Speakur: leveraging Web Components for composable applications

Speakur: leveraging Web Components for composable applications

by

Preston Brent Landers, B.A.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2015

Dedicated to my wife Andrea and to my parents.

Acknowledgments

I wish to thank the multitudes of people who helped me. Time would fail me to tell of the multitudes of individuals …

Speakur: leveraging Web Components for composable applications

Preston Brent Landers, M.S.E.
The University of Texas at Austin, 2015

Supervisor: Adnan Aziz

This report is a case study of applying encapsulation, composition and distributed synchronization techniques to web application architecture with the use of Web Components, a proposed extension to the W3C HTML5 document standard. The author presents Speakur, a real-time social discussion plugin for the mobile and desktop web, as an example of applying HTML5 Web Component technologies to realize software engineering principles such as modularity, encapsulation and separation of concerns when composing applications from components sourced from diverse authors.

Web authors can add a Speakur discussion to their page by inserting a simple HTML element at the desired spot to give the page a real-time discussion or feedback system. Speakur uses the Polymer framework's implementation of the draft Web Components (WC) standard to achieve encapsulation of its internal implementation details from the containing page and present a simplified, well defined interface (API).

Web Components are a proposed World Wide Web Consortium (W3C) standard for writing custom HTML tags that take advantage of new browser technologies like Shadow DOM, package importing, CSS Flexboxes and data-bound templates. This report reviews Web Component technologies and provides a case study for structuring a real-world WC applet that is embedded in a larger app or system. The major research question is whether W3C Web Components provide a viable path towards the encapsulation and composition principles that have largely eluded web engineers thus far. In other words, *are components really the future of the web*? Subsidiary topics include assessing the maturity and suitability of current Web Components technologies for widespread deployment and how to efficiently synchronize component state across a distributed network.

Table of Contents

List of Tables

List of Figures

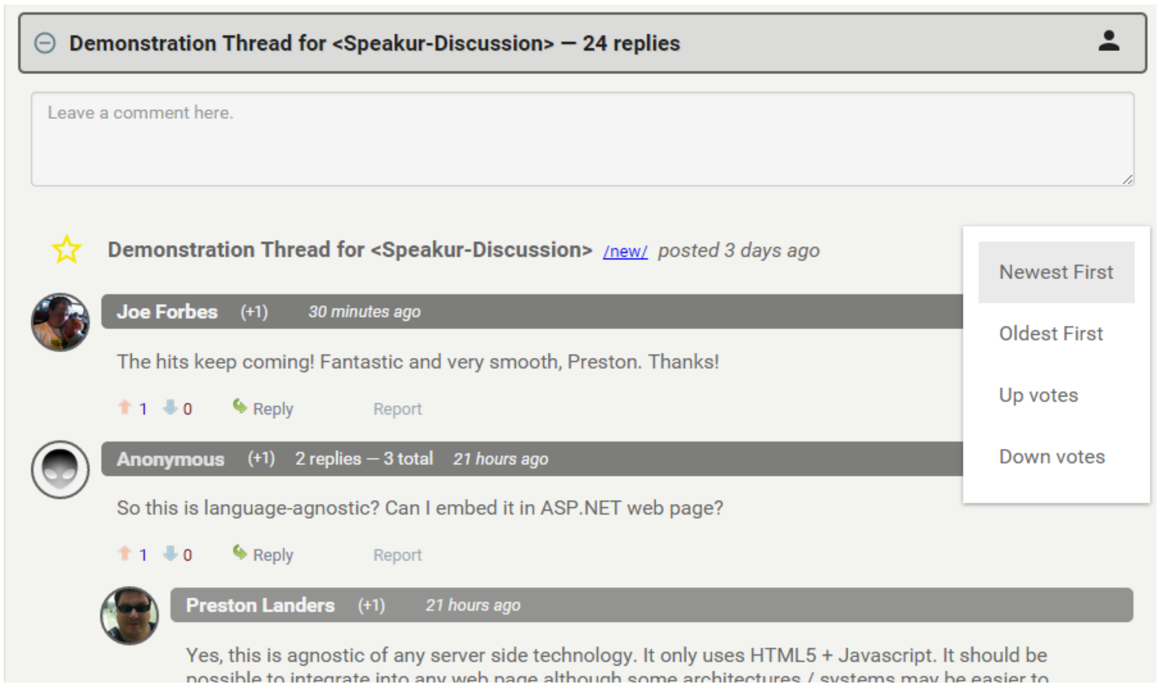
Chapter 1

Introduction

This report presents a case study of applying W3C Web Components to achieve encapsulation and modularity within the context of collaborative web authoring. The author has created Speakur, a real-time discussion social plugin for the web, as an experiment to determine the maturity and viability of using Web Components to create modern, highly composable web applications.

Like most web applications, Speakur is written in HTML markup and Javascript code. HTML is a declarative markup language used to create documents — web pages — which are viewed with the help of a program called the browser. The Hypertext Markup Language (HTML) standard has proven wildly successful since its introduction in 1993 by British computer scientist Tim Berners-Lee, with billions and billions of pages served, and millions of public and private web sites forming a major part of our information landscape. [CITE?] More than any other invention, other than perhaps email, the World Wide Web (WWW) has shaped how we see and use the global network.

Those designing and programming applications for the Web as a computing platform have long dreamed of the ability to mix and match independent, reusable chunks of functionality — components — in their documents without mutual cou-



Demonstration Thread for <Speakur-Discussion> — 24 replies

Leave a comment here.

Demonstration Thread for <Speakur-Discussion> /new/ posted 3 days ago

Joe Forbes (+1) 30 minutes ago

The hits keep coming! Fantastic and very smooth, Preston. Thanks!

1 0 Reply Report

Anonymous (+1) 2 replies — 3 total 21 hours ago

So this is language-agnostic? Can I embed it in ASP.NET web page?

1 0 Reply Report

Preston Landers (+1) 21 hours ago

Yes, this is agnostic of any server side technology. It only uses HTML5 + Javascript. It should be

Figure 1.1: A Speakur thread inside a demonstration page.

pling and interference. The original and current Document Object Model (DOM) browser abstraction provided by HTML does not allow for significant decoupling; everything lives together on one big page. Hacks like the `<iframe>` tag let you work around some of these restrictions, usually in a limited and inelegant way.

At the time of HTML's introduction the concept of quickly and easily composing a static web page, much less a full-fledged dynamic application, out of Lego-like reusable building blocks seemed like a distant dream at best. The introduction of the Javascript[1] (JS) programming language to web browsers in 1995 allowed

[1]Javascript, also rendered as JavaScript or JS, has no significant relation to Sun's (now Oracle's) popular Java programming language; the name is an unfortunate coincidence at best.
To complicate the naming situation even further, Javascript is officially standardized under the

for a completely new dimension of dynamic behavior that was not possible before. Eventually web apps powered by Javascript like Gmail and Google Docs rivaled traditional desktop applications in functionality and usability while being instantly accessible from nearly anywhere. Still, web apps had to be stitched together 'by hand' in ways that carefully ensured that the different parts didn't step on each other's toes, else disaster frequently ensued. Each component or area of the system could not help but be coupled to the others at some level as a result of the programming model imposed by the DOM and HTML.

Over the years the dynamic behaviors afforded by Javascript grew in importance along with the web, and helped contribute to its success. The flexible, loosely typed nature of Javascript aided the prototyping process and initial development, but the difficulties of maintaining a semblance of coherence in a large sprawling application soon became apparent. Over time a bewildering array of frameworks and libraries sprang up around the HTML/JS ecosystem to help manage this complexity and to provide scaffolding and structure for client side web apps. For many years individual JS frameworks seemed to come and go as ephemerally as teenage pop idols. The industry kept searching for the Next Big Thing that would make writing high quality web apps less of bug-ridden, messy chore.

In recent years (roughly 2011 to 2014) Google's Angular [CITE] has emerged as a dominant client framework, due in part to its high perceived quality [CITE] and the fact that it represents an common point for a fragmented industry to rally

name ECMAScript (ES).

around. Facebook's React JS library with its Virtual DOM is an up-and-comer focused on high performance that is more complementary in nature to Angular than a true challenger.

Yet despite the emergence of the updated HTML5 standard in 2011 and the recent successes of web frameworks like Angular and React in capturing developer attention, a clear picture still did not exist of how web apps could achieve the encapsulated component model that had become prevalent in other areas of software engineering. That is, until engineers from Google[2] and Mozilla[3] and other organizations got together [**WHEN?**] to draft a new standard called **Web Components** that will extend and enhance HTML5 in ways that could have a significant long-term impact. For example, as of the time of this writing, a rewrite of Angular called Angular 2.0 is slated to include Web Components as a core architectural element [CITE].

1.1 Web Components Overview

Fundamentally, the Web Components standard consists of four new core DOM technologies — extensions to the current HTML5 standard. If these standards are accepted by major browser vendors and the World Wide Web Consortium (W3C) which maintains HTML, they will eventually become native browser features and available directly to any web page without needing to use any addi-

[2]As of March 2015, Google's Chrome browser is the most popular desktop browser. [CITE]

[3]The Mozilla Foundation is the sponsor of the popular Firefox web browser. It grew out of Netscape, whose Navigator browser helped bring the web to a mass audience.

tional JS frameworks or libraries. The core Web Component technologies are:

- **Custom Elements**: extending HTML with author-created tags
- **Shadow DOM**: encapsulation for the internals of custom elements
- **Templates**: scaffolding for instantiating blocks of HTML from inert templates
- **Imports**: packaging for HTML components

This report also explores several related web standards initiatives that are frequently associated with Web Components but are not formally grouped under them, including mutation observers, model driven views, and the CSS Flexible Boxes and CSS Grid systems. In part because these technologies are not yet formally accepted as W3C standards and are not yet widely implemented in typical mobile and desktop browsers, Speakur has been implemented using Google's experimental Polymer framework [CITE]. Polymer provides a Javascript 'polyfill' library to implement many of the new Web Component features in browsers which would otherwise not support them. Eventually this platform polyfill should become unnecessary, in theory, as WC becomes widely adopted in browsers. Some browsers like Google Chrome already have at least some native Web Component support and on these browsers the polyfill is effectively a 'no-op'.

The potential componentization of the web is one of the most exciting developments in web engineering in years and follows the overall growth in software-as-a-service (SaaS) and the service oriented architecture model. The conversion of dynamic web logic—not mere snippets of plain HTML—into bundles of reusable, extendable, composable components enables web developers to move to a higher level of abstraction than was previously possible.

The move towards a component-based Web will enable interesting new composite services, mashups, and may help broaden the potential pool of web developers. What previously required a highly integrated, high-overhead development model or lots of tedious glue code can become as simple as importing a custom element and dropping it onto a page.

1.2 Structure of This Report

The goal of this report is to demonstrate the application of software engineering design patterns embodied in the W3C proposed Web Components standard such as encapsulation, modular composition, and the synchronization of distributed application state. This report discusses many of the goals and principles of the Web Components initiative and how a number of different technologies taken together help raise the overall level of abstraction for content authors, web engineers, and application developers — which I will refer to collectively as (web) authors for short.

The Background section provides an introduction to some of the architectural problems inherent in modern web authoring and how Web Components (WC) address them. It also provides some background on software engineering design patterns that are embodied in Web Components such as encapsulation and composition. It describes some of the motivations behind the development of

Speakur and some of the specific software engineering questions it addresses, such as the ability to provide a hassle-free way to host an embedded discussion forum inside an arbitrary web resource in a way that is fully encapsulated.

The Approach section details the specific structures and techniques used when constructing a Web Component, and describes the high level software architecture choices that went into Speakur. It describes how Speakur uses Web Components to implement encapsulated modules whose internals are protected from unintentional outside influence. It also describes how the choice of the Firebase cloud database service and its Websocket-based event notification system impacts Speakur's architecture.

The Implementation section shows how to apply Web Component principles to the task of creating a flexible and generic discussion forum for both desktop and mobile browsers. It describes the low level architecture, code flow, and synchronization process. An important topic in this section is security: how can we implement a largely client-based system while maintaining some kind of data integrity?

This is followed by an Analysis section which discusses some of the outcomes as compared to the original goals and also looks at the impact of the selection of Web Components, Polymer, Firebase and some of the other architectural choices. A few quantitative results are included, I hope **(TODO)**.

Finally, the Conclusion section is there and wraps up the report with various words **(TODO)**.

1.3 Source Code and Demonstration Resources

The source code for Speakur consists of HTML and Javascript files located in a Git version control repository. These files constitute an "HTML Import" package that provides a **`<speakur-discussion>`** custom HTML element for the use of web authors in their own pages.

The Speakur source code and component documentation can be found on the social coding site GitHub.com:

`https://github.com/Preston-Landers/speakur-discussion`

Demonstrations of several web pages which show off embedded Speakur discussions are available at the following location:

`https://preston-landers.github.io/speakur-discussion/components/speakur-discussion/demo.html`

Chapter 2

Background

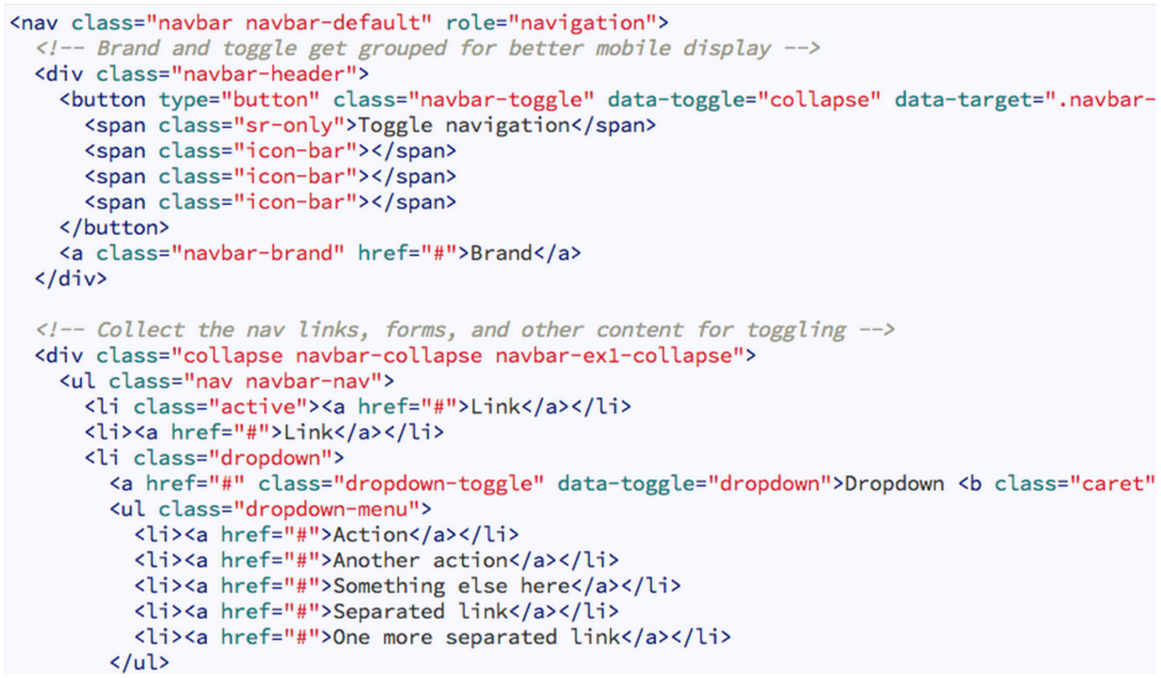
When the Web was first created by Tim Berners-Lee in 1989, web pages were largely envisioned as static *documents* with a single author or a small group of coordinating authors. The idea of composing a complex web application out of simple components like snapping together Lego blocks seemed like a distant dream at best. Until recently, web authors were limited to using the predefined HTML elements or 'tags' that were listed in the W3C standard and understood by browser programs, such as `<title>` and `<video>`. Creating your own *sui generis* HTML elements with unique behaviors seemed beyond the capabilities of the web browsers of the day like Mosaic and Netscape Navigator.

As of early 2015, modern web apps are typically written with a Javascript framework that provides a cohesive set of structures, design patterns and practices designed to facilitate composing web applications, large or small, from a number of sub-components. Angular, Meteor, and Backbone are three such frameworks. The difference between a 'framework' and a library is somewhat arbitrary, but typically frameworks are more comprehensive than narrowly focused utility libraries. Yet all frameworks must exist within the confines of the programming model provided by the browser and the Document Object Model (DOM). In this model, the entire web page or app belongs to a single ‘document’, constituent parts are not encapsulated or isolated from each other, and authors are limited to working with the predefined HTML tags. These issues make it difficult to create and share generic, reusable *web components* — in the abstract sense — among different users who may not use the same frameworks or follow the same set of assumptions and conventions.

2.1 Current challenges in web authoring

To illustrate how these problems affect the ability of authors to share and reuse code, let’s look at an example from the popular Twitter Bootstrap library [CITE]. Twitter Bootstrap is a collection of Cascading Style Sheet (CSS) rules and Javascript widgets or components designed to allow web authors to quickly ‘bootstrap’ an attractive, consistent look-and-feel onto a web page. Bootstrap provides pre-styled User Interface (UI) widgets such as menus, buttons, panels, dropdown selectors, alerts, dialogs, and so on, to be used as building blocks to construct web sites or applications. Because Bootstrap must work within the confines of the DOM and the HTML5 standard, this necessarily exposes a great deal of Bootstrap’s internals to its users. For example, to add a Bootstrap site navigation bar to your page, you must essentially copy and paste a large block of HTML and then customize it to your needs as shown in figure 2.1.

This forces Bootstrap’s users to tightly couple the layout of their page with the internal structure required by Bootstrap’s navigation bar widget. This coupling militates against Bootstrap significantly refactoring the internal structure of



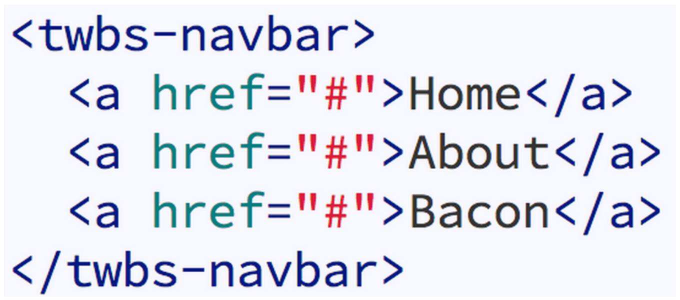
```
<nav class="navbar navbar-default" role="navigation">
  <!-- Brand and toggle get grouped for better mobile display -->
  <div class="navbar-header">
    <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
      <span class="sr-only">Toggle navigation</span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
      <span class="icon-bar"></span>
    </button>
    <a class="navbar-brand" href="#">Brand</a>
  </div>

  <!-- Collect the nav links, forms, and other content for toggling -->
  <div class="collapse navbar-collapse navbar-ex1-collapse">
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">Link</a></li>
      <li><a href="#">Link</a></li>
      <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown">Dropdown <b class="caret"
        <ul class="dropdown-menu">
          <li><a href="#">Action</a></li>
          <li><a href="#">Another action</a></li>
          <li><a href="#">Something else here</a></li>
          <li><a href="#">Separated link</a></li>
          <li><a href="#">One more separated link</a></li>
        </ul>
```

Figure 2.1: A partial example of Twitter Bootstrap navigation bar HTML.

the navigation widget because that would require a large community of developers to update their applications accordingly. In addition, because CSS rules normally apply across the entire page, the authors of Bootstrap must carefully select the scope and nomenclature of all rules to ensure minimal interference with other components and unintended effects. Even then, conflicts are inevitable when the entire page is treated as a single sandbox and you combine components from many different vendors.

What if instead one could create and share a reusable chunk of functionality — a web component – that hid all of these tedious structural details and encapsulated its private, internal state? What if web authors could create their *own*

HTML elements? Using Bootstrap's navigation bar could be as simple as replacing the code in figure 2.1 with a custom element like the one in figure 2.2.



```
<twbs-navbar>
  <a href="#">Home</a>
  <a href="#">About</a>
  <a href="#">Bacon</a>
</twbs-navbar>
```

Figure 2.2: Hypothetical Bootstrap nav bar custom element.

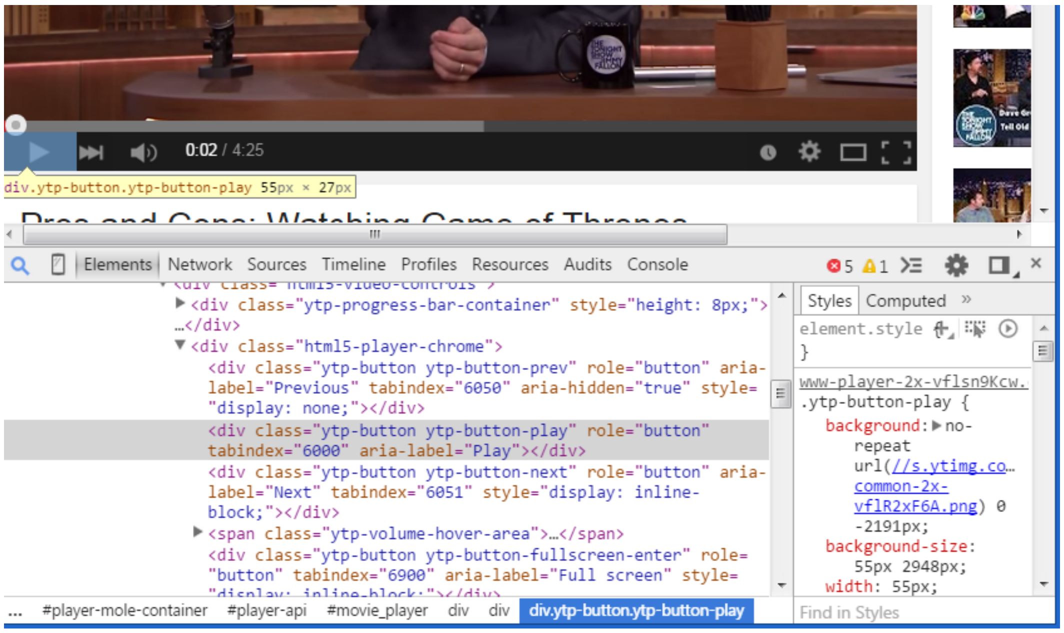
2.1.1 Encapsulation and composition

The Web Components working group, consisting of software engineers from several major browser vendors, looked at this situation and found that, in practice, browsers already had a suitable model for encapsulating components that hide complexity behind well-defined interfaces. That model was that one used internally by browsers to implement the newer HTML5 tags like the **`<video>`** element. The `<video>` element presents a simple interface (API) to HTML authors that hides the complexities of playing high definition video. Internally, however, browsers implement `<video>` with a 'shadow' or hidden document inside the object that contains the internal state. For example, an author can write:

```
<video loop src=...> </video>
```

to cause the video to loop repeatedly.

This shadow Document Object Model (DOM) inside the `<video>` tag creates the user interface (UI) needed to control video playback such as the volume controls, the timeline bar, and the pause and play buttons. These inner playback controls are themselves built out of HTML, CSS and JS but these details are not exposed to web authors who simply place a `<video>` element on their page. Figure 2.3 illustrates how this works. It shows the shadow (internal) DOM of a `<video>` element on a page with the Play button `<div>` highlighted.


0:02 / 4:25
div.ytp-button.ytp-button-play 55px × 27px
Elements
Network
Sources
Timeline
Profiles
Resources
Audits
Console
5
1
Styles
Computed
Find in Styles
... #player-mole-container #player-api #movie_player div div div.ytp-button.ytp-button-play

```
▶<div class="ytp-progress-bar-container" style="height: 8px;">
…</div>
▼<div class="html5-player-chrome">
  <div class="ytp-button ytp-button-prev" role="button" aria-
  label="Previous" tabindex="6050" aria-hidden="true" style=
  "display: none;"></div>
  <div class="ytp-button ytp-button-play" role="button"
  tabindex="6000" aria-label="Play"></div>
  <div class="ytp-button ytp-button-next" role="button" aria-
  label="Next" tabindex="6051" style="display: inline-
  block;"></div>
 ▶<span class="ytp-volume-hover-area">…</span>
  <div class="ytp-button ytp-button-fullscreen-enter" role=
  "button" tabindex="6900" aria-label="Full screen" style=
```

```
element.style {
}
www-player-2x-vflsn9Kcw.
.ytp-button-play {
  background:▶no-
    repeat
    url(//s.ytimg.co…
    common-2x-
    vflR2xF6A.png) 0
    -2191px;
  background-size:
    55px 2948px;
  width: 55px;
```

Figure 2.3: Opera's shadow DOM for `<video>` highlighting the Play button

This example illustrates two design principles that are widely followed in other areas of software engineering [CITE]:

- Use **encapsulation** and well defined interfaces, as the `<video>` element does, to protect private state, hide implementation complexity, and leave implementors free to refactor internals.

- Prefer **composition** or *has-a* relationships over inheritance or *is-a* relationships when building modules.

Composition helps reduce coupling or structural between modules by forcing them to interact using only public interfaces. In the case of the interface for `<video>`, it's composed of simple block elements and scoped CSS roles and the Volume and Play controls aren't particularly special objects, just `<divs>` with CSS rules and click handlers.

The solution, therefore, to these coupling problems in web authoring is to expose these internal brower APIs for creating elements in a safe and portable fashion. This will allow web authors to create their own rich custom elements using standard portable APIs, encapsulate their internals, and enable easier sharing, composition and integration. The question remains, which specific browser internals must be exposed and standardized in order to support Web Components?

2.2 Web Components

The Web Component initiative consists of two main technologies and two supporting features. Custom HTML Elements and Shadow DOM are the two key players while HTML Imports and Templates support these features. One of the central goals of the Web Components initiatives is to maintain interoperability across different browsers and frameworks, so that modules which adhere to the

Web Components standard can provide a consistent experience no matter what framework the developer chooses or which browser the user selects.

2.2.1 Custom HTML elements

Never before have web authors been able to define their own custom HTML elements that were not found in the official list. Actually, many authors and web frameworks have been doing exactly that for years, primarily for internal purposes where the custom elements are preprocessed and compiled down to standard HTML. The custom elements would not get sent to the end user's browser because it would not know what to do with them. Technically, the DOM has long supported creating custom-named elements, but it was not possible to do much with them because they were treated like an ordinary `<span>`. However the possibility now exists to create custom elements in a standard way that will work consistently across browsers.

TODO: Custom elements:

`http://www.w3.org/TR/custom-elements/`

The primary restriction is that all custom elements must have a `-` character (dash) in their name, such as `<my-element>`. This is to avoid a name collision with future built-in HTML elements. To create a new Custom Element, you must first register the element:

```
var MyElement = document.registerElement('my-element');
```

Then you place your new element on the page, either declaratively in HTML:

```
<my-element> hello, world! </my-element>
```

or imperatively with Javascript:

```
var MyElement = document.registerElement('my-element');

// instantiate a new instance of the element
var thisOne = new MyElement();
document.body.appendChild(thisOne); // add to the <body>
```

With a simple example like this the result does not look all that different from a `<span>`. To do something more interesting with your custom element you will need to the other features of Web Components: Shadow DOM, templates and imports.

2.2.2 Shadow DOM

Shadow DOM encapsulates the internal structure of an element. As we have seen, browsers use Shadow DOM to encapsulate the private state of standard elements like `<video>` but now this capability is extended to custom-defined elements.

TODO:

`http://www.w3.org/TR/shadow-dom/`

You can think of Shadow DOM like an HTML fragment inside an element that describes its external appearance without exposing these structural details[1]. Typically a custom element definition has a template (more on these in a moment) which produces the shadow DOM necessary to render the element. The actual

[1] Shadow DOM should not be confused with the React framework's Virtual DOM concept, which is closer in nature to HTML5 Templates than Shadow DOM.

contents of the shadow DOM are just ordinary elements.

Custom elements can wrap regular text, normal HTML elements, or other custom elements and then project that content into its own internal structure. In the example in figure 2.2 above, a simple `<twbs-navbar>` element consumes a set of three `<a>` (anchor or link) elements but internally transforms that to something like the example in figure 2.1, projecting the set of links into the nav menu structure with appropriate wrappers.

The `<content>` tag is used inside a custom element's template to indicate the spot where the consumed (wrapped) content should be *projected*. This wrapped content is known as *light DOM*, because it's given by the user and projected through into the shadow. Together the shadow DOM and light DOM form the *logical DOM* of a custom element. It is also possible for elements to have multiple shadow DOM sub-trees. This is used particularly for emulating object-oriented-like inheritance relationships between custom elements.

In languages like C# and Java, the encapsulation of classes and the protection of private object fields are a relatively strong guarantee by the language. But in the case of Web Components, Shadow DOM is not completely and utterly isolated from the containing page. It is possible to "reach inside" and break encapsulation to at least some degree, but the point is that this must be an intentional act by the developer and not an unexpected side-effect.

2.2.3 HTML Imports

One significant problem faced by web developers is the lack of any built-in packaging system for modules in HTML. Prior to Web Components there was no way to import a snippet of HTML or Javascript from an external location and insert it exactly one time into the current document, similar to an `#include` directive in the C language or the packaging and import systems popular in scripting languages like Python, Go and Ruby. Javascript could always be loaded with a `<script>` tag like usual, but this did not ensure that resources were loaded and executed exactly once, a process known as *de-duping*. A component that uses a certain JS resource might be found in two different spots on the page but that resource would be requested from the server twice, degrading application performance.

In order to fix these problems the HTML Imports standard allows for bringing in snippets of HTML, CSS or Javascript into the current document in a way that ensures automatic de-duping of repeated requests. The one major caveat is that de-duping only happens if the resources are named in exactly the same fashion in each case. Dealing with HTML Imports in a consistent fashion will be discussed in the Implementation section.

TODO: HTML imports:

`http://www.w3.org/TR/html-imports/`

`http://www.html5rocks.com/en/tutorials/webcomponents/imports/`

2.2.4 Templates

The last major piece of the Web Component puzzle is the native HTML5 `<template>` tag. Unlike the rest of Web Components, `<template>` has already become a standard part of the HTML5 specification, although one that is not yet widely used. *Template* is a frequently overloaded word with different meanings in different programming environments. While HTML5 templates have some similarities to the concept of templates popularized by frameworks like Angular and Django, there are some important differences.

TODO: Template elements:

`http://www.w3.org/TR/html5/scripting-1.html#the-template-element`

HTML5 templates are inert hunks of HTML embedded in the page that can be instantiated into 'real' elements by Javascript. Their basic function is to give a template for custom element representation.

However, templates are most useful in combination with 'live' data, not static, unchanging text. Binding data into templates with special operators[2] is **not** a part of the standard HTML5 template spec. The following example of a data bound template is something that does not work with plain Web Components alone:

```
<template>
  The temperature is {{ temp }} in {{ city }} right now.
</template>
```

This functionality is handled by a Javascript framework such as React or

[2]Sometimes called 'mustaches' or curly braces.

Polymer. Data-bound templates are discussed in more detail in the following chapter.

The primary benefit of HTML Templates from a performance perspective is that external resources referenced from the template (images, stylesheets, etc) will not be fetched until the template is actually instantiated. Templates are often used to declare the internal structure (shadow DOM) of custom elements. Therefore the resources needed to use the custom element aren't downloaded until they are actually needed, which is necessary when composing a large application out of numerous distinct elements.

2.2.5 Related technologies

There are a number of related W3C initiatives for web standards. Sometimes these are loosely grouped under the label Web Components, and they do help support componentization, but they are separate part of the HTML5 standard.

Mutation observers:

`http://www.w3.org/TR/dom/#mutation-observers`

Model driven views:

`http://mdv.googlecode.com/svn/trunk/docs/design_intro.html`

Pointer events:

`http://www.w3.org/TR/pointerevents/`

Web animations:

http://www.w3.org/TR/web-animations/

Selectors (similar to jQuery selectors)

http://www.w3.org/TR/selectors-api/

```
// find an element based on 'id' attribute.
var someElem = document.querySelector("#some-id");
```

Placeholder Notes: A significant problem with 'web components' story - scoping! There is just one global scope on the page. This leads to the practice of 'prefixing as poor man's scope'. E.g. instead of facebook providing a <like-button> element, they provide <facebook-like-button>. apparently this may be an area of future development (citation?)

Flex boxes layout: intended for smaller page components...

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes

CSS Grid layout intended for overall page layout...

http://www.w3.org/TR/css3-grid-layout/

2.3 Literature Review

2.3.1 Popular Javascript frameworks

2.3.2 Google Polymer framework

2.4 Speakur

My desire to learn more about modern web development led me to investigate web frameworks like Angular and Meteor. I spent some time building (very) simple demos with these two frameworks in particular. Although they are expressive and powerful, and are used every day to power high-traffic applications, I was unhappy with the non-standard and idiosyncratic nature of these frameworks. They relied on 'proprietary' (even if open source) extensions that were not native to HTML and not easily transportable across different frameworks and architectures. This dissatisfaction led me to learn about the Web Components initiative.

2.4.1 Origin

Learning about Web Components quickly led me to the Polymer project. I wanted a component that demonstrated common use cases for Web Components and also showed off some of the design possibilities provided by Polymer and Material Design. I was also intrigued by the possibilities of a server-free design afforded by Firebase. Some kind of 'live' social plugin seemed like a natural fit for the capabilities of Polymer and Firebase, so this led to a discussion plugin for blogs and other articles. My hope was that it would required little or nothing in the way of dedicated server resources in order to actually use it.

2.4.2 Motivations

I wanted my discussion component to have some of the following attributes:

- Provide a simple API to consumers that hid most implementation details.
- Require minimal server resources. Ideally nothing would need to be "installed" and it could be loaded in a cross-origin fashion from online web developer tools like `https://jsbin.com`.

- Support Markdown formatted comments including syntax highlighting for code snippets.

- Support internationalization (i18n) and localization (l10n) features for a global audience.

- Support distributed event notification similar to the publish-subscribe (pub-sub) design pattern.

  In essence, 'live' data updates: when someone replies to a post it should become instantly available to anyone viewing the thread.

- If any framework was used at all, it should be based on Web Components.

  This instantly ruled out the vast majority of frameworks, leaving only Polymer and the less-comprehensive X-Tags project [CITE] and a few other smaller contenders.

In the following section we will discuss some of the high level architectural considerations in designing such a component.

Chapter 3

How to Use the utdiss2 Package

3.1 Preamble

The preamble of the document starts like this:

```
\documentclass[12pt]{report}
\usepackage{utdiss2}
```

The first line declares "`report`" as the document class, with an option of 12pt for the character size, which is slightly greater that usual (the default is 10pt), but is what the Office of the Graduate School (OGS) recommends. You may include other options, as in any other LaTeX document.

The second line loads the `utdiss2` package, which contains a set of commands intended to produce a document fulfilling the official requirements for a doctoral dissertation or master's thesis or report. Besides that, you may include other packages. For instance:

```
\usepackage{amsmath,amsthm,amsfonts,amscd}
```

for mathematical symbols, or,

```
\usepackage{draftcopy}
```

to have a large "watermark" across each page of your document that says, "DRAFT."

The next few commands in the preamble are required.

`\author{Craig William McCluskey}` Replace my name in the command by your full, official University name. Make it combination of lower and upper-cases.

`\address{9905 Chukar Circle\\ Austin, Texas 78758}` Replace my address with your *permanent* (not local) address. Use \\ to separate address lines.

`\title{Writing a Doctoral Dissertation with \LaTeX{} at the University of Texas at Austin}` Replace the name of this document in this command by your dissertation title. If the title consists of more than one line, it should be in inverted pyramid form. You may have to specify the line breakings by \\ commands.

`\supervisor[Isaac Newton]{Johannes Kepler}` This document has two supervisors listed. See the source file (disstemplate.tex) for information on how to have only one supervisor. This command can be broken across lines as it is in the source file and as the `\committeemembers` command is shown below.

```
\committeemembers
        [Erwin Schr\"odinger]
        [Albert Einstein]
        [Charles Townes]
        {Arthur Schawlow}
```

This document shows four non-supervisor committee members. See the source file (disstemplate.tex) for information on how to have a different number.

`\previousdegrees{B.S.}` Replace B.S. with your previous degree.

The next few commands in the preamble are optional.

```
%\graduationmonth{...}
%\graduationyear{...}
%\typist{...}
```

Their use is documented in the source file.

At this point, if you are writing a master thesis or report you must use the optional `\degree` and `\degreeabbr` commands and specify

```
%\degree{MASTER OF ARTS}
%\degreeabbr{M.A.}
%\masterreport
%\masterthesis
```

as documented in the source file. By default the document is formated as a *dissertation*[1]

The default spacing for both text and quoted text is doublespaced. That can be changed with the following self-explanatory commands:

[1]The command `\dissertation` is also provided for symmetry.

```
\oneandonehalfspacing

\singlespacing

\oneandonehalfspacequote

\singlespacequote
```

Some versions of LaTeX in combination with some types of printers produce printed output that has incorrect vertical margins. The command `\topmargin 0.125` is provided to allow easy adjustment if it's needed.

If there are 10 or more sections, 10 or more subsections for a section, etc., you need to make an adjustment to the Table of Contents with the command `\longtocentry`. This command allocates the proper horizontal space for double-digit numbers.

3.2 Document

Next comes the actual text. It could be a sequence of chapters divided into sections, subsections, etc., all in the main file:

```
\chapter{...}      % The first chapter. The
                   % \chapter command is of the form
                   % \chapter[..]{..} or \chapter{..} where
    ... text ...   % [...] is the entry in table of contents
                   % and {...} is the chapter heading printed
                   % in the body of the document.
\section{...}      %
                   % IMPORTANT: If your chapter heading consists
```

```
                  % of more than one line, it will be auto-
    ... text ...  % matically broken into separate lines.
                  % If you don't like the way LaTeX breaks the
                  % chapter heading into lines, however, use
\section{...}     % `\newheadline' command to break lines.
                  % NEVER USE \\ IN SECTIONAL (E.G., CHAPTER,
    ... text ...  % SECTION, SUBSECTION, SUBSUBSECTION) HEADINGS!
                  %
\chapter{...}     % This is Chapter 2.
    ... text ...
\section{...}
    ... text...
\subsection{...}
    ... more text ...
\subsubsection{...}
    ... more text ...
\appendix         % The appendix begins here.
% \appendices     % If more than one appendix chapters,
                  % use \appendices instead of \appendix
\chapter{...}     % First appendix chapter, i.e., Appendix A.
\section{...}     % This is appendix section A.1.
    ................
```

Or, the chapters can be written in different files like this document and be loaded by `\include` commands:

```
\include{chapter-introduction}
\include{chapter-instructions}
\include{chapter-howtouse}
\include{chapter-makingbib}
\include{chapter-tables+figs}
\include{chapter-math}
\appendices
\index{Appendices@\emph{Appendices}}%
\include{chapter-appendix1}
\include{chapter-appendix2}
\include{chapter-appendix3}
```

Having the chapters in separate files makes the main .tex file simpler and allows chapters to be easily re-ordered (just swap the order of the include commands) or left (commented) out for draft copies.

Note: If you have only one appendix, in addition to using `\appendix` instead of `\appendices`, you must leave out the `\chapter` definition at the start of the appendix's text. Putting it in will cause the insertion of an extra page with only the word Appendix on it and will cause the appendix to be labeled Appendix 1, both of which are poor form if there is only one appendix.

If you are writing a short dissertation that does not require chapters, you need to use the command `\nochapters` just before the first section:

```
\nochapters


\section{...}     % First section.
    ... text ...
\section{...}     % Second section.
    ... text ...
        (...)
```

Next comes the bibliography. It can be made by hand like this:

```
\begin{thebibliography}{foo}
\bibitem ...
\end{thebibliography}
```

Or it can also be generated with BiBTEX, as explained in chapter 4.

Finally the vita is produced like this:

```
\begin{vita}
     % Insert your brief biographical sketch here.
     % Your permanent address and the name of the
     % typist(s) are generated automatically.
\end{vita}
```

Chapter 4

Making the Bibliography with BiBTEX

BiBTEX allows one to generate automatically the bibliography from a database of bibliographic items. You need to do the following:

1. Create the bibliographic database, which is a file whose name ends in `.bib`. Let us call it `diss.bib`. Entries in this file are like this:

```
@BOOK{knuth:tb,
  author = "Donald K. Knuth",
  title = "The \TeXbook",
  publisher = "Addison-Wesley",
  year = "1984",
}
@TECHREPORT{poorten:sp,
  author = "Alf~J.~van der Poorten",
  title = "Some problems of recurrent interest",
  institution = "School of Mathematics and Physics,
                 Macquarie University",
  address = "North Ryde, Australia 2113",
  number = "81-0037",
  month = "August",
  year = "1981",
}
@ARTICLE{erdos:oap,
 author = "Paul Erd{\"o}s and Paul Turan",
 title = "On a problem in the theory of uniform
         distribution, {I}",
 journal = "Indag. Math.",
 volume = "10",
 year = "1948",
 pages = "370--378",
}
```

2. Include a `\bibliographystyle` command in your LaTeX file, say `\bibliographystyle{plain}` and a `\bibliography` command to load the bibliography, in this case `\bibliography{diss}`, at the point of your document where the bibliography should be inserted.

   The document at this point will look like this:

   ```
   \bibliographystyle{plain}
   \bibliography{diss}
   ```

3. Run LaTeX on your main file, say `foo.tex`: `latex foo`. This generates an auxiliary file `foo.aux` with a list of `\cite` references.

4. Run BiBTEX on your file: `bibtex foo`. BiBTEX reads the auxiliary file, looks up the bibliographic database (`diss.bib`), and writes a `.bbl` file with the bibliographic information formated according to the bibliographic style file (`.bst`, say `plain.bst`) specified. Messages about resources used and error messages are written to a `.blg` file (in the case of this template, disstemplate.blg).

5. Run LATEX again: `latex foo`, which now reads the `.bbl` reference file.

6. Run LATEX for a third time: `latex foo`, resolving all references.

This includes all bibliographic items that have been cited in the document with a `\cite` command. In order to include non cited items in the bibliography, use the command `\nocite`. For example, `\nocite{knuth:tb}` anywhere in the document (after `\begin{document}`) includes in the bibliography the item with label `knuth:tb`. In order to include *all* items of the bibliographic database, use the command `\nocite{*}`.

Chapter 5

Making Tables and Including Figures

The *tabular* environment allows us to create complex tables and figures, and draw boundaries around and within it. The following example illustrates this:

Table 5.1: An example of a table.

Gegenwart	
ich	bin
du	bist
er	
sie	ist
es	
wir	sind
ihr	seid
sie	sind
Sie	sind

Imperfekt	
ich	war
du	warst
er	
sie	wart
es	
wir	waren
ihr	wart
sie	waren
Sie	waren

Perfekt		
ich	bin	gewesen
du	bist	gewesen
er		
sie	ist	gewesen
es		
wir	sind	gewesen
ihr	seid	gewesen
sie	sind	gewesen
Sie	sind	gewesen

Note: The assistance of Herr Professor Lothar Frommhold
in generating this table of German declensions
is gratefully acknowledged.

This table was created with the following sequence of commands:

```
\begin{table}[h]

\begin{center}

\caption{An example of a table.}
```

```
\vskip 10pt
\begin{tabular}{|ll|l|ll|l|lll|}
\cline{1-2} \cline{4-5} \cline{7-9}
\multicolumn{2}{|c|} {\textsl{Gegenwart}} & &
\multicolumn{2}{|c|} {\textsl{Imperfekt}} & &
\multicolumn{3}{|c|} {\textsl{Perfekt}} \\
\cline{1-2} \cline{4-5} \cline{7-9}
ich & bin  & & ich & war   & & ich & bin  & gewesen \\
du  & bist & & du  & warst & & du  & bist & gewesen \\
er  &      & & er  &       & & er  &      &         \\
sie & ist  & & sie & wart  & & sie & ist  & gewesen \\
es  &      & & es  &       & & es  &      &         \\
\cline{1-2} \cline{4-5} \cline{7-9}
wir & sind & & wir & waren & & wir & sind & gewesen \\
ihr & seid & & ihr & wart  & & ihr & seid & gewesen \\
sie & sind & & sie & waren & & sie & sind & gewesen \\
\cline{1-2} \cline{4-5} \cline{7-9}
Sie & sind & & Sie & waren & & Sie & sind & gewesen \\
\cline{1-2} \cline{4-5} \cline{7-9}
\end{tabular} \\[10pt]
Note: The assistance of Herr Professor Lothar Frommhold \\
in generating this table of German declensions \\
is gratefully acknowledged.
```

```
\vskip -20pt
\end{center}
\end{table}
\index{commands!environments!table}%
```

The argument `h` indicates the position for the table, in this case "here if possible". Other values of this argument are: `t` (top of the page), `b` (bottom of the page), `p` (on the page of floats) and `H` (HERE! - requires using the package float.sty. Note: When this option is used, LaTeX ignores all of its formatting rules and does what you say, putting the entire float exactly where it is defined. Check your output to make sure it is what you want! If you are having trouble with LaTeX wanting to put a figure that's larger than roughly half-a-page, as well as all of the figures following it, at the end of a chapter, try using the command `\clearpage` immediately following the large figure — and maybe a `\newpage` later.) It is possible to combine several arguments, such as `ht` ("here if possible, otherwise on top of the page"). The default is `tbp`.

Figure 5.1 is a typical example of inclusion of a figure contained in an encapsulated PostScript file. In order to use it, it is necessary to include the command `\usepackage{psfig}` at the beginning of the document.

You can see the commands that generated this figure in the source file. Look for the line `\begin{figure}[htb] % Imported eps example.`

The command that imports the file is `\psfig`, and it also controls its size (`height` and `width`), and can rotate the figure (`angle`).

Figure 5.1: An example of an imported jpg file.

Figures can also be drawn by using LaTeX commands. Figure 5.2 is an example (taken from [?]).

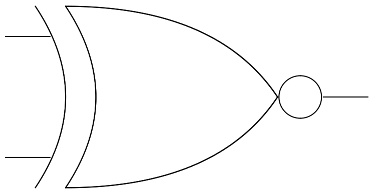


Figure 5.2: An example of a picture

The commands that generated this picture are in the source file following the line `\begin{figure}[htb] % Picture example.`

The commands used have rather obvious meanings. In particular, the command `\qbezier` draws a quadratic Bezier curve, defined by its two ending points, and a third point (whose coordinates are in the middle) that is used as control point. Figure 5.3 illustrates the effect of the control point:

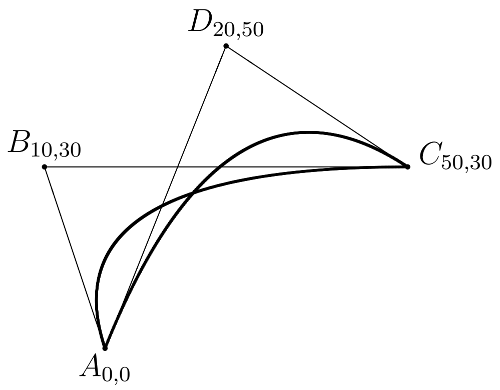


Figure 5.3: Bezier curves

This figure has been generated with the following commands:

```
\begin{figure}[htb] % Bezier curves example.
\begin{center}
  \setlength{\unitlength}{.8mm}
  \begin{picture}(55,55)(-15,0)
    \linethickness{1pt}
    \qbezier(0,0)(-10,30)(50,30)
    \qbezier(0,0)(20,50)(50,30)
    \thinlines
    \put(0,0){\line(-1,3){10}}
    \put(50,30){\line(-1,0){60}}
    \put(0,0){\line(2,5){20}}
    \put(50,30){\line(-3,2){30}}
    \put(0,0){\circle*{1}}
```

```
      \put(0,-1){\makebox(0,0)[t]{$A_{0,0}$}}
      \put(-10,30){\circle*{1}}
      \put(-10,31){\makebox(0,0)[b]{$B_{10,30}$}}
      \put(50,30){\circle*{1}}
      \put(58,29){\makebox(0,0)[b]{$C_{50,30}$}}
      \put(20,50){\circle*{1}}
      \put(20,51){\makebox(0,0)[b]{$D_{20,50}$}}
   \end{picture}
\caption{Bezier curves}
\label{f:qb}
\end{center}
\end{figure}
```

Chapter 6

An Example of Mathematical Writing

6.1 Generalized Fatou's Lemma

Here we show an application of the following lemma:

Lemma 6.1.1 (Generalized Fatou's Lemma)**.** *Let A be a Dedekind ring and F a rational series in $A[[X]]$, i.e., $F = p/q$ for some $p, q \in A[X]$. Then there exist two polynomials $P, Q \in A[X]$ such that $F = P/Q$, where P and Q are relatively prime and $Q(0) = 1$.*

Proof. See [?], p. 15, theorem 1.3. □

Theorem 6.1.2. *Let $\{c_n\}_{n=-\infty}^{\infty}$ a set of elements from K such that $c_n \in k'$ for every $n \geq n_0$, and verifying the following recurrence relation of order M:*

$$c_n = r_1 c_{n-1} + r_2 c_{n-2} + \cdots + r_M c_{n-M} \tag{6.1}$$

for every $n \in \mathbb{Z}$, where $r_1, r_2, \ldots, r_M$ are in K, $r_M \neq 0$. Then:

(i) The coefficients $r_1, r_2, \ldots, r_M$ are in k', and for every $n \in \mathbb{Z}$, $c_n \in k'$.

(ii) If $c_n \in \mathcal{O}_{k',v}$ for every $n \geq n_0$, then the coefficients $r_1, r_2, \ldots, r_M$ are all in $\mathcal{O}_{k',v}$.

Proof.

(i) Let C_n and R be the matrices:

$$C_n = \begin{pmatrix} c_n & c_{n+1} & \dots & c_{n+M-1} \\ c_{n+1} & c_{n+2} & \dots & c_{n+M} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+M-1} & c_{n+M} & \dots & c_{n+2M-2} \end{pmatrix} \tag{6.2}$$

and

$$R = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ r_M & r_{M-1} & r_{M-2} & \dots & r_1 \end{pmatrix} \tag{6.3}$$

We have that $C_{n+1} = R\,C_n$. Since the recurrence relation is of order M, C_n is non singular. On the other hand, $R = C_{n+1}\,C_n^{-1}$. Since the elements of C_n are in k' for $n \geq n_0$, the entries of R, and those of R^{-1}, will be in k'. Since $C_{n-1} = R^{-1}\,C_n$, we get that the entries of C_n will be in k' also for $n < n_0$.

(ii) For each $t \geq n_0$ define the formal power series

$$F_t(X) = \sum_{n=0}^{\infty} c_{t+n}\,X^n \tag{6.4}$$

which is in $\mathcal{O}_{k',v}[[X]]$. We have $F_t(X) = p_t(X)/q(X)$, where $p_t(X), q(X) \in k'[X]$ are the following:

$$p_t(X) = \sum_{j=0}^{M-1} \Big(c_{t+j} - \sum_{i=1}^{j} r_i\,c_{t+j-i} \Big)\,X^j \tag{6.5}$$

$$q(X) = 1 - r_1\,X - r_2\,X^2 - \dots - r_M\,X^M \tag{6.6}$$

This can be checked by multiplying $F_t(X)$ by $q_t(X)$ and using the recurrence relation, which gives $F_t(X)\,q(X) = p_t(X)$ (see [?]).

Now we will prove that $p_t(X)$ and $q(X)$ are relatively prime. To do so, we will see that they cannot have any common root (in $\overline{k'}$). In fact, assume that α is a common root of $p_{t_0}(X)$ and $q(X)$ for some $t_0 \geq n_0$, i.e.: $p_{t_0}(\alpha) = q(\alpha) = 0$. Since $q(0) = 1$, then $\alpha \neq 0$. Now we have:

$$X\, F_{t_0+1}(X) = F_{t_0}(X) - c_{t_0} \tag{6.7}$$

so:

$$\begin{aligned} X\, p_{t_0+1}(X) &= X\, q(X)\, F_{t_0+1}(X) \\ &= q(X)\,(F_{t_0}(X) - c_{t_0}) = p_{t_0}(X) - c_{t_0}\, q(X) \end{aligned} \tag{6.8}$$

Hence $p_{t_0+1}(\alpha) = 0$, which means that α is also a root of $p_{t_0+1}(X)$. By induction we get that $p_t(\alpha) = 0$ for every $t \geq t_0$. Grouping the terms of $p_t(X)$ with respect to $c_t, c_{t+1}, \ldots, c_{t+M-1}$, we get:

$$p_t(X) = \sum_{j=0}^{M-1} a_j(X)\, c_{t+j} \tag{6.9}$$

where

$$a_j(X) = X^j \left(1 - \sum_{i=1}^{M-j-1} r_i\, X^i\right) \tag{6.10}$$

Note that $a_0(X), a_1(X), \ldots, a_{M-1}(X)$ do not depend on t. On the other hand $p_t(\alpha) = 0$ implies

$$\sum_{j=0}^{M-1} a_j(\alpha)\, c_{t+j} = 0 \tag{6.11}$$

for every $t \geq t_0$. Note that $a_{M-1}(\alpha) = \alpha^{M-1} \neq 0$, so $a_0(\alpha), a_1(\alpha), \ldots, a_{M-1}(\alpha)$ are not all zero, and (6.11) means that the columns of the matrix C_{t_0} are linearly dependent, so $\det C_{t_0} = 0$, which contradicts the fact that C_{t_0} is non singular. Hence, the hypothesis that $p_t(X)$ and $q(X)$ have a common root has to be false. This proves that $p_t(X)$ and $q(X)$ are relatively prime.

By (generalized Fatou's) lemma 6.1.1, and taking into account that $\mathcal{O}_{k',v}$ is a Dedekind ring, we get that there exist two relatively prime polynomials $P_t(X)$ and $Q_t(X)$ in $\mathcal{O}_{k',v}[X]$ such that $F_t(X) = P_t(X)/Q_t(X)$ and $Q_t(0) = 1$. Hence: $p_t(X)\,Q_t(X) = q(X)\,P_t(X)$. By unique factorization of polynomials in $k'[X]$, there is a $u \in k'$ such that $P_t(X) = u\,p_t(X)$ and $Q_t(X) = u\,q_t(X)$. Since $Q_t(0) = q(0) = 1$, we get that $u = 1$, so $P_t(X) = p_t(X)$ and $Q_t(X) = q(X)$. Hence, the coefficients of $q(X)$ are in $\mathcal{O}_{k',v}$.

□

6.2 Other Examples of Mathematical Writing

6.2.1 An Example of a Commutative Diagram

The following is an example of a commutative diagram. It requires the `amscd` package.

$$\begin{tikzcd}
S^{\mathcal{W}_\Lambda} \otimes T \arrow[r, "j"] \arrow[d] & T \arrow[d, "\operatorname{End} P"] \\
(S \otimes T)/I \arrow[r, equal] & (Z \otimes T)/J
\end{tikzcd}$$

That diagram has been made with the following commands:

```
\newcommand{\End}{\operatorname{End}}
```

```
\begin{CD}
S^{{\mathcal{W}}_\Lambda}\otimes T   @>j>>   T\\
@VVV                                    @VV{\End P}V\\
(S\otimes T)/I                  @=      (Z\otimes T)/J
\end{CD}
```

6.2.2 Using AMS Fonts

To use AMS fonts it is necessary to choose from an assortment of LaTeX packages. For instance the command `\usepackage{amsfonts}` calls in the *ams-fonts* package, which provides blackboard bold letters (e.g. $\mathbb{R}$) and some math symbols. A superset of that package is *amssymb*. Other packages are *eufrak* for Frankfurt letters (e.g. $\mathfrak{R}$) and *eucal* for Euler script (e.g. $\mathcal{R}$). Consult the LaTeX documentation about this subject for additional information.

Appendices

Appendix A

Lerma's Appendix

The source LaTeX file for this document is no longer quoted in its entirety in the output document. A LaTeX file can include its own source by using the command `\verbatiminput{\jobname}`.

Appendix B

My Appendix #2

B.1 The First Section

This is the first section. This is the second appendix.

B.2 The Second Section

This is the second section of the second appendix.

B.2.1 The First Subsection of the Second Section

This is the first subsection of the second section of the second appendix.

B.2.2 The Second Subsection of the Second Section

This is the second subsection of the second section of the second appendix.

B.2.2.1 The First Subsubsection of the Second Subsection of the Second Section

This is the first subsubsection of the second subsection of the second section of the second appendix.

B.2.2.2 The Second Subsubsection of the Second Subsection of the Second Section

This is the second subsubsection of the second subsection of the second section of the second appendix.

Appendix C

My Appendix #3

C.1 The First Section

This is the first section. This is the third appendix.

C.2 The Second Section

This is the second section of the third appendix.

Vita

Preston Brent Landers was born in Texas and attended high school on the Nevada side of Lake Tahoe. He received his Bachelor of Arts in English from the University of Texas at Austin. He works as a web and mobile software engineer for Journyx, Inc.[*] in Austin, Texas and began graduate studies in Software Engineering at the University of Texas at Austin in August 2012.

Permanent address: `planders@utexas.edu`

This report was typeset with LaTeX[†] by the author.

[*] `http://www.journyx.com`

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.