# Web Components: Web's Polymer Future

Abhishek Tiwari

Citation: *A. Tiwari*, ”Web Components: Web's Polymer Future”, Abhishek Tiwari, 2014. doi:10.59350/7y4jy-n5e33

Published on: June 11, 2014

Recently I have been playing with Web Components (WC). W3C specification drafts describe Web Components as the component model for the web [1]. This component model consists of five new constructs: *HTML Templates*, *Decorators*, *Custom Elements*, *Shadow DOM* and *HTML Imports*. These constructs will be used as new primitives in the browser. This includes new standard and custom reusable HTML elements. So for instance a `google-analytics` Web Component element,

```
<google-analytics domain="example.com" code="UA-XXXXX-Y"></google-
    analytics>
```

is equivalent to following but a lot more cleaner, composable and standardised.

```
<!-- Google Analytics -->
<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function
    (){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement
    (o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore
    (a,m)
})(window,document,'script','//www.google-analytics.com/analytics.js','ga'
    );

ga('create', 'UA-XXXXXX-Y', 'example.com');
ga('send', 'pageview');

</script>
<!-- End Google Analytics -->
```

Coming back to these new constructs, each of these can be used individually as well as in combination. By combining these constructs one can create reusable and composable components. These components will allow developers to create very large applications in a declarative way without dealing with complexity and monolithicity of current approaches.

## Why

So what is wrong with current constructs and why we need these Web Components?

### Fragmented and Blotted

Although there has been a lot of innovation on both CSS and JavaScript side , current ecosystem of constructs like frameworks, plugins, widgets, templates, etc. is highly fragmented and either not reusable or lack standardisation. This has resulted into complex, non-modular and monolithic web

[1] W3C Introduction to Web Components

applications. In some cases blotted with plugins and libraries doing very tiny bits on a web page but loaded on each and every web page. More often, developers are trying to solve the problems using frameworks and plugins which should be ideally tackled by web community (W3C) with help of browser vendors.

### Encapsulation

Moreover, current ecosystem lacks proper encapsulation which means it is always possible styles and script might break in unpredictable way. As an example, when a page includes an external widget, DOM tree inside widget is visible to page's script and style. Hence when page has id or class name overlap with widget then page style or script can be accidentally applied to widget DOM tree as well. Currently iFrames offer most basic way to encapsulate external HTML but they have serious issues such as security threats, SEO leaks and poor web tracking.

## New primitives

### HTML Templates

- Act as scaffold or blueprint
- Inert chunks of markup which can be activated or intended to use later
- Parsed by parser by not rendered (script not executed, stylesheets/images not loaded, media not played)
- Hidden from document, after activation a template clone is populated and appended to document
- `template` element has a property `content` which holds the content of the template in a document fragment
- Get the template element → Access the template content document fragment and clone it → Populate content to clone → Append populated clone to document

```
<template id="commentTemplate">
    <div>
        <img src=""> <!-- Populated on run time -->
        <div class="comment-text"></div> <!-- Populated on run time -->
    </div>
</template>
<script>
function addComment(imageUrl, text) {
  // Get the template element
  var t = document.querySelector("#commentTemplate");
  // Access the template content document fragment and clone it
  var comment = t.content.cloneNode(true);
  // Populate content to clone
  comment.querySelector('img').src = imageUrl;
  comment.querySelector('.comment-text').textContent = text;
  // Append populated clone to document
  document.body.appendChild(comment);
}
</script>
```

### Decorators

- Enhances or overrides the presentation of an existing element
- Controlled by CSS
- Still on drawing board no specification yet

### Custom Elements

- Create new HTML elements - expand HTML's existing vocabulary
- Extend existing DOM objects with new imperative APIs

We can create custom HTML element for concepts like cars, movies, books etc. Following example declares custom HTML element `car` and `gallery` and composes them (including `a` element) to produce a markup which conceals the complexity of displaying details about cars.

```
<car make="Toyota" model="Camry" year="2013">
    <gallery type ="exterior" ></gallery>
    <gallery type ="interior" ></gallery>
    <a href="/more">Learn more</a>
</car>
```

### Shadow DOM

- Act as mortar or glue
- DOM & Style encapsulation boundaries for more reliable composition of user interface elements
- Similar mechanics browsers vendors have been using to implement their internal UI[2]

In following example, there are two `.outer` styles. Due to encapsulation boundary, style in main page can not access the Shadow DOM elements and hence can not style them. Only `.outer` declared as part of Shadow DOM will apply. Result will display "Hi! My name is Not Bob" in red colour.

[2] Web Components Shift

```
<div id="nameTag">Bob</div> <!-- Populated on run time -->
<template id="nameTagTemplate"> <!-- This is template -->
<style>
.outer {
  color: red; /* This style is part of Shadow DOM Tree, Applied on Shadow
      DOM */
}
</style>
<div class="outer">
  <div class="boilerplate">
    Hi! My name is
  </div>
  <div class="name">
    Not Bob
  </div>
</div>
</template>

<script>
// Create Shadow Root
var shadow = document.querySelector('#nameTag').createShadowRoot();
// Get Template Content
var template = document.querySelector('#nameTagTemplate');
// Clone Template Content and Populate the node
shadow.appendChild(template.content.cloneNode());
</script>

<style>
.outer {
  color: black; /* This style is part of Main Page or DOM Tree, Not
      applied on Shadow DOM */
}
</style>
```

## HTML Imports

- Load element definitions and other resources declaratively
- Defines how templates, decorators and custom elements are packaged and loaded as a resource
- Allows sharing and reuse of Web Components

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="import" href="x-foo.html"> <!-- Import x-foo.html -->
  </head>
  <body>
    <x-foo></x-foo>  <!-- Element definition is in x-foo.html -->
  </body>
```

```
</html>
```

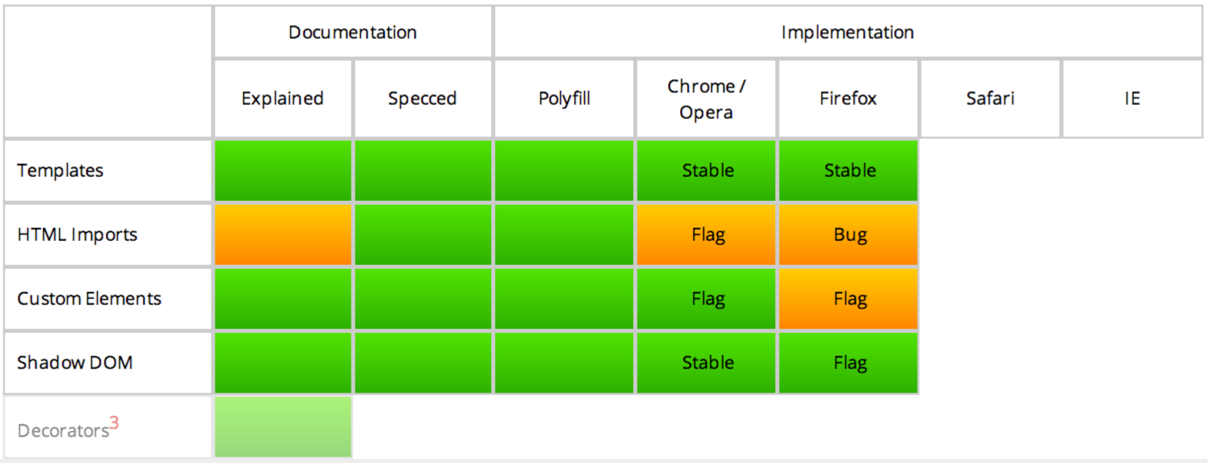
```
<!-- Content of x-foo.html -->
<element name="x-foo">
  <h1>I am X FOO</h1>
</element>
```

**Figure 1:** Polymer web is future but using Polyfills like X-Tags and Polymer we can start implementing some of the Web Components now

## Current Status

Following describes the current status of Web Components [3]. Except the *Decorators*, all other Web Component types have a working specification [4][5][6][7].

[3] Are We Componentized Yet?
[4] Shadow DOM Spec
[5] HTML Imports Spec
[6] HTML Templates Spec
[7] Custom Elements Spec



| | Documentation | | Implementation | | | | |
|---|---|---|---|---|---|---|---|
| | Explained | Specced | Polyfill | Chrome / Opera | Firefox | Safari | IE |
| Templates | 🟩 | 🟩 | 🟩 | 🟩 Stable | 🟩 Stable | | |
| HTML Imports | 🟧 | 🟩 | 🟩 | 🟧 Flag | 🟧 Bug | | |
| Custom Elements | 🟩 | 🟩 | 🟩 | 🟩 Flag | 🟧 Flag | | |
| Shadow DOM | 🟩 | 🟩 | 🟩 | 🟩 Stable | 🟩 Flag | | |
| Decorators[3] | 🟩 | | | | | | |

**Figure 2:** Current Status of Web Components Specification, Polyfills and Browser Implementation

In terms of implementation,

- two polyfills of web components X-Tags[8] and Polymer[9] can support *HTML Templates*, *HTML Imports*, *Custom Elements* and *Shadow DOM*.
- browser are progressing well with working specification. Chrome, Opera and Firefox are already shipping stable implementations. For instance, HTML Templates are now supported by latest version of Chrome, Opera and Firefox[10].

## Closing thoughts

Web Components are already here and they are gone change the way we build apps. With new constructs and modern tooling around them we can rapidly build complex apps with ease. We will cover each of these constructs in detail in upcoming posts.

[8] X-Tags
[9] Polymer - Building blocks for the web
[10] Template Compatibility