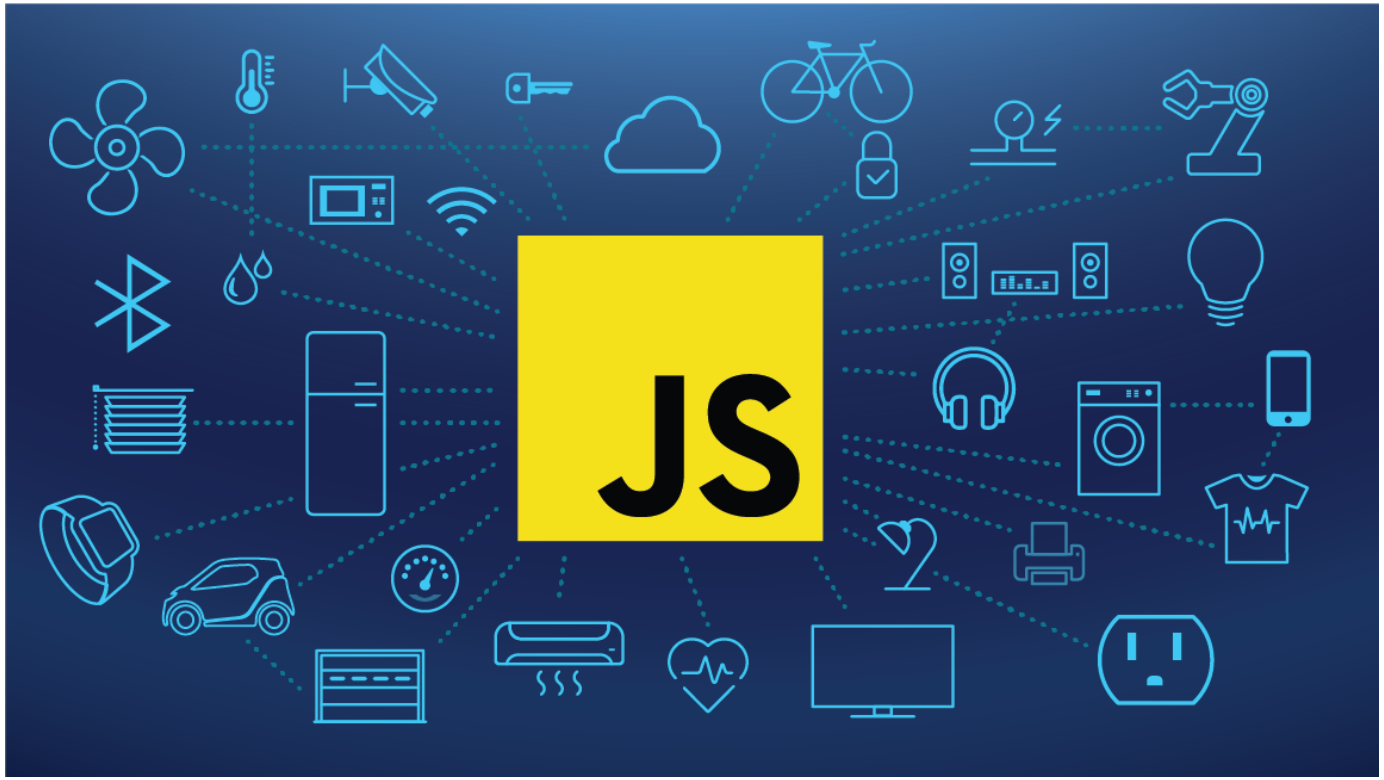
JavaScript components are reusable and easy to maintain.

- JavaScript has a language feature to support modules, which cleanly encapsulate functionality.
- Useful components, such as UI elements or business logic, can be maintained independently and used across many products.

Building IoT applications in JavaScript allows your organization to work in the context of a massive open source effort.

- Thousands of JavaScript modules exist in open source repositories, many of which are relevant for IoT product development.
- Ecma TC53 is working to standardize a core set of modules for embedded systems to give manufacturers and silicon vendors a common target for support.

You can build beautiful and responsive user interfaces in JavaScript using open source modules.



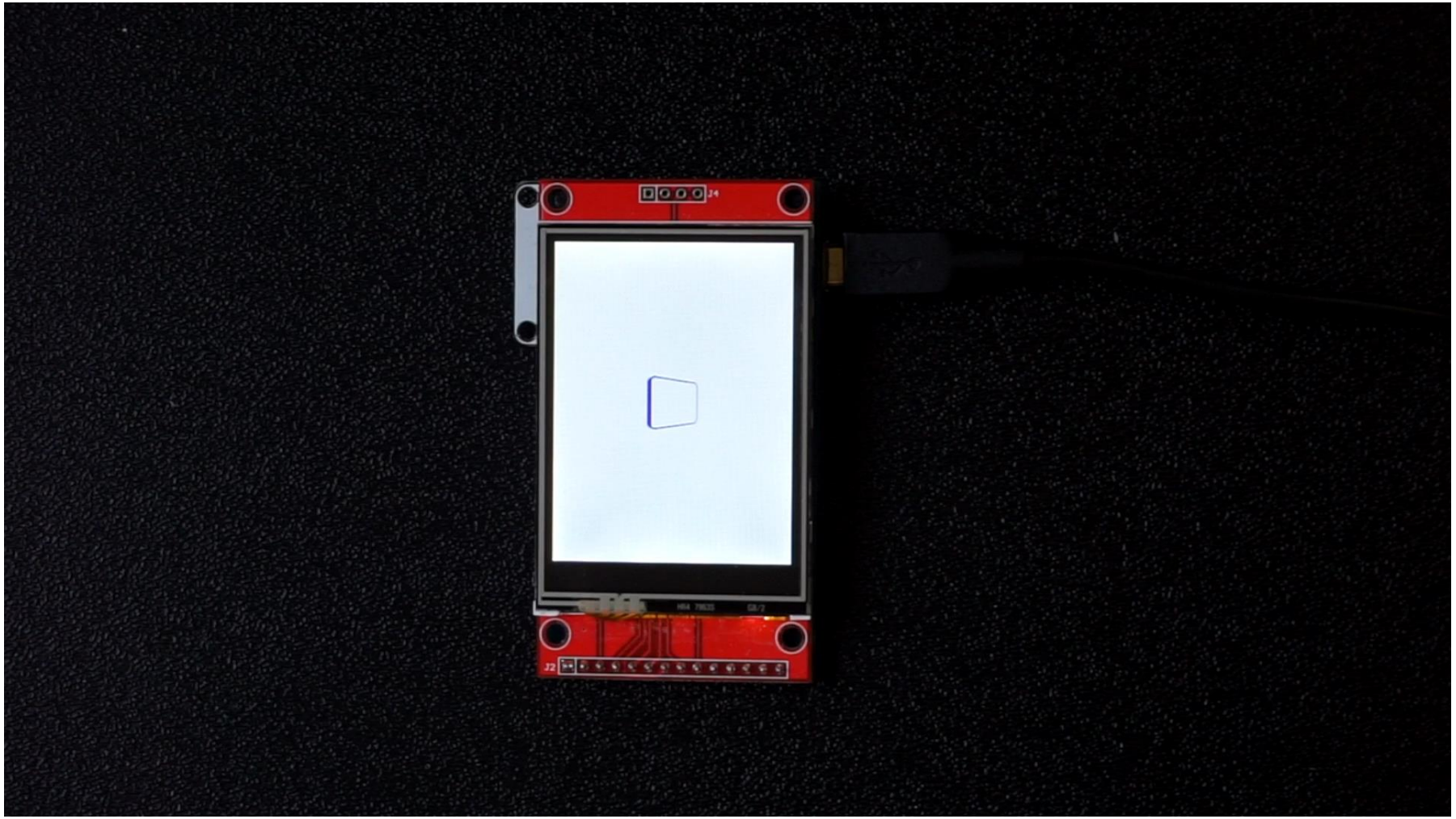
The **Commodetto** graphics rendering engine delivers high frame rates on extremely inexpensive hardware.

- Capable of efficiently rendering a single scanline at a time, so no frame buffer is required on the microcontroller.
- Includes asset loaders for images and fonts.
- Pixel Output modules deliver rendered pixels to displays, files, and network streams.
- Open source: https://github.com/Moddable-OpenSource/moddable/tree/public/modules/commodetto

See these examples animated at:
https://github.com/Moddable-OpenSource/moddable/blob/public/examples/commodetto/readme.md

The **Piu** user interface framework lets developers construct a hierarchy of objects to define layouts and contents.

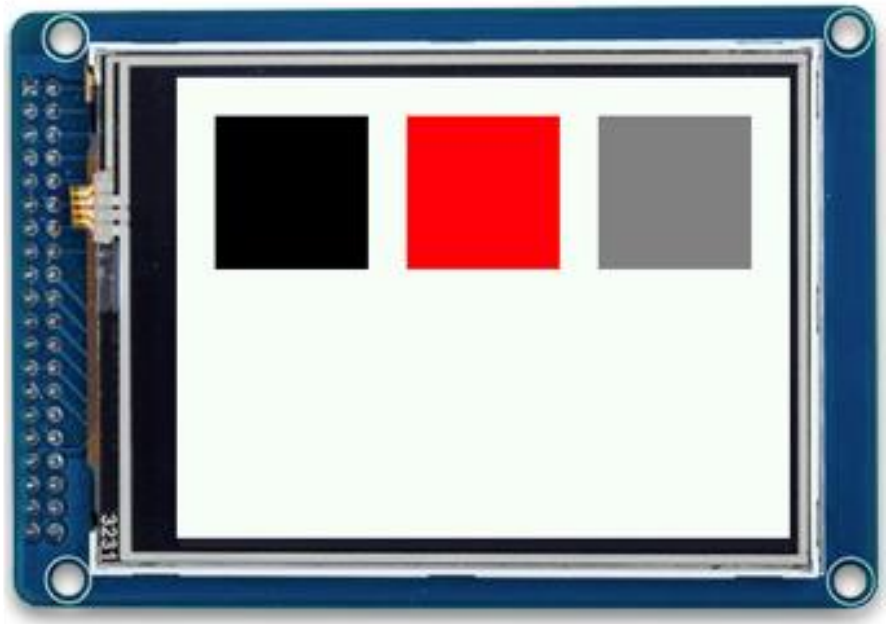
- Unusual for embedded development, which often uses simple 2D graphics APIs.
- Rich feature set makes for easy app development:
  - Cascading styles
  - Event-driven behaviors
  - Robust font support
  - Built-in tools for animations, transitions, and responsive layouts
- Open source: https://github.com/Moddable-OpenSource/moddable/tree/public/modules/piu



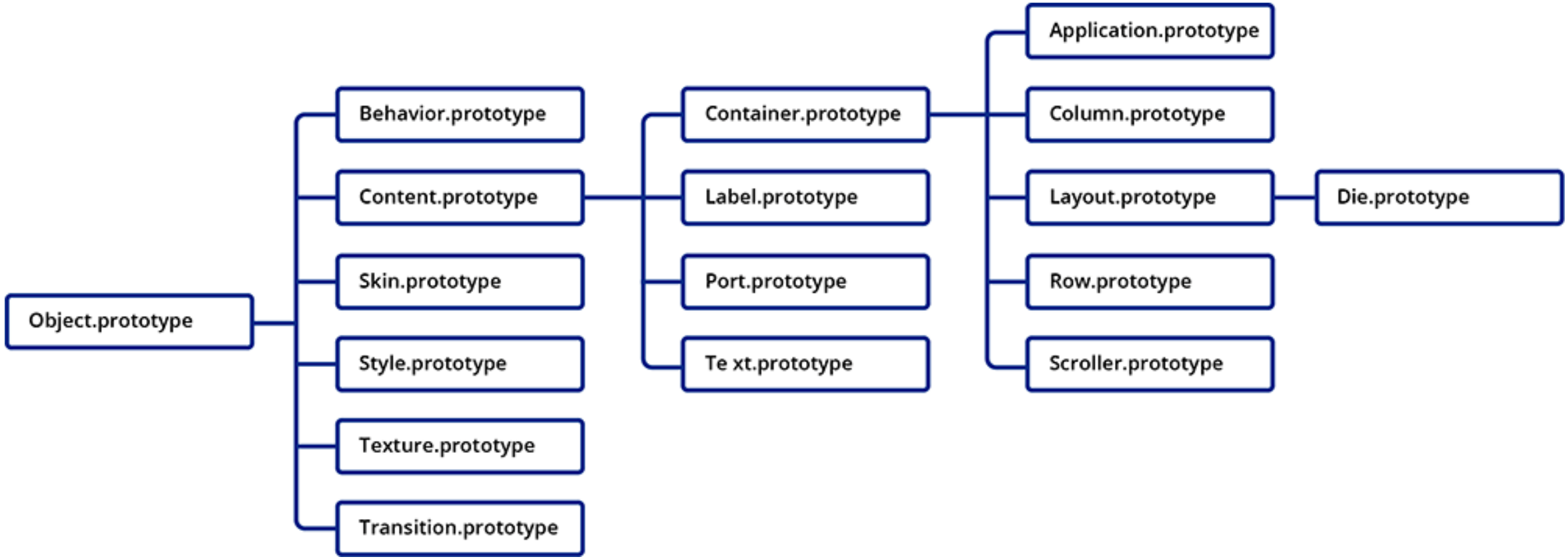
Watch this video at https://www.youtube.com/watch?v=JhRyMoUbmXA

Piu Object Construction

```
let multiColoredSkin = new Skin({ fill: ["black", "white", "red"] });
let blackContent = new Content(null, {
    top: 20, left: 20, width: 80, height: 80,
    skin: multiColoredSkin, state: 0,
});
let redContent = new Content(null, {
    top: 20, left: 120, width: 80, height: 80,
    skin: multiColoredSkin, state: 2
});
let grayContent = new Content(null, {
    top: 20, left: 220, width: 80, height: 80,
    skin: multiColoredSkin, state: 0.5
});
application.add(blackContent);
application.add(redContent);
application.add(grayContent);
```

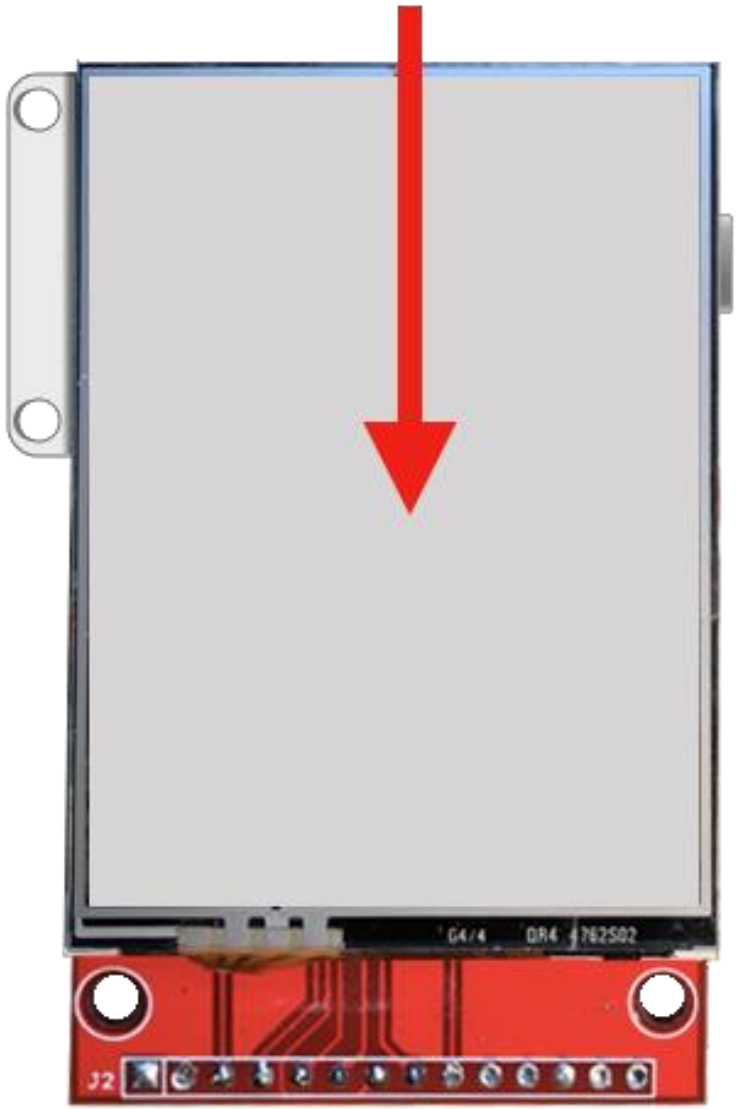


Piu Prototype Hierarchy




Heat to 75°
72°
J2

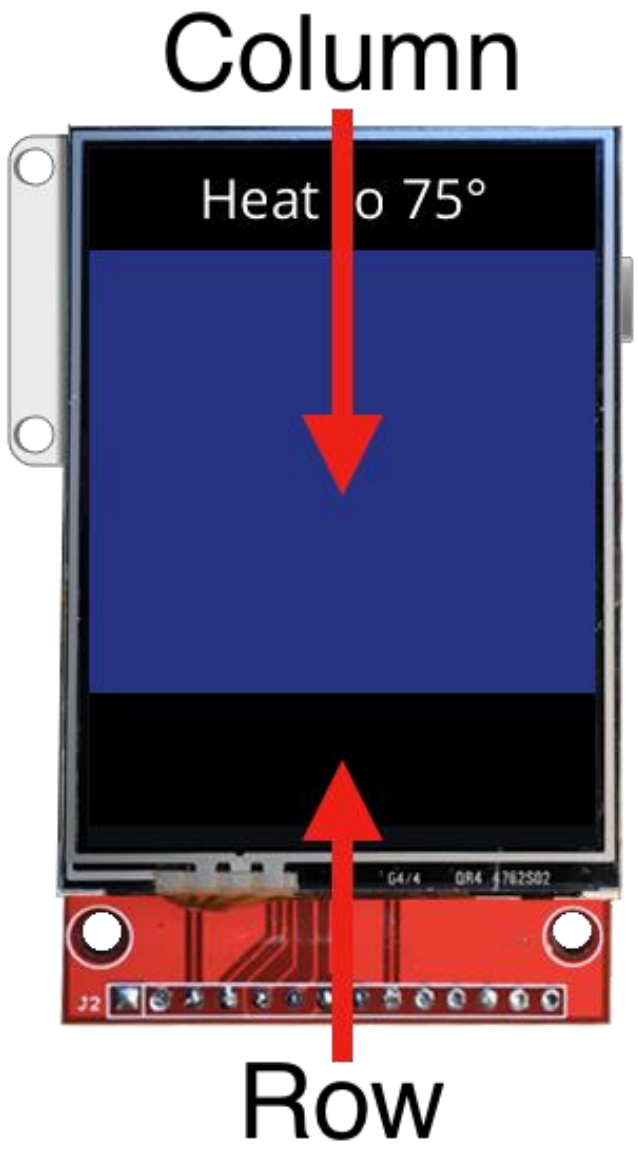
Column



```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0
}));
```

Label


```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0,
  contents: [
    Label($, {
      left: 0, right: 0, height: 40, string:"Heat to 75°",
      style: OpenSans26
    }),
  ],
}));
```

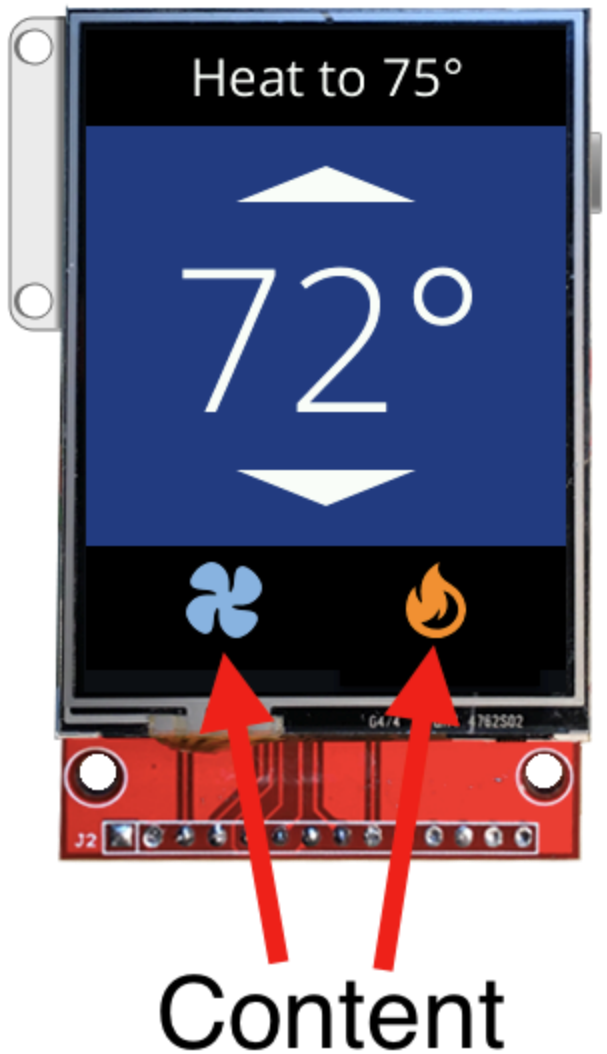



```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0,
  contents: [
    Label($, {
      left: 0, right: 0, height: 40, string:"Heat to 75°",
      style: OpenSans26
    }),
    Column($, {
      top: 0, bottom: 0, left: 0, right: 0, skin: blueSkin
    }),
    Row($, {
      left: 0, right: 0, height: 62, skin: blackSkin
    })
  ],
}));
```

Content

Label

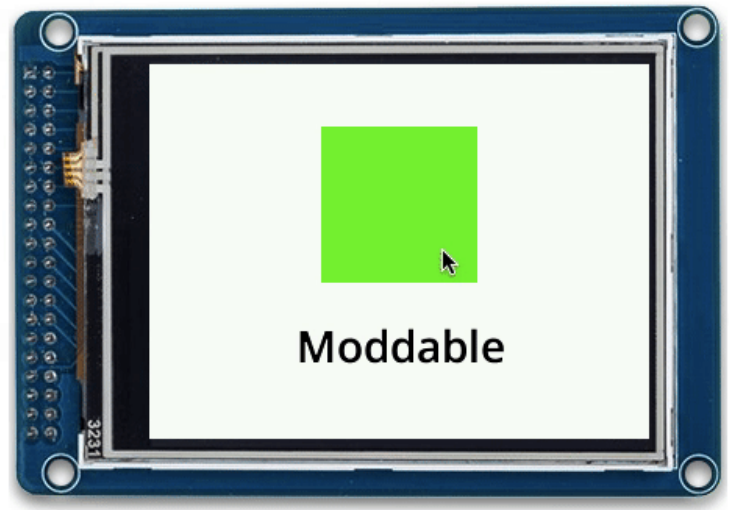
```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0,
  contents: [
    Label($, {
      left: 0, right: 0, height: 40, string:"Heat to 75°",
      style: OpenSans26
    }),
    Column($, {
      top: 0, bottom: 0, left: 0, right: 0, skin: blueSkin,
      contents: [
        Content($, {
          active: true, top: 20, skin: upSkin
        }),
        Label($, {
          top: 0, bottom: 0, style: OpenSans100
        }),
        Content($, {
          active: true, bottom: 20, skin: downSkin
        }),
      ]
    }),
    Row($, {
      left: 0, right: 0, height: 62, skin: blackSkin
    })
  ],
}));
```

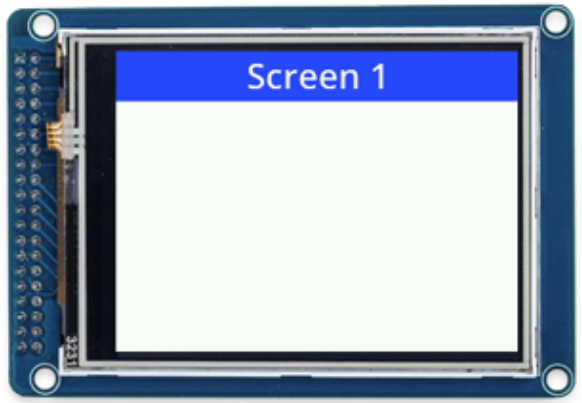



```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0,
  contents: [
    Label($, {
      left: 0, right: 0, height: 40, string:"Heat to 75°",
      style: OpenSans26
    }),
    Column($, {
      top: 0, bottom: 0, left: 0, right: 0, skin: blueSkin,
      contents: [
        Content($, {
          active: true, top: 20, skin: upSkin
        }),
        Label($, {
          top: 0, bottom: 0, style: OpenSans100
        }),
        Content($, {
          active: true, bottom: 20, skin: downSkin
        }),
      ]
    }),
    Row($, {
      left: 0, right: 0, height: 62, skin: blackSkin,
      contents: [
        Content($, { skin: fanSkin, left: 50,
                     top: 8, width: 40, height: 40}),
        Content($, { skin: heatSkin, right: 50,
                     top: 8, width: 40, height: 40})
      ]
    })
  ],
}));
```

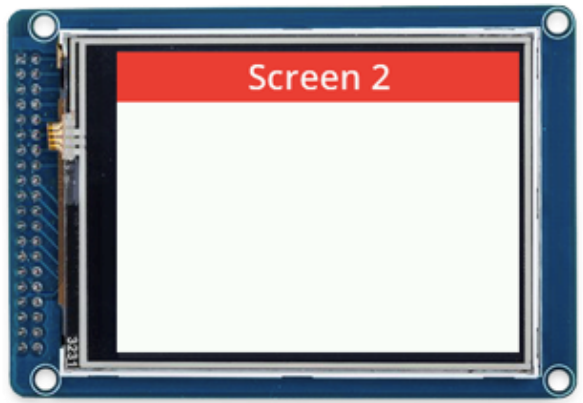

```
const DefaultHomeScreen = Column.template($ => ({
  top: 0, bottom: 0, left: 0, right: 0,
  contents: [
    Label($, {
      left: 0, right: 0, height: 40, string:"Heat to 75°",
      style: OpenSans26
    }),
    Column($, {
      top: 0, bottom: 0, left: 0, right: 0, skin: blueSkin,
      contents: [
        Content($, {
          active: true, top: 20, skin: upSkin
        }),
        Label($, {
          top: 0, bottom: 0, style: OpenSans100
        }),
        Content($, {
          active: true, bottom: 20, skin: downSkin
        }),
      ]
    }),
    Row($, {
      left: 0, right: 0, height: 62, skin: blackSkin,
      contents: [
        Content($, { skin: fanSkin, left: 50,
                     top: 8, width: 40, height: 40}),
        Content($, { skin: heatSkin, right: 50,
                     top: 8, width: 40, height: 40})
      ]
    })
  ],
}));
```

```
const textStyle = new Style({ font:"semibold 28px Open Sans", color:"black" });
const whiteSkin = new Skin({ fill: "white" });
const blueSkin = new Skin({ fill: "blue" });

class ButtonBehavior extends Behavior {
    onTouchBegan(content) {
        content.state = 1;
    }
    onTouchEnded(content) {
        content.state = 0;
        content.next.delegate("onButtonTapped");
    }
}

class LabelBehavior extends Behavior {
    onButtonTapped(label) {
        if (label.string.length === 0) label.string = "Moddable";
        else label.string = label.string.slice(1);
    }
}

const SampleApp = Application.template($ => ({
    top: 0, bottom: 0, left: 0, right: 0, skin: whiteSkin,
    contents: [
        Content($, {
            active: true, top: 40, height: 100, width: 100,
            skin: new Skin({ fill: ["#66ff33", "#0066ff"] }),
            Behavior: ButtonBehavior
        }),
        Label($, {
            bottom: 40, style: textStyle, string: "Moddable",
            Behavior: LabelBehavior
        })
    ],
}))

export default new SampleApp({}, {touchCount: 1});
```

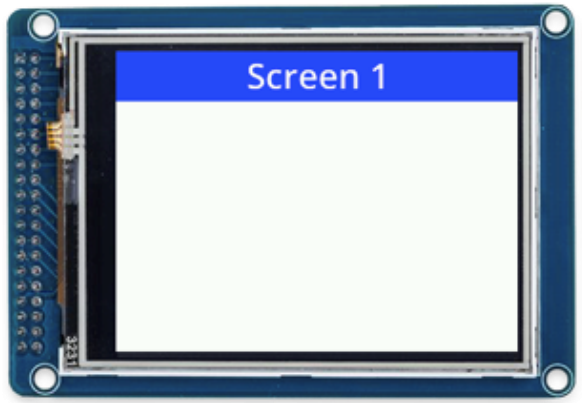




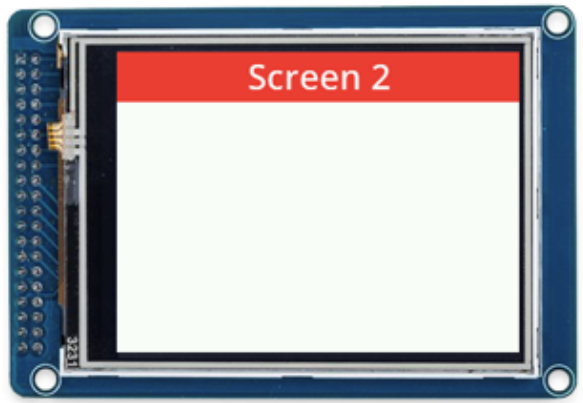

```
let screen1 = new Column(null, {
   top: 0, bottom: 0, left: 0, right: 0,
   skin: new Skin({ fill: "blue" }),
   contents: [
      Label(null, {
         top: 0, height: 40, left: 0, right: 0,
         string: "Screen 1"
      }),
      Content(null, {
         top: 0, bottom: 0, left: 0, right: 0,
         skin: new Skin({ fill: "white" })
      }),
   ]
});
```




```
let screen2 = new Column(null, {
   top: 0, bottom: 0, left: 0, right: 0,
   skin: new Skin({ fill: "red" }),
   contents: [
      Label(null, {
         top: 0, height: 40, left: 0, right: 0,
         string: "Screen 2”,
      }),
      Content(null, {
         top: 0, bottom: 0, left: 0, right: 0,
         skin: new Skin({ fill: "white" })
      }),
   ]
});
```

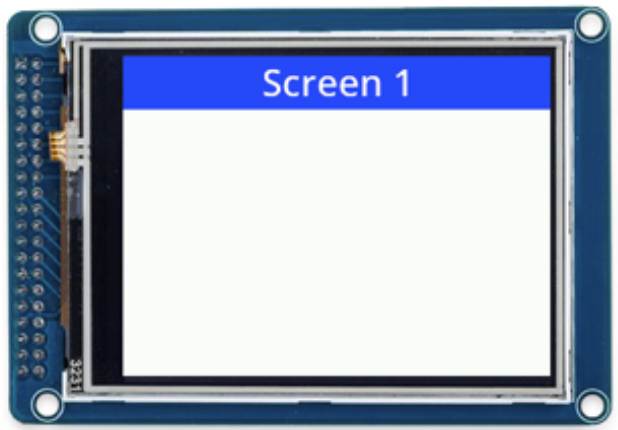



```
let screen1 = new Column(null, {
   top: 0, bottom: 0, left: 0, right: 0,
   skin: new Skin({ fill: "blue" }),
   contents: [
      Label(null, {
         top: 0, height: 40, left: 0, right: 0,
         string: "Screen 1"
      }),
      Content(null, {
         top: 0, bottom: 0, left: 0, right: 0,
         skin: new Skin({ fill: "white" })
      }),
   ]
});
```

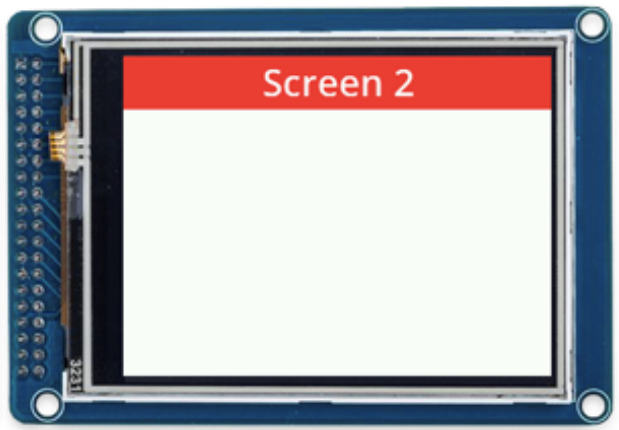



```
let screen2 = new Column(null, {
   top: 0, bottom: 0, left: 0, right: 0,
   skin: new Skin({ fill: "red" }),
   contents: [
      Label(null, {
         top: 0, height: 40, left: 0, right: 0,
         string: "Screen 2”
      }),
      Content(null, {
         top: 0, bottom: 0, left: 0, right: 0,
         skin: new Skin({ fill: "white" })
      }),
   ]
});
```

```
let BasicScreen = Column.template($ => ({
   top: 0, bottom: 0, left: 0, right: 0,
   skin: new Skin({ fill: $.headerColor }), style: sampleStyle,
   contents: [
      Label(null, {
         top: 0, height: 40, left: 0, right: 0,
         string: $.title
      }),
      Content(null, {
         top: 0, bottom: 0, left: 0, right: 0,
         skin: new Skin({ fill: "white" })
      }),
   ]
}));
```




```
let screen1 = new BasicScreen({
    title: "Screen 1",
    headerColor: "blue"
});
```




```
let screen2 = new BasicScreen({
    title: "Screen 2",
    headerColor: "red"
});
```

JavaScript can radically improve the way your organization develops HMI software.

- Recent JavaScript engine engineering breakthroughs have made it possible to run full, modern JavaScript on inexpensive microcontrollers.
- Now that it is an option, JavaScript is a natural fit for embedded application development.
- Building embedded applications with JavaScript makes entire organizations work better.
- You can build beautiful and responsive user interfaces in JavaScript using open source modules.

Get started with the Moddable SDK:

- Open source: https://github.com/Moddable-OpenSource/moddable
- Start on the simulator and then move to Moddable development boards, available at Moddable.com

Moddable One

Moddable Two

Moddable Three

Scripts & Things

Meets monthly in Downtown Palo Alto.

https://www.meetup.com/Scripts-and-Things

Better IoT Product Development Using JavaScript on Microcontrollers

Andy Carle, PhD

Co-Founder, Product & Prototype Engineering

@PrototypingAndy

andy@moddable.com