

Copyright
by
Preston Brent Landers
2015

The Report Committee for Preston Brent Landers
certifies that this is the approved version of the following report:

**Speakur: leveraging Web Components
for composable applications**

APPROVED BY

SUPERVISING COMMITTEE:

Adnan Aziz, Supervisor

Christine Julien

**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 pages out of components sourced from a diverse set of 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

Acknowledgments	v
Abstract	vi
List of Tables	x
List of Figures	xi
Chapter 1. Introduction	1
1.1 Web Components Overview	4
1.2 Structure of This Report	6
1.3 Source Code and Demonstration Resources	8
Chapter 2. Background	9
2.1 Current challenges in web authoring	10
2.1.1 Encapsulation and composition	12
2.2 Web Components	14
2.2.1 Custom HTML elements	14
2.2.2 Shadow DOM	15
2.2.3 HTML Imports	15
2.2.4 Templates	15
2.2.5 Related technologies	15
2.3 Literature Review	15
2.3.1 Popular Javascript frameworks	15
2.3.2 Google Polymer framework	15
2.4 Speakur	15
2.4.1 Origin	15
2.4.2 Motivations	15

Chapter 3. How to Use the utdiss2 Package	16
3.1 Preamble	16
3.2 Document	19
Chapter 4. Making the Bibliography with BiBTeX	23
Chapter 5. Making Tables and Including Figures	26
Chapter 6. An Example of Mathematical Writing	32
6.1 Generalized Fatou’s Lemma	32
6.2 Other Examples of Mathematical Writing	35
6.2.1 An Example of a Commutative Diagram	35
6.2.2 Using AMS Fonts	36
Appendices	37
Appendix A. Lerma’s Appendix	38
Appendix B. My Appendix #2	39
B.1 The First Section	39
B.2 The Second Section	39
B.2.1 The First Subsection of the Second Section	39
B.2.2 The Second Subsection of the Second Section	39
B.2.2.1 The First Subsubsection of the Second Subsection of the Second Section	39
B.2.2.2 The Second Subsubsection of the Second Subsec- tion of the Second Section	40
Appendix C. My Appendix #3	41
C.1 The First Section	41
C.2 The Second Section	41
Vita	42

List of Tables

5.1	An example of a table.	26
-----	--------------------------------	----

List of Figures

1.1	A Speakur thread inside a demonstration page.	2
2.1	A partial example of Twitter Bootstrap navigation bar HTML. . .	11
2.2	Hypothetical Bootstrap nav bar custom element.	12
2.3	Opera's <i>shadow DOM</i> for <video> and its Play button	13
5.1	An example of an imported jpg file.	29
5.2	An example of a picture	29
5.3	Bezier curves	30

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-

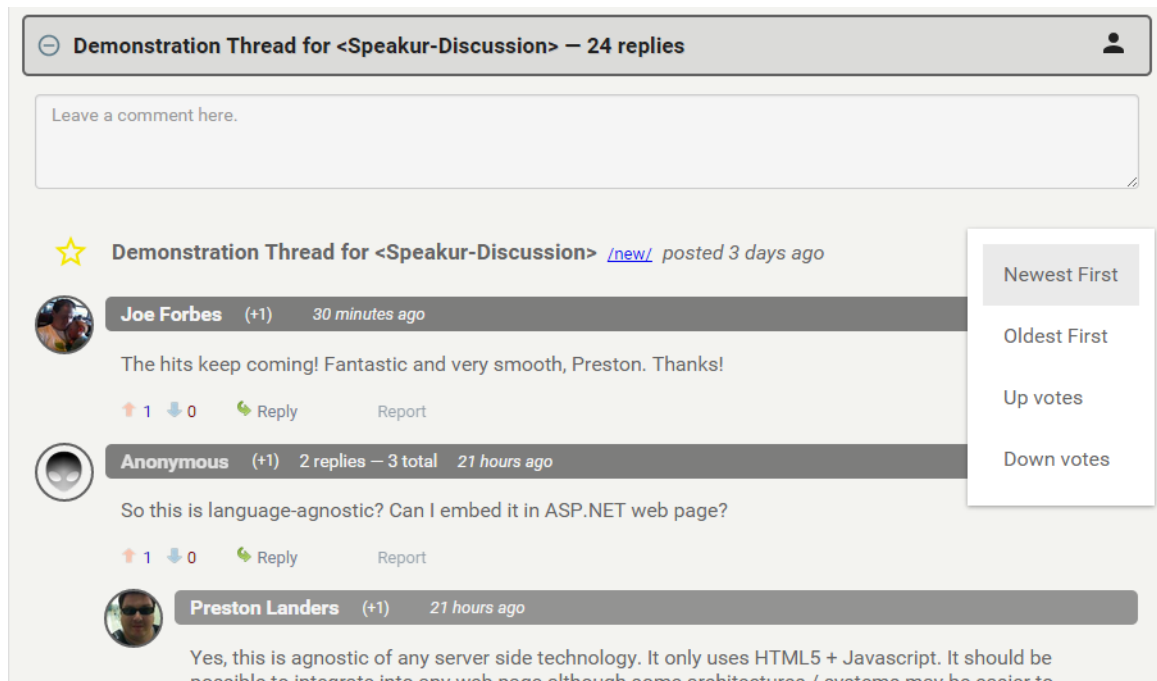


Figure 1.1: A Speakur thread inside a demonstration page.

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 have allowed one to 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 Javascript¹ (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)

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]

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 name ECMAScript (ES).

and the fact that it represents an common point for a fragmented industry to rally 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, there still did not exist a clear picture 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 Google² and Mozilla³ 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.

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

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

³The 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.

- **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. 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 automatic synchronization of 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, composition, and inheritance, as well as technologies such as WebSockets and NoSQL databases. 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 technology and 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.

The Implementation section describes the application of Web Component principles to the specific task of providing a flexible and suitably generic discussion forum / commenting plugin for both desktop and mobile browsers. It describes the overall 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 just all kinds of awesome and wraps up the report (**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 here on 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 facility composing web applications, large or small, from a number of sub-components. 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 follow the same set of assumptions or 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 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 authors that wish to use it. For example, to add a Bootstrap site navigation bar to your page, you must copy and paste a large block of HTML and then customize it to your needs as shown in figure 2.1.

This forces users of the library to tightly couple the layout of their page with the internal structure required by Bootstrap's navigation bar widget. This coupling militates against Bootstrap refactoring or modifying the internal structure of the navigation widget because this would require the large community of users to up-

```

<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>
        </li>
      </li>
    </ul>
  </div>

```

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

date 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. In short, there is no encapsulation between components. In fact, neither HTML5 nor Javascript (ECMAScript 5) have a well-defined notion of a discrete ‘component’ [CITE].

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’ hidden document inside the object that contains the internal state. For example, an author can write `<video loop src=...>` 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 volume controls, a timeline bar and 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 `<div>` for the Play button highlighted.

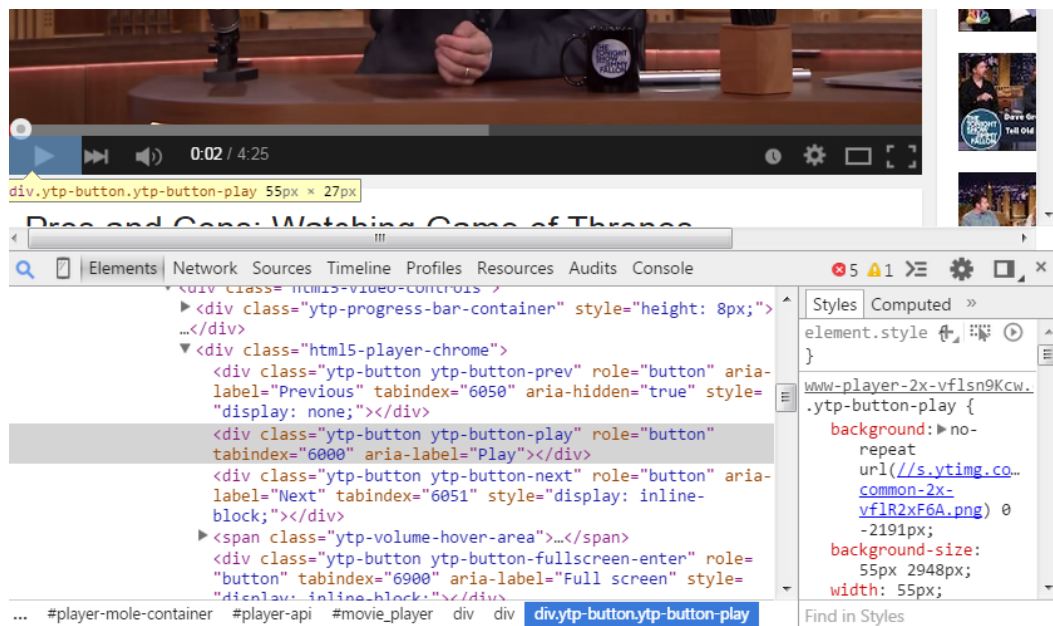


Figure 2.3: Opera's *shadow DOM* for `<video>` and its Play button

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

- Use **encapsulation** and well defined interfaces as the `<video>` element does to protect private state and hide implementation complexity.
- Prefer **composition** or *has-a* relationships over inheritance or *is-a* relationships when building modules. In the case of `<video>`, it's composed of sim-

ple block elements and scoped CSS roles and the Volume and Play controls aren't "special" objects, just `<divs>` with CSS rules and click handlers.

The solution, therefore, to these coupling problems in web authoring was expose these internal browser APIs for creating elements in a safe and portable fashion. This would allow web authors to create their own rich custom elements using standard portable APIs, encapsulate their internals, and share them with others. The question remained, which specific browser features needed to be standardized in order to support Web Components?

2.2 Web Components

The Web Component initiative consists of two main technologies and two supporting features. The main technologies are Custom HTML Elements and the Shadow DOM, while HTML Imports and Templates support these features.

2.2.1 Custom HTML elements

Never before have web authors been able to define their own custom HTML elements not found in the official list. Actually, many authors and web frameworks have done exactly that for years, primarily for internal pre-processing purposes. The custom elements would not get sent to the end-user's browser because the browser would not understand what to do with them. However the possibility now exists to create custom elements in a standard way that will be supported by browsers. The primary restriction to avoid a name collision with future built-in

HTML elements is that all custom elements must have a - character (dash) in their name, such as `<my-element>`.

2.2.2 Shadow DOM

Shadow DOM is available to any element, not just custom elements, but the primary need for them is to encapsulate the internals of custom elements.

2.2.3 HTML Imports

2.2.4 Templates

2.2.5 Related technologies

2.3 Literature Review

2.3.1 Popular Javascript frameworks

2.3.2 Google Polymer framework

2.4 Speakur

2.4.1 Origin

2.4.2 Motivations

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 than 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 L^AT_EX 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 formatted 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:

¹The 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 BiB_TE_X, 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 BiB_TE_X

BiB_TE_X 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 BiB_TE_X on your file: `bibtex foo`. BiB_TE_X reads the auxiliary file, looks up the bibliographic database (`diss.bib`), and writes a `.bbl` file with the bibliographic information formatted 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 L^AT_EX again: `latex foo`, which now reads the `.bbl` reference file.
6. Run L^AT_EX 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.

<i>Gegenwart</i>	<i>Imperfekt</i>	<i>Perfekt</i>
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
wir sind	wir waren	wir sind gewesen
ihr seid	ihr wart	ihr seid gewesen
sie sind	sie waren	sie sind gewesen
Sie sind	Sie waren	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 -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 [?]).

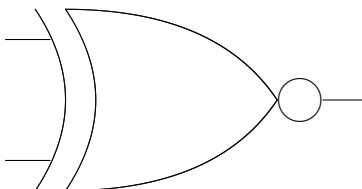


Figure 5.2: An example of a picture

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:

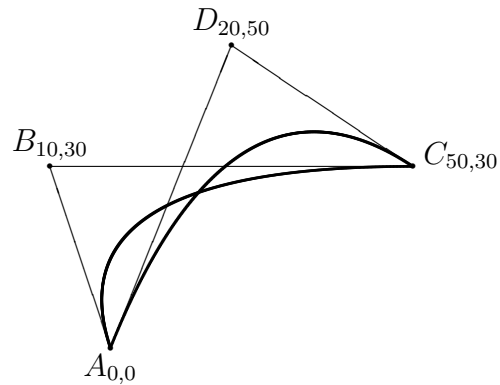


Figure 5.3: Bezier curves

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 :*

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

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

- (i) The coefficients r_1, r_2, \dots, 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, \dots, r_M are all in $\mathcal{O}_{k',v}$.*

Proof.

(i) Let C_n and R be the matrices:

$$C_n = \begin{pmatrix} c_n & c_{n+1} & \cdots & c_{n+M-1} \\ c_{n+1} & c_{n+2} & \cdots & c_{n+M} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+M-1} & c_{n+M} & \cdots & c_{n+2M-2} \end{pmatrix} \quad (6.2)$$

and

$$R = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ r_M & r_{M-1} & r_{M-2} & \cdots & r_1 \end{pmatrix} \quad (6.3)$$

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

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

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:

$$p_t(X) = \sum_{j=0}^{M-1} \left(c_{t+j} - \sum_{i=1}^j r_i c_{t+j-i} \right) X^j \quad (6.5)$$

$$q(X) = 1 - r_1 X - r_2 X^2 - \cdots - r_M X^M \quad (6.6)$$

This can be checked by multiplying $F_t(X)$ by $q(X)$ 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:

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

so:

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

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 $c_t, c_{t+1}, \dots, c_{t+M-1}$, we get:

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

where

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

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

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

for every $t \geq t_0$. Note that $a_{M-1}(\alpha) = \alpha^{M-1} \neq 0$, so $a_0(\alpha), a_1(\alpha), \dots, a_{M-1}(\alpha)$ 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.

$$\begin{array}{ccc} S^{\mathcal{W}_\Lambda} \otimes T & \xrightarrow{j} & T \\ \downarrow & & \downarrow \text{End } P \\ (S \otimes T)/I & \xlongequal{\quad} & (Z \otimes T)/J \end{array}$$

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{\text{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 *amsfonts* 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^\dagger by the author.

*<http://www.journyx.com>

$^\dagger\text{\LaTeX}$ is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's \TeX Program.