Our motto is 'move slowly and break things'. No, wait, that came out wrong...

Almost a year after we first started talking about version 2 on the Svelte issue tracker, it's finally time to make some breaking changes. This blog post will explain what changed, why it changed, and what you need to do to bring your apps up to date.

tl;dr

Each of these items is described in more depth below. If you get stuck, ask for help in our friendly Discord chatroom.

- Install Svelte v2 from npm
- Upgrade your templates with svelte-upgrade
- Remove calls to `component.observe`, or add the `observe` method from svelte-extras
- Rewrite calls to `component.get('foo')` as `component.get().foo`
- Return `destroy` from your custom event handlers, rather than `teardown`
- Make sure you're not passing numeric string props to components

New template syntax

The most visible change: we've made some improvements to the template syntax.

A common piece of feedback we heard was 'ewww, Mustache' or 'ewww, Handlebars'. A lot of people who used string-based templating systems in a previous era of web development *really* dislike them. Because Svelte adopted the `{{curlies}}` from those languages, a lot of people assumed that we somehow shared the limitations of those tools, such as weird scoping rules or an inability to use arbitrary JavaScript expressions.

If you need to show an actual `{` character, it's as easy as `&#123;`

Beyond that, JSX proved that double curlies are unnecessary. So we've made our templates more... svelte, by adopting single curlies. The result feels much lighter to look at and is more pleasant to type:

```
<h1>Hello {name}!</h1>
```

There are a few other updates. But you don't need to make them manually — just run svelte-upgrade on your codebase:

```
npx svelte-upgrade v2 src
```

This assumes any `.html` files in `src` are Svelte components. You can specify whichever directory you like, or target a different directory — for example, you'd do `npx svelte-upgrade v2 routes` to update a Sapper app.

To see the full set of changes, consult the svelte-upgrade README.

Computed properties

Another thing that people often found confusing about Svelte is the way computed properties work. To recap, if you had a component with this...

```
export default {
    computed: {
        d: (a, b, c) => a = b + c
```

```
    }
};
```

...then Svelte would first look at the function arguments to see which values `d` depended on, and then it would write code that updated `d` whenever those values changed, by injecting them into the function. That's cool, because it allows you to derive complex values from your component's inputs without worrying about when they need to recomputed, but it's also... *weird*. JavaScript doesn't work that way!

In v2, we use destructuring instead:

```
export default {
    computed: {
        d: ({ a, b, c }) => a = b + c
    }
};
```

The Svelte compiler can still see which values `d` depends on, but it's no longer injecting values — it just passes the component state object into each computed property.

Again, you don't need to make this change manually — just run svelte-upgrade on your components, as shown above.

Sorry, IE11. It's not you, it's... well actually, yeah. It's you

Svelte v1 was careful to only emit ES5 code, so that you wouldn't be forced to faff around with transpilers in order to use it. But it's 2018 now, and almost all browsers support modern JavaScript. By ditching the ES5 constraint, we can generate leaner code.

If you need to support IE11 and friends, you will need to use a transpiler like Babel or Bublé.

New lifecycle hooks

In addition to `oncreate` and `ondestroy`, Svelte v2 adds two more lifecycle hooks for responding to state changes:

```
export default {
    onstate({ changed, current, previous }) {
        // this fires before oncreate, and
        // whenever state changes
    },

    onupdate({ changed, current, previous }) {
        // this fires after oncreate, and
        // whenever the DOM has been updated
        // following a state change
    }
};
```

You can also listen to those events programmatically:

```
component.on('state', ({ changed, current, previous }) => {
    // ...
});
```

component.observe

With the new lifecycle hooks, we no longer need the `component.observe(...)` method:

```
// before
export default {
    oncreate() {
        this.observe('foo', foo => {
            console.log(`foo is now ${foo}`);
        });
    }
};

// after
export default {
    onstate({ changed, current }) {
        if (changed.foo) {
            console.log(`foo is now ${current.foo}`);
        }
    }
};
```

This shrinks the amount of code Svelte needs to generate, and gives you more flexibility. For example, it's now very easy to take action when any one of *several* properties have changed, such as redrawing a canvas without debouncing several observers.

However, if you prefer to use `component.observe(...)`, then you can install it from svelte-extras:

```
import { observe } from 'svelte-extras';

export default {
    methods: {
        observe
    }
};
```

component.get

This method no longer takes an optional `key` argument — instead, it always returns the entire state object:

```
// before
const foo = this.get('foo');
const bar = this.get('bar');
```

```
// after
const { foo, bar } = this.get();
```

This change might seem annoying initially, but it's the right move: among other things, it's likely to play better with type systems as we explore that space more fully in future.

event_handler.destroy

If your app has custom event handlers, they must return an object with a `destroy` method, *not* a `teardown` method (this aligns event handlers with the component API).

No more type coercion

Previously, numeric values passed to components were treated as numbers:

```
<Counter start='1'/>
```

That causes unexpected behaviour, and has been changed: if you need to pass a literal number, do so as an expression:

```
<Counter start={1}/>
```

Compiler changes

In most cases you'll never need to deal with the compiler directly, so this shouldn't require any action on your part. It's worth noting anyway: the compiler API has changed. Instead of an object with a mish-mash of properties, the compiler now returns `js`, `css`, `ast` and `stats`:

```
const { js, css, ast, stats } = svelte.compile(source, options);
```

`js` and `css` are both `{ code, map }` objects, where `code` is a string and `map` is a sourcemap. The `ast` is an abstract syntax tree of your component, and the `stats` object contains metadata about the component, and information about the compilation.

Before, there was a `svelte.validate` method which checked your component was valid. That's been removed — if you want to check a component without actually compiling it, just pass the `generate: false` option.

My app is broken! Help!

Hopefully this covers everything, and the update should be easier for you than it was for us. But if you find bugs, or discover things that aren't mentioned here, swing by Discord chatroom or raise an issue on the tracker.