Displaying Heraldic Blazons

WILLIAM MATHEWSON

4th Year Project Report
Computer Science

SCHOOL OF INFORMATICS

UNIVERSITY OF EDINBURGH

2018

Acknowledgements

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*William Mathewson*)

4TH YEAR PROJECT REPORT

COMPUTER SCIENCE
SCHOOL OF INFORMATICS
UNIVERSITY OF EDINBURGH

CONTENTS

1

INTRODUCTION

1.1 Motivation

In 1874 — 4 years after his death — John Papworth's *Ordinary of British Armorials* was published.[1] In this work, he recorded approximately 50,000 entries of descriptions of families' coats of arms, none annotated.

[1] S. M. Collins. 'Papworth and his Ordinary'. In: *The Antiquaries Journal* 22.1 (1942). Accessed: January 2018, pp. 6–7. DOI: 10.1017/S0003581500003668.

This honours project would make it possible to have these descriptions, or *blazons* as they are termed in heraldry (see 2.1), drawn freely for people to view. This has potential application for ancestry companies that build family trees for people. Given the blazon, they would be able to construct the shields visually.

1.2 Contributions

In this honours project, my contributions included:

- Drawing the charges and quarters used on the shield, *escutcheon*,
- Writing the base web server and
- Writing the quarter and charge drawing algorithm

2

BACKGROUND

2.1 *Heraldry*

Many families, countries and organisations — primarily in Europe — have coats of arms. At the centre of a coat of arms is a shield known as an *escutcheon*. The language used to describe how the escutcheon is to be drawn is known as a *blazon*. Blazons have been used since the Norman conquest and have been refined to a regular language in the process,[1] although, as John Brooke-Little said, "many of the supposedly hard and fast rules laid down in heraldic manuals [including those by heralds] are often ignored."[2] This flagrant disregard for the rules introduces difficulty in parsing the blazons as the language loses some of its regularity.

[1] Charles Boutell. *Heraldry, historical and popular*. Third edition. Accessed: January 2018. London, Bentley, 1864, pp. 8–9.

[2] J. P. Brooke-Little. *An Heraldic Alphabet*. New and revised edition. Accessed: January 2018. London, Robson Books, 1985.

Blazons have a few key attributes:

- The *field*, which is the background of the shield;
- *Ordinaries*, which are geometric shapes (an example of which can be seen in Figure 2.3);
- *Charges*, which are small emblems, such as fleur-de-lis and lions;
- *Variations*, which describe how the field or charge are patterned. Variations can indicate patterns such as chequered or coloured lines (an example can be seen in Figure 2.1); and
- *Tinctures*, which are the colours and patterns for charges, ordinaries and fields.

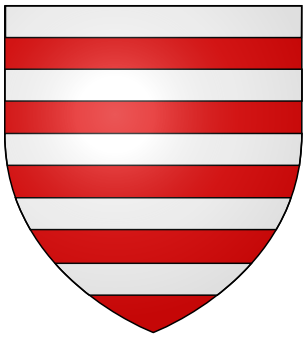


Figure 2.1: The shield of the town of Albert, France. *Barry of ten argent and gules.* Source: `https://en.wikipedia.org/wiki/File:Blason_Albert.svg`

The tinctures are derived from Norman French and are divided into 3 groups, typically known as *metals*, *colours* and *furs*. In British heraldry, the colours are derived from Norman French and so the names appear archaic. In heraldry, blue is *azure* and red is *gules* for instance. The metals are *or* and *argent* for gold and silver respectively. Whilst the tinctures are linked to colours, the College of Arms does not which shade of that colour is required for the tinctures, leaving it to the artist to decide.[3]

[3] *FAQs: heraldry - College of Arms*. Accessed: January 2018. College of Arms. URL: `http://www.college-of-arms.gov.uk/resources/faqs`.

Blazons conventionally follow a form of starting with the *tincture* or *variation* of the field. After the description of the field, *ordinaries* and *charges* are named with their tinctures. An example of this is "*Purpure, a chief Gules*". This blazon describes an escutcheon with a field of *purpure* (purple), with a *Chief* ordinary — a bar across the top of the shield — of *gules* (red). This can be seen drawn by the web app in Figure 2.2.

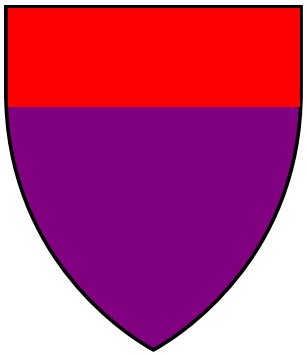


Figure 2.2: *Purpure, a chief Gules.*

A simple — but notable — blazon is that of the Scrope family. In the 14th century, the Baron Scrope brought a case action against Sir Roberts Grosvenor when he noticed that they both had the same coat of arms. Many witnesses gave evidence in the case, including Geoffrey Chaucer.[4] The case was ultimately decided in Scrope's favour. The Scrope coat of arms has a blazon of *Azure, a bend Or*; a depiction of this as drawn by the web app written for this project can be seen in Figure 2.3.

[4] Sir N. Harris Nicolas. *The Controversy between Sir Richard Scrope and Sir Robert Grosvenor in the Court of Chivalry*. Accessed: January 2018. London, Bentley, 1832, p. 404.

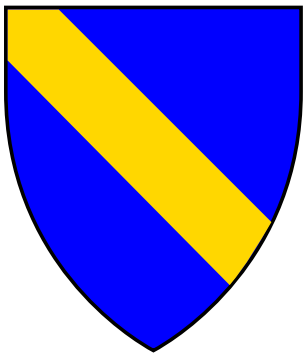
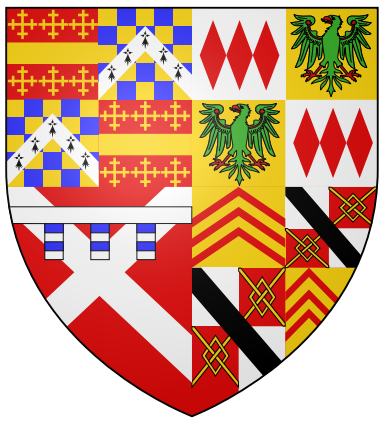
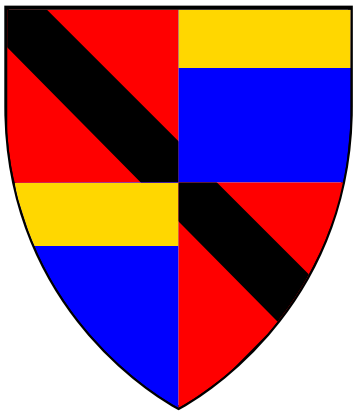


Figure 2.3: The Scrope escutcheon; *Azure, a bend Or.*

Whilst the Scrope arms are prominent in heraldry, they are simplistic and indicative of mediæval arms. Coats of arms became more complex as they developed through the centuries, with instances of *quarterly* shields, *grand-quarterlies* — quarterlies within quarterlies — and *differenced* arms. *Differenced* arms involve adding an *ordinary* over an existing coat of arms. This was typically used to differentiate similar looking coats of arms, especially between father and sons. Common examples of differentiated shields are seen in duchies' coats of arms, particularly those which were given to Charles II's illegitimate children. Examples of more complex shields can be seen in Figure 2.4.



(a) Neville, 16th Earl of Warwick's coat of arms. An example of grand-quarterlies and differenced arms. Source: `https://en.wikipedia.org/wiki/File:Neville_Warwick_Arms.svg`.



(b) A quarterly shield drawn by the web app. *Quarterly: 1st and 4th: Gules, a bend Sable; 2nd and 3rd: Azure, a chief Or.*

Figure 2.4: Some examples of more complex coats of arms.

For a time, it was considered bad form to repeat a *tincture* in a blazon, and use a reference to the tincture's previous use. The Heraldic Society gives an example as such: "'*Azure on a fess argent three billets azure*' [would have been written as] '*Azure on a fess argent three billets of the first*'". The '*of the first*' refers to the field's tincture of azure. This blazon describes a blue shield, with a white bar hori-

zontally across the middle with 3 white rectangles arranged along the bar. The Heraldic Society advocates repeating tinctures to reduce ambiguity.[5]

[5] *Blazon in CoA | The Coat of Arms*. Accessed: January 2018. The Heraldic Society. URL: http://www.the-coat-of-arms.co.uk/blazon-in-coa/.

2.2 Related Works

Add more related works

Whilst many escutcheons have been drawn and uploaded to WikiMedia in SVG format (some of which have been used in this report), many appear to have been created in Inkscape,[6] rather than being created programmatically. Much work has been done in collecting and cataloguing blazons themselves — namely John Papworth as mentioned in Section 1.1.

[6] Inkscape Team. *Inkscape: A vector drawing tool*. Accessed: October 2017. 2004. URL: http://www.inkscape.org.

2.3 Summary

In this chapter, we covered basic heraldry, including core terminology, as well as works related to the project. Core heraldry terminology includes:

- *Escutcheon* — the shield in the coat of arms;
- *Field* — the background of the escutcheon;
- *Ordinaries* — geometric shapes on the escutcheon;
- *Charges* — small emblems, such as fleur-de-lis and lions; and
- *Tincture* — the colours and patterns for charges, ordinaries and fields.

3

DESIGN

3.1 Core Concepts

The core design for this project centred around having a split stack; with a Python back-end parsing the blazon, serialising it into JSON and passing it to the TypeScript front-end, which would then draw it onto the webpage. This allowed for large amounts of flexibility, allowing the two halves of the project to be developed in tandem with the Separation of Concerns principle being adhered to throughout. It also allows for pluggable rendering implementations as the JSON schema for drawing payloads can be well-defined. Python seemed like an obvious choice with its good support for Natural Language Processing (NLP) through the Natural Language Tool Kit (NLTK). TypeScript was chosen as a nicer alternative to programming in pure JavaScript, thanks to the addition of powerful features such as types, access control and abstract classes.

The Python back-end would receive a JSON payload from the webpage containing the blazon; it would then parse the blazon using a Context-Free Grammar (CFG) parser and identify the most important parts of the blazon. It would then serialise these back into a JSON response to be sent to the webpage for rendering. The specification was such that if the webpage was given a blazon of "Azure, a bend Or", it would return a JSON payload of:

```
{
  "field": "azure",
  "charges": [
    {
      "charge": "bend",
      "tincture": "or"
    }
  ]
}
```

The TypeScript front-end would receive this payload, apply the

azure CSS class to the field element, then draw a bend onto the field with an *or* CSS class.

Escutcheons would be drawn using the SVG format for portability across browsers as well as the defined scalability of SVG images. This allows drawn escutcheons to be embedded elsewhere with ease, either directly or through rendering the SVGs as other image formats via programmes like Inkscape.

The back-end was written by my partner, Anthony Gallagher, and the writing of it will not be covered in this report.

3.2 External Dependencies

The front-end would depend on a pair of libraries for Scalable Vector Graphics[1] (SVG) rendering and Document Object Model[2] (DOM) manipulation: the selection library of D3.js[3] and jQuery.[4] D3.js provides many powerful functions for SVG and DOM manipulation, especially creating and editing elements. It would also rely on jQuery for further DOM manipulation.

For further assets, SASS[5] would be used as a CSS pre-processor and Bootswatch[6] would be used for the base styling. Webpack[7] would be used to transpile TypeScript down to JavaScript — minifying and uglifying it in the process — and concatenates all source files and their dependencies into one main JavaScript *bundle* file. *Minification* of JavaScript assets involves stripping out all unnecessary whitespace and tokens. *Uglification* transforms the JavaScript code by renaming all variables and functions into short, obfuscated names to reduce the footprint of the assets. These two techniques can decrease loading times of web apps as the browser have smaller asset payloads to download than the original, raw source code.

[1] Jon Ferraiolo, Fujisawa Jun and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. Accessed: October 2017. iuniverse, 2000.

[2] Lauren Wood et al. *Document Object Model (DOM) level 3 core specification*. Accessed: February 2018. 2004.

[3] Mike Bostock. *D3.js - Data-Driven Documents*. Accessed: September 2017. 2017. URL: https://d3js.org.

[4] The jQuery Foundation. *jQuery*. Accessed: October 2017. 2017. URL: https://jquery.com/.

[5] Hampton Catlin, Natalie Weizenbaum and Chris Eppstein. *Sass: Syntactically Awesome Stylesheets*. Accessed: September 2017. 2017. URL: https://sass-lang.com/.

[6] Thomas Park. *Bootswatch: Flatly*. Accessed: September 2017. 2017. URL: https://bootswatch.com/flatly/.

[7] JS Foundation. *webpack*. Accessed: October 2017. 2017. URL: https://webpack.js.org.

3.2.1 Development Dependencies

To maintain code readability and prevent bugs, TSLint[8] would be used and set up to automatically run as part of the Travis Continuous Integration[9] (CI) service, causing a build to fail if the linter detected a style violation. For unit testing, Jest[10] (with ts-jest[11] for TypeScript support) would be used, especially for its powerful mocking and expectation matcher functionality. Automated documentation generation would be provided by TypeDoc.[12]

[8] Palantir Technologies. *TSLint*. Accessed: September 2017. 2016. URL: https://palantir.github.io/tslint/.

[9] Travis CI GmbH. *Travis CI - Test and Deploy with Confidence*. Accessed: November 2017. 2018. URL: https://travis-ci.com.

[10] Inc. Facebook. *Jest - Delightful JavaScript Testing*. Accessed: February 2018. 2018. URL: https://facebook.github.io/jest/.

[11] Kulshekhar Kabra. *ts-jest*. Accessed: February 2018. 2016. URL: https://github.com/kulshekhar/ts-jest.

[12] TypeDoc Contributors. *TypeDoc - Documentation generator for TypeScript projects*. Accessed: February 2018. 2016. URL: http://typedoc.org.

3.3 Initial Design

The initial design had a specific focus on basic charge drawing, with a plan to produce a new design for adding quarterly rendering with

lessons learnt from implementing charge drawing. The later design is described in Section 4.2.

All drawing logic would be client-side, existing in a single module with minimum dependencies. The shield outline would be rendered on the page on load as part of the HTML template. This helps support a reliable entry point for the drawing logic as it can easily select the shield element to begin appending other SVG elements to. Appending these SVG elements allows layering to be achieved as SVG orders layers based on the order of elements in the document. This would be used to great effect later when drawing quarters (see Section 4.2).

The front-end would be designed around functional paradigms; breaking up major functionality into functions that would deal with smaller, encapsulated functionality, such as adding extra layers to the HTML template or clearing the shield when drawing a new blazon. This would allow for a steady API, as the single point of access function would not be renamed but all other functions may be changed and updated separately. As described in Section 3.1, the front-end would first access the *field* value in the parsed JSON payload, apply the value as the CSS class for the shield and then move onto the charges. It would then iterate over the *charges* array in the payload, drawing each charge onto the shield and applying the tincture as the CSS class. Due to SVG layering, as mentioned earlier, if there were multiple charges specified in the payload, all would be drawn on in the order specified in the array ordering.

4

IMPLEMENTATION

4.1 Initial Approach

As described in Section 3.3, the initial approach was to have all methods in the core `index.ts` file that would be transpiled and loaded in the browser. This meant a smaller footprint when the code was bundled by Webpack and easier maintenance as all relevant functions were next to one another, following the Step-Down Rule.[1]

[1] Robert C Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009, p. 37.

The web app had a single entry point of `drawShield(blazon)`, where `blazon` was the whole JSON payload returned from the `/_parse` endpoint. (See Section 3.1 for an example payload.) This presented a problem initially as TypeScript didn't handle the unstructured parsed data well due to it being a JavaScript `Object` instance. Attempting to access members of this object (such as `field`) causes TypeScript to produce an error that the contents might be undefined and thus return a `null` object. To fix this, I designed `interfaces` with the expected fields in the payload; one for the whole object, `IBlazon`, and one for the charges contained within, `ICharge`. Similarly, to avoid problems with strings not matching, I defined 2 `enums` to represent the supported tinctures and charges, `ETincture` and `ECharge` respectively. As discussed in Section 4.2, I later added another `enum` for quarters. (All `interfaces` and `enums` can be found in Appendix A.) Having fixed this data problem, `drawShield(blazon)` was now able to access members of the blazon object safely.

To avoid the problem of overlapping charges, I had to write a `clearShield()` method that would iterate over all the `path` nodes in the SVG block, and delete them. This, however, promptly deleted the shield outline, so I had to add a check to prevent deleting `path` nodes with a `#shield` id, instead only removing the CSS class. Having cleared the shield of any possible obstructions, the `drawShield` method would then assign the contents of the `field` value as the CSS class and iterate over the `charges` array, passing each charge to `drawCharge(charge: ICharge)`.

To draw shapes in SVG, a `path` node requires a `d` attributes which contains the commands for drawing said shape. To generate all these attributes, I drew all the charge shapes in Inkscape and extracted the `d` attribute from the generated SVGs.

4.2 Adding Quarterly Rendering

4.3 Refactoring Charge Rendering

5

RESULTS AND DISCUSSION

6

CONCLUSION

BIBLIOGRAPHY

Blazon in CoA | The Coat of Arms. Accessed: January 2018. The Heraldic Society. URL: http://www.the-coat-of-arms.co.uk/blazon-in-coa/.

Bostock, Mike. *D3.js - Data-Driven Documents*. Accessed: September 2017. 2017. URL: https://d3js.org.

Boutell, Charles. *Heraldry, historical and popular*. Third edition. Accessed: January 2018. London, Bentley, 1864, pp. 8–9.

Brooke-Little, J. P. *An Heraldic Alphabet*. New and revised edition. Accessed: January 2018. London, Robson Books, 1985.

Catlin, Hampton, Natalie Weizenbaum and Chris Eppstein. *Sass: Syntactically Awesome Stylesheets*. Accessed: September 2017. 2017. URL: https://sass-lang.com/.

Collins, S. M. 'Papworth and his Ordinary'. In: *The Antiquaries Journal* 22.1 (1942). Accessed: January 2018, pp. 6–7. DOI: 10.1017/S0003581500003668.

Contributors, TypeDoc. *TypeDoc - Documentation generator for TypeScript projects*. Accessed: February 2018. 2016. URL: http://typedoc.org.

Facebook, Inc. *Jest - Delightful JavaScript Testing*. Accessed: February 2018. 2018. URL: https://facebook.github.io/jest/.

FAQs: heraldry - College of Arms. Accessed: January 2018. College of Arms. URL: http://www.college-of-arms.gov.uk/resources/faqs.

Ferraiolo, Jon, Fujisawa Jun and Dean Jackson. *Scalable vector graphics (SVG) 1.0 specification*. Accessed: October 2017. iuniverse, 2000.

Foundation, JS. *webpack*. Accessed: October 2017. 2017. URL: https://webpack.js.org.

Foundation, The jQuery. *jQuery*. Accessed: October 2017. 2017. URL: https://jquery.com/.

GmbH, Travis CI. *Travis CI - Test and Deploy with Confidence*. Accessed: November 2017. 2018. URL: https://travis-ci.com.

Kabra, Kulshekhar. *ts-jest*. Accessed: February 2018. 2016. URL: https://github.com/kulshekhar/ts-jest.

Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009, p. 37.

Nicolas, Sir N. Harris. *The Controversy between Sir Richard Scrope and Sir Robert Grosvenor in the Court of Chivalry*. Accessed: January 2018. London, Bentley, 1832, p. 404.

Park, Thomas. *Bootswatch: Flatly*. Accessed: September 2017. 2017. URL: https://bootswatch.com/flatly/.

Team, Inkscape. *Inkscape: A vector drawing tool*. Accessed: October 2017. 2004. URL: http://www.inkscape.org.

Technologies, Palantir. *TSLint*. Accessed: September 2017. 2016. URL: https://palantir.github.io/tslint/.

Wood, Lauren et al. *Document Object Model (DOM) level 3 core specification*. Accessed: February 2018. 2004.

Appendix A

Interfaces and Enums

```
enum ETincture {
  /** For specifying Quarters */
  Quarterly = "quarterly",
  /** Gold/yellow */
  Or = "or",
  /** White */
  Argent = "argent",
  /** Blue */
  Azure = "azure",
  /** Red */
  Gules = "gules",
  /** Purple */
  Purpure = "purpure",
  /** Black */
  Sable = "sable",
  /** Green */
  Vert = "vert",
}

enum ECharge {
  Bend = "bend",
  Cross = "cross",
  Chief = "chief",
  Saltire = "saltire",
}

enum EQuarter {
  TL = "quarterly_tl",
  TR = "quarterly_tr",
  BL = "quarterly_bl",
  BR = "quarterly_br",
}

interface ICharge {
  charge: ECharge;
```

```
  sinister?: boolean;
  tincture?: ETincture;
}

interface IBlazon {
  field: ETincture;
  charges: ICharge[];
}
```