# Mesop Custom Components

*Note: this doc is publicly accessible.*
https://github.com/google/mesop/issues/95

## Motivation

Mesop provides dozens of built-in components and we plan to build more, but there's some components which shouldn't belong in the core Mesop framework because they aren't general enough, but at the same time we should support them because they are valid use cases.

### Types of use cases

- **Third-party JS libraries** - In particular, there's been quite a few requests to support interactive visualizations using libraries like plotly.js, altair, bokeh, etc. The problem with visualization libraries is that there's many options with significant usage and various tradeoffs. In addition, each visualization library is rather large (e.g. 1MB+) so we'd want to avoid loading them unless they were actually used by the Mesop app (currently, Mesop naively loads all components JS code). Outside of visualization libraries, there are lots of useful third-party libraries with JS such as pygwalker which could provide useful functionality for Mesop developers, but are unlikely to meet our bar to qualify in the core framework.
    - **Headless JS libraries** - There's a distinct set of use cases where users want to run custom JavaScript (e.g. analytics library, call an API) and in many cases, it's critical that the JavaScript is run in the main execution context.
- **Rich client-side interactivity** - Custom components can support stateful interactions without a network round trip, which is a key drawback with Mesop's architecture.
- **Experimental browser APIs** - Although Mesop has a goal of supporting standard web platform APIs, browser vendors will often provide experimental APIs (example). Mesop won't support these experimental APIs because by definition they aren't stable and can break with future browser versions.

# High-level Options

## 1. Support native web components (Recommended)

This option is about supporting native web components into Mesop.

All modern browsers support the core web components standards (e.g. custom elements, HTML templates and shadow DOM).

**Pros:**

- **Web Components are supported natively in the browser** - Unlike picking a specific JS framework, there's basically no risk that web components will be deprecated because they are a widely supported web standard. In addition, because it's supported natively, theoretically the overhead of using web components is minimal because you can write one in vanilla JS (however, see below, as we expect this to be uncommon).

**Cons:**

- **Web Components aren't a popular direct authoring format** - Although web components have a sizable adoption by websites, it's fairly uncommon for developers to write web components without any libraries/frameworks because the APIs are more low-level and not very easy to use.
    - I don't think this is a significant con because we are using web components as an intermediary format, and we expect people to author their web components using a library/framework.

## 2. Support framework-specific web components

Because Mesop itself is built on Angular and heavily depends on Angular Material components for many of its built-in native components, we initially explored supporting custom Angular components in google3. Although we have a working prototype in google3, there's several issues with supporting a similar system in the open-source world.

- **Angular is only one of many options** - Google, by policy, only recommends one of two UI frameworks. The open-source world, in contrast, has much more variety, and other frameworks like React have much higher adoption.

- **Risk to future Angular upgrades** - Based on my understanding, Angular apps should be compiled in a single pass (e.g. the compiler must be aware of all of the FE code) and can then do module bundling and splitting. Because of this, all the Angular components, including the built-in Mesop components and custom components, would be compiled together relying on a single version of the Angular runtime. As Angular migrates and evolves its APIs (e.g. zoneless), it may become very difficult to upgrade the core Angular framework Mesop uses because of the tight coupling to runtime versions (e.g. if a popular community component relies on zone.js, then we can't upgrade Angular in Mesop without causing a schism in the community).

## 3. Iframe-as-components

Other Python UI frameworks like Streamlit have used iframes for custom components. Every custom component instance is rendered in an iframe, which ensures isolation between the custom component and the rest of the app.

**Pros:**

- **Provides framework/runtime isolation.** Because every component instance is isolated from everything else, there's no issue with running multiple versions of the same framework as long as they're isolated on different frames.

**Cons:**

- **Iframe overhead** - iframes have a non-trivial overhead for browsers and creating a large number of custom component instances can cause a performance issue with this approach.
- **Breaks some headless JS use cases** - see headless use cases, running inside an iframe is problematic for some of these use cases.
- **Non-intuitive layouts** - being in an iframe makes layouts non-intuitive because the browser cannot automatically layout content across frames, so you need to manually "relayout" (e.g. Streamlit has a helper function to update the iframe height).

# Detailed Design (Native Web Components)

## Python API

Each Mesop UI component needs to have a Python API. Mesop will provide a small helper function to define custom components:

```python
import mesop.labs as mel

@dataclass
class IncrementEvent:
  key: str
  value: int


@mel.web_component(js_path=js_file_path)
def foo_custom_component(
  *,
  value: int,
  on_increment: Callable[[IncrementEvent], Any],
  key: str | None = None,
):
  mel.add_web_component(
    name="foo-component",
    key=key,
    events={
      "increment-event": on_increment,
    },
    properties={
      "value": value,
    },
  )
```

## JS / Web Components API

**Inputs:**

- `${properties}` - these are component-specific properties to configure the component instance (these are the equivalent of “props” in React components or input properties in Angular components)
    - The property names can be anything the component developer specifies except for reserved names like “style”.
- `${events}` - these are properties where the key is the custom event name and the value is the handler id.

**Outputs:**

- Each web component can dispatch a custom event which subclasses a MesopEvent (which itself is a kind of CustomEvent).

## Example JS custom component definition

```javascript
class FooComponent extends LitElement {
  static properties = {
    value: {type: Number},
    style: {type: String},
    incrementEvent: {type: String},
  };

  constructor() {
    super();
    this.value = 0;
    this.style = '';
    this.incrementEvent = '';
  }

  static styles = css`
```

```
    :host {
      display: block;
    }
  `;

  render() {
    return html`
      <div class="container" style="${this.style}">
        <span>Value: ${this.value}</span>
        <button id="increment-btn" @click="${this._onIncrement}">
          Increment
        </button>
      </div>
    `;
  }

  _onIncrement() {
    this.dispatchEvent(
      new CustomEvent('mesop-event', {
        detail: {
          payload: {value: this.value + 2},
          handlerId: this.incrementEvent,
        },
        bubbles: true,
      }),
    );
  }
}

customElements.define('foo-component', FooComponent);
```

## Lazy Loading

https://github.com/google/mesop/issues/232

# Prototype Implementation

**Using Lit:** https://github.com/google/mesop/pull/416

The following shows an example of defining a simple “counter” component which increments by 2.

# Framework integrations

Because web frameworks are heavily used and it’s not common to write frameworkless JavaScript, we should try to have framework integrations, or at the least guides, that makes it easy to write custom components in the various popular frameworks.

## Lit (Recommended)

Lit is a lightweight library that provides a convenient, React-like API for building web components and has been security hardened.

I propose having Lit be the “default” (but not exclusive) way of writing Mesop custom components for the following reason:

- **First-class web component support** - Lit is web component-centric and was designed around the web components standards/API. Some of the other frameworks have limited web component support, including limited documentation and limitations.
- **Lightweight** - Lit core is <7KB gzipped.
- **Secure** - Lit embraces modern web security best practices like Trusted Types and is safer than writing web components in vanilla JS.

As an added bonus, Lit is a Google project and their maintainers have been helpful in reaching out to us about this topic.

- Notes from justinfagnani@

**Questions:**

- If different Lit components (in separate script modules) load Lit at different (major) versions, is there an issue? **Short answer (from Lit team): no.**
- Should we recommend developers to import the `core` or `all` bundle? Doesn't really matter since it'll end up being single file bundles, but picking `all` seems reasonable. Developers can still pick the other one, but encouraging the ecosystem to pick one, could help minimize network requests.

## React

It should be possible to have a custom element class wrap a React component. Justin Fagnani is willing to build some examples as part of the Lit React integration library.

## Angular

Angular has supported custom elements for a long time with Angular Elements so it's technically possible to write custom components in Angular and use them in Mesop, however this would likely become problematic because it would create a tight coupling to a specific version of the Angular runtime.

# Web Security

Mesop tries to adhere to many web security best practices and we'd like to continue to make Mesop apps "secure-by-default", however with custom components there's more room for vulnerabilities to creep in.

## Innocent component developers

Assuming the component developer doesn't have malicious intents, the two main risks with custom components are the following:

- The user supplies a value (e.g. a string) which is used as the JS path for the custom component and becomes a means to execute malicious JS code.
    - **Mitigation:** Ensure that the value supplied for JS path is a static value (e.g. it must not be based on user/session-based values).
- The user could supply a value (e.g. text input) which is used as an inner HTML value for the custom component and becomes a means to execute malicious JS code.

    - **Mitigation:** We will recommend custom component developers to use a library like Lit which has its own Trusted Types policy (which will be allow-listed by Mesop), otherwise by default Angular's Trusted Types policy will block unsafe access to innerHTML.

**Note:** In the full scenario, the threat could be more elaborate where the user has manipulated a database value, query param value, and then the malicious value affects other users.

### Compromised component developers

In the broader open-source ecosystem, we (core Mesop team) will want to promote custom components from the community which a) are useful and b) do not present a significant security risk. It's unclear how we can do this at scale, but initially we should have a gallery/showcase of custom components and link to GitHub repos.

## Todos

1. Example of importing relative files
2. Build an example with composability

## Roadmap

Because custom components is both critical and complex, I propose splitting it into two milestones so we can get feedback from the community and iterate before committing to a final, long-term solution.

### MVP (Mesop Labs)

**Goal:** The goal is to unblock some current use cases and gather feedback on whether this is something we can commit to long-term. We will try to have a *somewhat* stable API during this phase, but we should be willing to iterate (e.g. make breaking changes to fix edge cases).

**Requirements:**

- ☑ ~~Complete basic implementation: https://github.com/google/mesop/pull/416~~
    - ☑ ~~Make sure attributes (boolean, reflected) work properly.~~
    - ☑ ~~Build an example of importing a third-party library (e.g. Plotly.js)~~

- ☑ ~~Support importing relative files~~
- ☑ ~~Composable components using slot~~
  - ☑ ~~Mesop built-in component in web component (this would be more complicated, I think since we'd need to compile to Angular Elements)~~
  - ☑ ~~Web component in web component~~
- ☑ ~~Ensure Trusted Types (w/ Lit) works well~~

- ☑ ~~CustomEvent~~
- ☑ ~~Support Bazel usage (and downstream)~~
- ☑ ~~Support in pip install mesop~~
- ☑ ~~Create examples:~~
  - ☑ ~~quickstart/counter~~
  - ☑ ~~shared module~~
  - ☑ ~~slots (counter + counter)~~
  - ☑ ~~plotly~~
- ☑ ~~Write tests~~
- ☑ ~~Clean up code~~
- ☑ ~~Create docs for creating a custom component~~
  - ☑ ~~Recommend Lit as the "default" (but not only) path.~~

## v1 (Mesop Core)

**Goal:** Provide a long-term stable API (1+ year) that can support a wide range of use cases, with a well-lit path for writing and publishing components.

**Requirements:** *(note: these are subject to change, but as a rough starting point, I think we should tackle most/all of these requirements)*

- ☐ Enable packaging components for pip
- ☐ Custom component gallery
- ☐ Support npm / build integration
- ☐ Publish a Mesop JS package (TBD)
- ☐ Guides for integrating with the major frameworks (react, vue, angular)
- ☐ Loading non-JS assets (e.g. images, CSS)
- ☐ Lazy loading (don't load custom components until it's needed)
- ☐ Named slots

# Open Questions

## Should Mesop use native web components internally?

As mentioned above, Mesop itself is built on Angular and heavily depends on Angular Material components for many of its built-in native components.

It's theoretically possible for Mesop to shift from Angular to using web components for its built-in native components, however this is not advisable right now:

- **Component coverage** - Angular Material has many more components than Material Web components. It's unclear when or if Material Web would reach parity.
- **Major migration work** - we'd need to change a very non-trivial part of the Mesop web codebase. One of the benefits of using Angular is that we rely heavily on their toolchain (example) particularly to integrate with Bazel. Although the material-web repo is used in google3, it doesn't appear any of the Bazel integrations is available open-source.

## How should custom components be packaged?

Because the typical Mesop app developer is a Python developer, who is probably not familiar with the FE/Web ecosystem, we should encourage custom component developers to publish their components to PyPI, so you can `pip install custom-mesop-component`.

We will tackle custom component packaging in a separate proposal, but for now we expect custom components to be used in-house (e.g. only used by a single organization) where packaging isn't needed (and potentially prohibited by company policy).