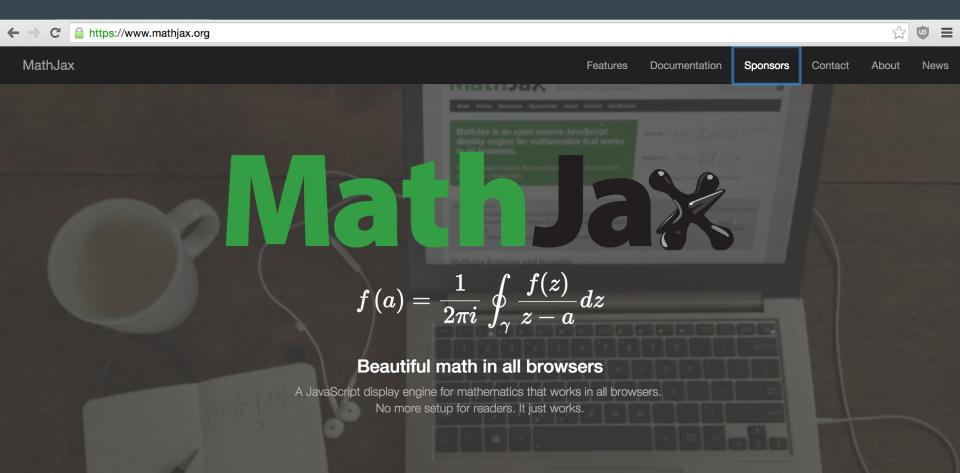
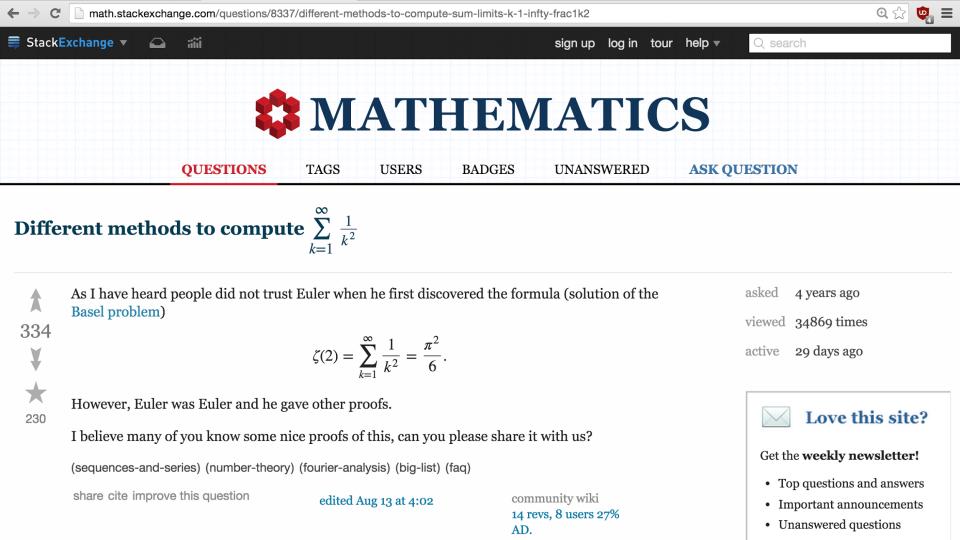
## MathJax and Custom Elements

Jan Marthedal Rasmussen http://janmr.com @janmarthedal





## Demo: Dynamic math with MathJax

demo/dynamic-math.html

## Dynamic math with MathJax

```
var elem = DynMath.createElement('a^2+b^2=c^2');
container.appendChild(elem);
DynMath.typeset(elem);
```

## **Custom Elements**



## But why?

#### **Semantics for humans**

https://elements.polymer-project.org/elements/google-map

#### **Semantics for machines?**

# But using custom tag names has been possible for years...

```
<head>
 <style>
   my-element { color: red; }
 </style>
</head>
<body>
 Some paragraph with an my-element>unknown</my-element>
    element
 <my-element></my-element>
 Another paragraph with a my-element>mysterious</my-element>
    element
</body>
```

## Demo: Unknown element

demo/unknown.html

## Registering a Custom Element

- document.createElement('my-element', ...)
- Tag name must
  - be a legal XML tag name
  - o contain a '-'
  - not contain upper-case letters

## Registering a Custom Element

- document.createElement('my-element', ...)
- Tag name must
  - be a legal XML tag name
  - o contain a '-'
  - not contain upper-case letters
  - not be any of annotation-xml, color-profile, font-face, font-face-src,
     font-face-uri, font-face-format, font-face-name, missing-glyph

## Lifecycle callbacks

Get notified when your custom element

- gets created
- is attached to the DOM
- is detached from the DOM
- has an attribute changed

#### <my-element>

```
(function () {
 var counter = 0,
     element prototype = Object.create #TMLElement.prototype);
 element prototype.createdCallback = function () {
   this.number = ++counter;
   console.log('my-element[%d] created', this.number);
 element prototype.attachedCallback = function () {
   this.textContent = '[my-element ' + this.number + ']';
   console.log('my-element[%d] attached', this.number);
```

#### <my-element>

```
element_prototype.detachedCallback = function () {
  console.log('my-element[%d] detached', this.number);
}
element_prototype.attributeChangedCallback =
  function (attr, oldVal, newVal) {
   console.log('my-element[%d] attribute %s change: %s -> %s',
        this.number, attr, oldVal, newVal);
  }
```

## <my-element>

```
document.registerElement('my-element', {
    prototype: element_prototype
  });
})();
```

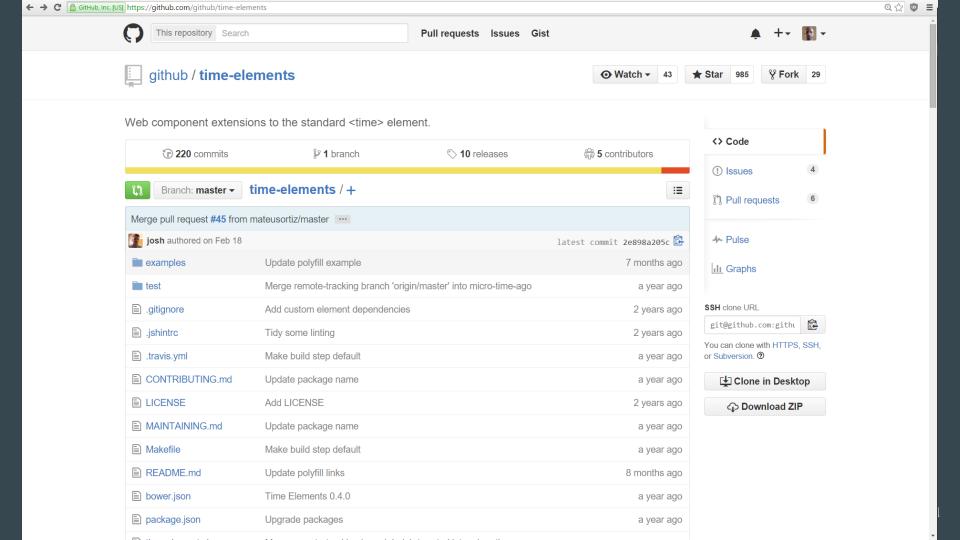
## **Demo: <my-element>**

demo/pre-load.html

## But what if <my-element> is defined *after* using it?

## **Demo: <my-element> with delay**

demo/post-load.html



## Type Extensions

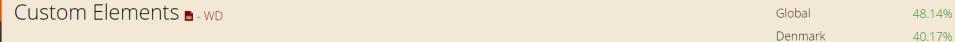
```
<time datetime="2015-02-18T15:20:33Z"

is="time-ago"

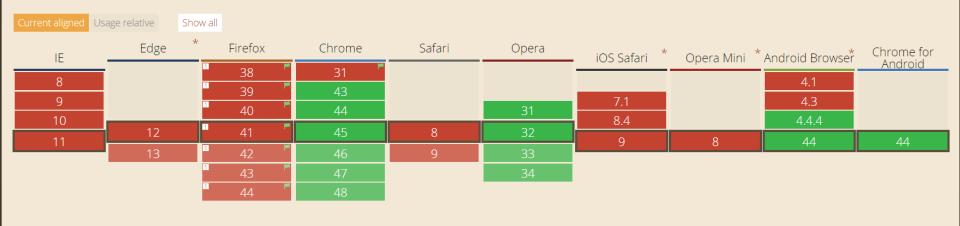
title="Feb 18, 2015, 4:20 PM GMT+1">7 months ago</time>
```

## Type Extensions

```
var TimeAgoPrototype = Object.create #TMLTimeElement.prototype);
window.TimeAgoElement = document.registerElement(time-ago', {
   prototype: TimeAgoPrototype,
   'extends': 'time'
});
```



Method of defining and using new types of DOM elements in a document.



Notes Known issues (0) Resources (10) Feedback

Current MS Edge status: Under Consideration

Enabled through the "dom.webcomponents.enabled" preference in about:config

## Polyfill: webcomponents.js

https://github.com/WebComponents/webcomponentsjs

#### **Browser Support**

Our polyfills are intended to work in the latest versions of evergreen browsers. See below for our complete browser support matrix:

| Polyfill           | IE10     | IE11+    | Chrome*  | Firefox* | Safari<br>7+* | Chrome<br>Android* | Mobile<br>Safari* |
|--------------------|----------|----------|----------|----------|---------------|--------------------|-------------------|
| Custom<br>Elements | ~        | <b>√</b> | ✓        | ✓        | <b>√</b>      | ✓                  | ✓                 |
| HTML Imports       | ~        | ✓        | ✓        | ✓        | <b>√</b>      | $\checkmark$       | ✓                 |
| Shadow DOM         | ✓        | ✓        | ✓        | ✓        | <b>√</b>      | $\checkmark$       | ✓                 |
| Templates          | <b>√</b> | ✓        | <b>√</b> | <b>√</b> | ✓             | ✓                  | ✓                 |

## <math-tex>

#### A <math-tex> custom element

```
Let <math-tex>x \in \mathbb R</math-tex> be a real number
<math-tex display="block">\sum_{k=1}^n k^2</math-tex>
```

## ... but first: <mathjax-loader>

```
(function () {
  var states = {start: 1, loading: 2, ready: 3, typesetting: 4},
      state = states.start,
      queue = [],
      src = 'https://cdn.mathjax.org/mathjax/latest/MathJax.js',
      element prototype = Object.create(HTMLElement.prototype);
```

```
function load mathjax(callback) {
  state = states.loading;
  window.MathJax = {
    skipStartupTypeset: true,
    jax: ['input/TeX', 'output/HTML-CSS'],
    TeX: { extensions: ['AMSmath.js', 'AMSsymbols.js'] },
    AuthorInit: function () {
     MathJax.Hub.Register.StartupHook('End', callback);
  };
  var script = document.createElement('script');
  script.type = 'text/javascript';
  script.src = src;
  script.async = true;
  document.head.appendChild(script);
```

```
element_prototype.attachedCallback = function () {
   if (this.hasAttribute('src'))
      src = this.getAttribute('src');
   load_mathjax(function () {
      state = states.ready;
      flush_queue()
   });
};
```

```
function flush queue() {
  var to process = queue.map(function (elem) {
    return [MathJax.Hub.isJax(elem), elem];
  }).filter(function (item) {
   return item[0] !== 0;
  });
  queue = [];
  if (to process.length) {
    state = states.typesetting;
    to process.forEach(function (item) {
      var action = item[0] < 0 ? 'Typeset' : 'Reprocess';</pre>
      MathJax.Hub.Queue([action, MathJax.Hub, item[1]]);
    });
    MathJax.Hub.Queue(flush queue);
  } else
    state = states.ready;
```

```
element_prototype.typeset = function (elem) {
   queue.push(elem);
   if (state === states.ready)
     flush_queue();
};
```

```
document.registerElement('mathjax-loader', {
    prototype: element_prototype
});
})();
```

## ... now back to <math-tex>

```
(function() {
 var mathjax,
     element prototype = Object.create(HTMLElement.prototype);
 function check mathjax() {
   if (mathjax) return;
   mathjax = document.querySelector('mathjax-loader') | |
              document.createElement('mathjax-loader');
   if (!mathjax | | typeof mathjax.typeset !== 'function')
     console.warn('no mathjax-loader');
   else if (!document.contains(mathjax))
     document.head.appendChild(mathjax);
```

```
element_prototype.createdCallback = function () {
  check_mathjax();
  this._jax = document.createElement('script');
  this._jax.type = 'math/tex';
};
```

```
element_prototype.attachedCallback = function () {
    this._jax.text = this.textContent;
    this.innerHTML = '';
    this.appendChild(this._jax);
    this.update();
};

element_prototype.detachedCallback = function () {
    this.textContent = this._jax.text;
};
```

```
element_prototype.attributeChangedCallback = function (attr) {
   if (attr === 'display')
      this.update();
};
```

```
Object.defineProperty(element_prototype, 'math', {
   get: function () {
     return this._jax.text;
   },
   set: function (value) {
     this._jax.text = value;
     this.update();
   }
});
```

```
document.registerElement('math-tex', {
    prototype: element_prototype
  });
})();
```

# Demo: Dynamic math again

demo/dynamic-math.html

# We can do better

- ShadowDOM
- MutationObserver
- Custom events

# Demo: A better < math-tex>

demo/better-math-tex.html

#### Some resources

- http://webcomponents.org
- https://customelements.io
- https://github.com/janmarthedal/math-tex

### Some resources

- http://webcomponents.org
- https://customelements.io
- https://github.com/janmarthedal/math-tex

# Thank you!