ul*q*da: A LaTeX package supporting Qualitative Data Analysis

Ivan Griffin
`ivan.griffin@ul.ie`

2009/06/11

Abstract

ul*q*da is a LaTeX package for use in Qualitative Data Analysis research. It assists in the analysis of textual data such as interview transcripts and field notes. This document corresponds to ul*q*da v1.1, dated 2009/06/11.

Contents

1 Introduction **2**
1.1 What is Qualitative Data Analysis? 2
1.2 What does this package do? 2
1.3 Why is this package named ul*q*da? 2
1.4 Acknowledgements 3
1.5 Legal Mumbo-Jumbo 3

2 Prerequisites **3**

3 Known Limitations and Issues **4**
3.1 docstrip woes - in this very document! 4

4 Why use LaTeX for QDA Automation? **4**

5 Installation **6**

6 Usage **6**
6.1 Options 6
6.1.1 Advanced Options Usage 7
6.2 Macros 8
6.3 Example 9
6.3.1 Coding Example 9
6.3.2 Typeset Example 10
6.3.3 CSV Cache File 10
6.3.4 Visualisation as a Table 10

6.3.5 Visualisation as a Cloud 11
6.3.6 Visualisation as Graphs 11

7 Implementation 14
7.1 Dependencies 14
7.2 Highlighting Style 14
7.3 Package Options 14
7.4 Testing the Shell Escape Mechanism 15
7.5 Active Macro Implementation 15
7.6 Inactive Macro Stubs 22

1 Introduction

This document describes ulqda, a LaTeX package which supports the integration of Qualitative Data Analysis (QDA) research tasks, specifically for Grounded Theory, into the LaTeX work flow. For a quick start example, see section 6.3.

1.1 What is Qualitative Data Analysis?

Qualitative Data Analysis is a field of inquiry that is popular in social science research [1]. Scientific methods within QDA aim to gain comprehensive and holistic understandings of the motivations for human behaviour in many different situations.

Grounded Theory is a qualitative methodology that emphasises the generation of new theory from its natural emergency through the process of continual collection, compaction and analysis [2, 3].

1.2 What does this package do?

The ulqda package provides the LaTeX user with macros which are used to markup textual information - for example, in-depth interviews - in order to facilitate the distillation of emerging themes from the data in a consistent and reliably manner, and to support visualisation of these themes.

In other words, this package lets the computer do the grunt work, and the researcher focus on recognising and comprehending the emerging theories from the work.

The package works by creating a comma-separate values (CSV) cache file of the codes and associated text it finds in your LaTeX source. It then post-processes this CSV file to GraphViz Dot language, and uses `dot2texi.sty` to optionally render this data as graphs. The filename for the CSV file is generated automatically from the LaTeX current jobname.

1.3 Why is this package named ulqda?

The name ulqda is simply the initials of my *alma mater*, the University of Limerick, prepended onto the abbreviation QDA to generate a unique name. The ulqda prefix is used within the package on macro names and conditionals to prevent naming clashes.

1.4 Acknowledgements

Special thanks to Marc van Dongen and Peter Flynn of the Irish TEX and LATEX In-Print Community for their assistance in creating the LATEX macro to perform the coding.

Thanks to Kjell Magne Fauskes for the excellent `dot2tex` and `dot2texi.sty` packages.

And finally, a special shout out to Matthias Noe for pointing out some issues with an earlier version of this package.

1.5 Legal Mumbo-Jumbo

2 Prerequisites

ulqda requires the use of pdfeTEX. The following LATEX packages, available on CTAN, are needed by the ulqda package:

- color.sty - provides LATEX support for colour;
- soul.sty - provides support for highlighting text;
- multicol.sty - defines an environment for typesetting text in multiple columns;
- PGF/Ti*k*Z - macro package for the creation of graphics in TEX;
- dot2texi.sty - allows the embedding of GraphViz graphs (described in Dot language) in LATEX documents.

In addition, the following external tools are required for processing and graph/list generation:

- GraphViz is a tool to automate graph visualisation [4], a means of graphically ‘representing structural information as diagrams of abstract graphs and networks’;

- dot2tex is a tool for converting graphs generated by GraphViz to PGF/Ti*k*Z that can be rendered with LaTeX [5];
- Perl and the Digest::SHA1 Perl Module are used to automate the conversion of coded output to Dot language.

3 Known Limitations and Issues

For some reason, the underlining trick provided by soul.sty and used by this package fails to work when a color model option is passed to xcolor.sty. The trouble seems to be with soul.sty's `\texthl{}` macro.

A rather unsatisfactory workaround is to redefine `\ulqdaHighlight` to something like the following, somewhere in your own document after you have used `\usepackage[cmyk]{xcolor}` and `\usepackage{ulqda}`:

```
\renewcommand{\ulqdaHighlight}[2]{%
  \colorbox{UlQda@lightblue}{\mbox{#2}}
  \marginpar%
  {\raggedright\hbadness=10000\tiny\it\colorbox{UlQda@lightblue}{#1}\par}%
}
```

Note however that this is not without its own typesetting abberations.

3.1 docstrip woes - in this very document!

As I am using a single `.dtx` file to produce both `ulqda.sty` and `ulqda.pl`, I used mechanisms to separate each - notably `<package>` and `<perl>` filters established with the docstrip `\generate` macro. However, for some reason these filters are being output in the typeset source listings for the LaTeX macros in this document. Unfortunately, the docstrip documentation is suitable terse and has not as of yet enlightened me as to how to fix this issue. Please ignore them - or better, suggest the fix!

4 Why use LaTeX for QDA Automation?

An obvious question at this point is why use LaTeX for QDA work flow automation? Surely there are plenty of commercial offerings on the market that can perform the same or similar task?

In my opinion, incorporating the coding markup into the LaTeX typesetting flow has a number of benefits:

- it helps keep coding near the data - developer Brad Appleton describes this well [6]:

  > '*The likelihood of keeping all or part of a software artifact consistent with any corresponding text that describes it, is inversely proportional to the square of the cognitive distance between them.*'

Appleton also expands on the concept of cognitive distance [6]:

> '*The phrase "out of sight, out of mind" gives a vague indication of what is meant by "cognitive distance" ... it relates to the interruption of "flow" of the developers' thoughts between the time they first thought of what they needed to do, and the time and effort expended before they were actually able to begin doing it.* '

- coding can easily be output as a recorded high-quality typeset deliverable - this is possible with other commercial tools, although the output is not as aesthetic as using LaTeX - it is certainly more difficult to do this with pen, paper and scissors techniques;
  - in addition, typesetting the coded data is very valuable - it allows others to check the validity of the output (theme emergence and theory building) of your work, and provides a resource for subsequent (perhaps affiliated) researchers to use (subject to confidentiality and disclosure agreements, etc.)
  - Using LaTeX allows you to easily keep the interviews typographically consistent with the styles and notations used in the main dissertation;
- it allows for a significant degree of flexible in the work flow, limited primarily by your imagination, and not by the functionality of a commercial package. A LaTeX based scheme can 'fit naturally into a work flow where there are many tools, each good at its own job'[7]. As the LaTeX typesetting run itself is generating the coded output data in an easily accessible format (comma-separated values), it is possible to post-process this and visualize the data in a number of different ways:
  - coupled with an appropriate version control system, the LaTeX QDA work flow can provide full traceability of a theme from the collection of source interview data, condensation into codes, iterative refinement of these codes into orthogonal and related sets, and presentation/visualisation of the generated ontologies;
  - it is possible to generate 'heatmaps', mixing qualitative analysis with some element of quantitative analysis, and to use color coding or font/size scaling based on frequency of occurrence of certain codes or themes;
  - it is also possible to visually recognize saturation occurring in emerging themes - again through the use of appropriate color coding of new themes on a per-interview basis - the output format includes the document section information per code to facilitate this post-processing;
- this package and the LaTeX typesetting system are freely available - you may be unwilling or unable to pay for commercial software;

5 Installation

The package `ulqda` is distributed as `dtx` archive together with a corresponding `Makefile`. `dtx` files are text files which combines a LaTeX package with other helper files and documentation for its own code.

In order to install this package, you must:

1. Run `make` to use the supplied `Makefile`. This will extract the macro and script files from the `dtx` archive, and it will also generate documentation for the packages user interfaced and code: When built with `make`, the following files are generated:
    - `ulqda.pdf` - contains this documentation;
    - `ulqda.sty` - contains the actual macro implementations;
    - `ulqda.pl` - a helper script to parse the CSV output.
2. Copy `ulqda.sty` to either the working directory of your current LaTeX project, or to your personal TeX tree. For Unix users, the procedure to copy to your personal tree is:
    ```
    $ make
    $ mkdir -p ~/texmf/tex/latex/ulqda
    $ cp ulqda.sty ~/texmf/tex/latex/ulqda
    ```
3. Tell TeX to re-index its directories to enable it to recognize the new package:
    ```
    $ texhash ~/texmf
    ```
4. Copy `ulqda.pl` to a directory in your path. Again, for Unix users, the procedure to do this is as follows:
    ```
    $ cp ulqda.pl ~/bin
    ```

6 Usage

We will now look at how the package is used - how to set its various options, the macros it provides, and an example of its operation.

6.1 Options

To use the package in your LaTeX document, insert `\usepackage[...]{foo}` in the preamble. There are a number of options which can be passed to the package:

- `active`: The default is inactive. If this option is not specified, the `ulqda` package will be inactive and the document will be typeset as if the `ulqda` package were not loaded, except that all macros defined by the package are still legal but only the `\ulqdaHighlight` macro has an effect.

  This allows final typesetting of the document and for page numbering to stabilize before running through for a coding pass. The recommendation is to activate for the last two LaTeX passes through the document - that way the

CSV file is generated once page numbering is allowed to settle. To activate subsequently, it is possible to invoke LaTeX as follows:

```
$ pdflatex --shell-escape "\PassOptionsToPackage{active}{ulqda}
     \input{filename.tex}"
```

- `cache`/`nocache`: This is an advanced option which controls whether the CSV file is generated or not.
- `debug`: This option enables verbose debug output from ul*q*da.
- `MiKTeX`: This determines whether MiKTeX is supported or not. MiKTeX is a version of TeX that runs on Microsoft Windows platforms.
- `shell`/`noshell`: These options control whether an attempt will be made to process the coding output file via spawning the `ulqda.pl` script directly, or whether it needs to be run explicitly by the user. `shell` is the default, but it requires `--shell-escape` (TeX Live) or `--enable-write18` (MiKTeX) as a command line argument to `latex` to enable it.
- `counts`: This option determines whether code output will include occurence counts or not. The default is to not output the counts.

In summary, to ensure correct section/page numbers, set the `active` and leave the cache setting at its default (`nocache`) for each run. It is possible to tweak both of these to reduce the processing time, being aware of potential side-effects!

6.1.1 Advanced Options Usage

The use of the `active` and `cache` options are primarily to speed up the process of performing QDA code extraction through the LaTeX typesetting flow. Some care is needed with their use, and it makes sense to select `active,nocache` as default options until comfortable with the typesetting flow for a particular document – otherwise section numbering/page numbering in the generated CSV file may be incorrect.

If this isn't a concern (i.e. traceability and per-section filtering for graph visualisation isn't required), then setting `active,cache` on one pass through LaTeX will give best performance.

If page numbers / section numbers are required, then the appropriate use of these options will need to be made as required by the specific LaTeX flow being used – i.e. enable as appropriate. It will need to run like this at least 3 times (once to generate the CSV file, once to generate the .Dot output, and once to import any generated figures or tables. I suggest integrating something like the following for the last 3 LaTeX passes through the source:

```
$ pdflatex --shell-escape "\PassOptionsToPackage{active,nocache}{ulqda}
     \input{filename.tex}"
$ pdflatex --shell-escape "\PassOptionsToPackage{active,cache}{ulqda}
     \input{filename.tex}"
$ pdflatex --shell-escape "\PassOptionsToPackage{active,cache}{ulqda}
```

```
\input{filename.tex}"
```

6.2 Macros

`\ulqdaCode` `\ulqdaCode` is used to assign a code a particular sentence or passage of text. Coding is a form of data condensing, where the words of the passage are compacted and distilled into as few succinct words as possible with the aim of capturing the essence or theme of the passage.

`\ulqdaCode` takes a list of codes as a first parameter, and the raw text as its second. It invokes `\ulqdaHighlight` in order to format the passage for typesetting purposes, and outputs the code, page number, section number, and raw text to the CSV file - one line per code.

The list of codes is a comma separated list; code hierarchies and connections can be expressed by chaining codes together using the exclamation mark - for example, ‘`geographical!urgency`’ would indicate a relationship between the code ‘`geographical`’ and the code ‘`urgency`’.
USAGE: `\ulqdaCode{code1,code2,code3}{Common Text}`

`\ulqdaHighlight` `\ulqdaHighlight` is used to format coded text for typesetting purposes. By default, it highlights the coded text in a light blue color, and it also lists the associated codes in the margin. It can be redefined to whatever formatting codes the package user requires.
USAGE: `\ulqdaHighlight{code1,code2,code3}{Common Text}`

`\ulqdaGraph` `\ulqdaGraph` is a macro which invokes processing of the generated CSV file to allow the visualisation of a coded ontology as a GraphViz diagram. It take two arguments:

- graph type - this can be either ‘`flat`’ which is an unstructured graph (see figure 1(a)), or ‘`net`’ (see figure 1(b)), where the ontology relationships are shown as a connected graph;
- dot2texi options - this is a list of options that would typically be used in a dot2tex environment. Listing these is outside the scope of this document, but the following set of options is used in the diagrams in this document: `neato,mathmode,options={--graphstyle "scale=0.5,transform shape"`.

USAGE: `\ulqdaGraph{`*graph type*`}{`*dot2texi options*`}`

`\ulqdaTable` `\ulqdaTable` is a macro which invokes processing of the generated CSV file to create a LaTeX table (see table 1).
USAGE: `\ulqdaTable`

`\ulqdaCloud` `\ulqdaCloud` is a macro which invokes processing of the generated CSV file to create a LaTeX cloud (see table 2).

Usage: `\ulqdaCloud`

`\ulqdaSetSectFilter` `\ulqdaSetSectFilter` establishes a filter for the next `\ulqdaGraph` or `\ulqdaTable` macro. If interviews are logically structured in a document with each in its own section (or sub-section etc.) then this command can be used to establish a filter restricting the graphing or table generation to a single interview.
Usage: `\ulqdaSetSectFilter{`*section label*`}`

`\ulqdaClearSectFilter` `\ulqdaClearSectFilter` clears a section filter establihed by `\ulqdaSetSectFilter` so that a subsequent `\ulqdaGraph` or a `\ulqdaTable` macro will process all sections from the CSV file.
Usage: `\ulqdaClearSectFilter`

6.3 Example

What follows is an interview excerpt that has been taken through the entire flow, i.e.:

- coded;
- typeset; and
- visualized as a tabular list of codes and also as graphs.

6.3.1 Coding Example

First, here is the raw LaTeX source:

```
\textbf{IG:} Do you think the social aspect of face
to face is important for the project?  ...

\textbf{Interviewee~XYZ:} ...  A cup of coffee is really
important because then what happens is that you get a
real perspective.  My general experience of having a
functional group in one site, while I was in the other
one, working for me and using video conferencing,
\ulqdaCode{geographical!urgency, geographical!face-eo-face}{if you
really wanted to get things done you had to jump on
a plane and fly over, there was nothing that could make
up for sitting in a room with people to both get across
the urgency and to ensure that communication among
the team took place to address any of the issues...}
```

6.3.2 Typeset Example

Next, we will see what happens when this source is typeset. The mainbody text is itself highlighted so that it stands out from surrounding text, and the codes are present in the margin.

> **IG:** Do you think the social aspect of face to face is important for the project? …
> **Interviewee XYZ:** … A cup of coffee is really important because then what happens is that you get a real perspective. My general experience of having a functional group in one site, while I was in the other one, working for me and using video conferencing, if you really wanted to get things done you had to jump on a plane and fly over, there was nothing that could make up for sitting in a room with people to both get across the urgency and to ensure that communication among the team took place to address any of the issues…
>
> geographical!urgency, geographical!face-to-face

6.3.3 CSV Cache File

The following shows an example of the comma-separated value cache file generated for the coded text above. The first line of this file is a header and is ignored in processing by the `ulqda.pl` script.

```
Page Number, Section, Code, Text
2, 0, geographical!urgency, "if you really wanted to get things done
you had to jump on a plane and fly over, there was nothing that could
make up for sitting in a room with people to both get across the
urgency and to ensure that communication among the team took place
to address any of the issues..."
2, 0, geographical!face-to-face, "if you really wanted to get things
done you had to jump on a plane and fly over, there was nothing that
could make up for sitting in a room with people to both get across the
urgency and to ensure that communication among the team took place to
address any of the issues..."
```

6.3.4 Visualisation as a Table

Table 1 illustrates the output from `\ulqdaTable`.

Table 1: List of QDA Codes

geographical	urgency	face-to-face

6.3.5 Visualisation as a Cloud

Table 1 illustrates the output from `\ulqdaCloud`.

Table 2: List of QDA Codes

FPGA HW HW attitude to risk HW bias HW fear of risk HW focus HW is fixed HW reluctance to design change HW vs SW IM IP SQA SW focus SW influence on System Arch SW is changeable SW models SW workarounds adherence to process adverse aggressive schedules agile methods algorithmic software ambition approach to test bring in software expertise early business model changeability of SW changing market requirements co-location communication communications difficulties competitive analysis competitiveness complexity complexity in SW control code complexity risk confidence constraints consumer electronics control software cost of changing HW cost of test cost of wrong HW cost-benefit of process cross-functional culture design modelling dimensioning HW early prototype engineers over-simplify experience fabless face to face false perception fluid specifications focus freedom to innovate friction geographical geographical mitigation geographical more impact than technical greatest impact gsd gsd mitigation hardware implementation importance of cross-functional skills importance of face to face inadequate testing incidental is most important incidental knowledge informal chats information sharing internalising keep SW model in sync with HW lack of mixed design skills learning curve limitations of SW models management market analysis market change market risk market window methodology mindset gap mitigation moving SW into HW moving schedule moving software into hardware multi-disciplinary new platform opportunity for HW change opportunity to change organisation overconfidence perception of other discipline process product specification project inception realtime missing from SW model reluctance to change requires hardware focus resource requirements resource usage analysis risk risk mitigation schedule schedule impact social social familiarity social risk social tools software specialisation specifying HW resources system resources system understanding tapeout set by hardware team building technical technical determinism technical language barrier techno-geographical split telecoms test code sharing testing HW without final SW time to market tool problems tools underestimate learning curve unedited validation value in test bench value of SW models value of reference platforms verification verification risk verify SW without HW visibility weight of HW risk workaround

---

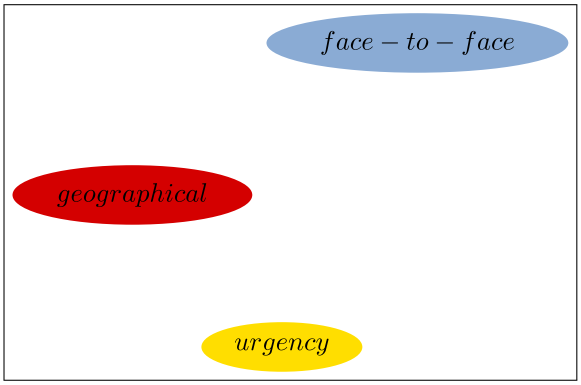
6.3.6 Visualisation as Graphs

Figure 1 shows the visualisation output possible from ul*q*da:

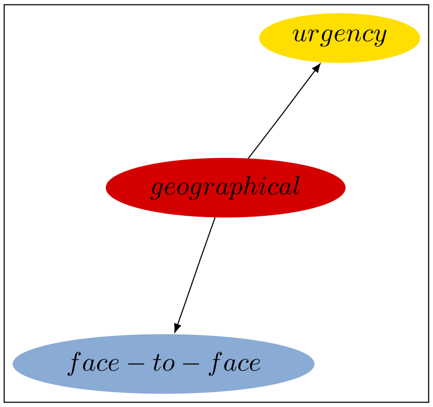
- figure 1(a) shows the image created using

```
\ulqdaGraph{flat}{neato,mathmode,
    options={--graphstyle "scale=0.5,transform shape"}}
```

- figure 1(b) shows the image created using `\ulqdaGraph{net}{neato,mathmode,`
  `options={--graphstyle "scale=0.5,transform shape"}}`.



(a) Flat Graph



(b) Hierarchical Graph with Connections

Figure 1: Visualisation through GraphViz

Figure 2 shows a more complex visualisation generated from a more comprehensive set of coding.

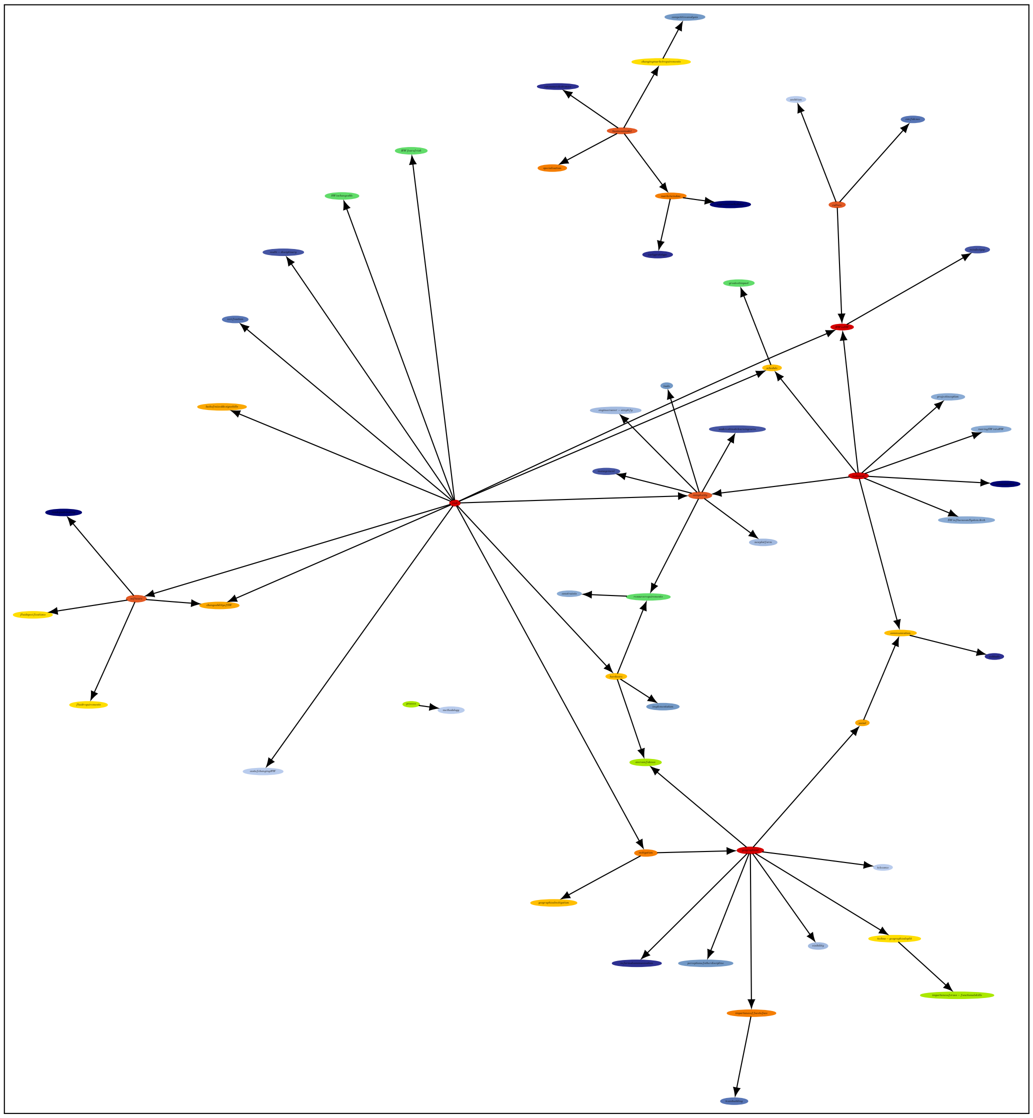


Figure 2: Complex Visualisation of Axial Coding Ontology

7 Implementation

7.1 Dependencies

We start be ensuring that the required packages are loaded when this file is loaded as a package by LaTeX.

```
⟨*package⟩
\RequirePackage{multicol}
\RequirePackage{tikz}
% \iffalse
%%  dot2texi.sty in CTAN doesn't support the cache option yet
%%  The SVN version does.
% \RequirePackage[cache]{dot2texi}
% \fi
\RequirePackage{dot2texi}
\usetikzlibrary{backgrounds,shapes,arrows,positioning}

⟨/package⟩
```

7.2 Highlighting Style

We next setup some default highlighting formatting defines. The user is free to change the highlighting formatting through redefining `\ulqdaHighlight`.

```
⟨*package⟩

\definecolor[named]{UlQda@lightblue}{rgb}{0.80,0.85,1}
\RequirePackage{soul}
\sethlcolor{UlQda@lightblue}

⟨/package⟩
```

7.3 Package Options

```
⟨*package⟩
\newif\ifUlQda@debug \UlQda@debugfalse
\newif\ifUlQda@cache \UlQda@cachefalse
\newif\ifUlQda@cachepresent \UlQda@cachepresentfalse
\newif\ifUlQda@shellescape \UlQda@shellescapetrue
\newif\ifUlQda@MiKTeX \UlQda@MiKTeXfalse
\newif\ifUlQda@active \UlQda@activefalse
\newif\ifUlQda@counts \UlQda@countsfalse

\DeclareOption{active}{\UlQda@activetrue}
\DeclareOption{debug}{\UlQda@debugtrue}
\DeclareOption{cache}{\UlQda@cachetrue}
\DeclareOption{nocache}{\UlQda@cachefalse}
\DeclareOption{shell}{\UlQda@shellescapetrue}
\DeclareOption{noshell}{\UlQda@shellescapefalse}
\DeclareOption{MiKTeX}{\global\UlQda@MiKTeXtrue}
```

```
\DeclareOption{counts}{\global\UlQda@countstrue}

\DeclareOption*{%
  \PackageWarning{ulqda}{Unknown option `\CurrentOption'}%
}

\ExecuteOptions{shell}
\ProcessOptions\relax

\ifUlQda@counts
   \def\UlQda@counts{--number }
\else
   \def\UlQda@counts{ }
\fi

⟨/package⟩
```

7.4 Testing the Shell Escape Mechanism

Needs to work on both Unix-type platforms and on MiKTEX on Microsoft Windows.

```
⟨*package⟩
%% test if shell escape really works
\ifUlQda@shellescape
  \def\tmpfile{/tmp/shellEscapeTest-\the\year\the\month\the\day-\the\time}
  \immediate\write18{\ifUlQda@MiKTeX rem >"\tmpfile" \else touch \tmpfile \fi}
  \IfFileExists{\tmpfile}{
    \UlQda@shellescapetrue
    \immediate\write18{\ifUlQda@MiKTeX del "\tmpfile" \else rm -f \tmpfile \fi}
  }{\UlQda@shellescapefalse}
\fi

\ifUlQda@shellescape
  \ifUlQda@debug
    \PackageInfo{ulqda}{TeX Shell escape enabled.}
  \fi
\else
  \PackageWarningNoLine{ulqda} {TeX Shell escape not enabled.\MessageBreak%
     Manually process the CSV output with ulqda.pl}
\fi

⟨/package⟩
```

7.5 Active Macro Implementation

`\ulqdaHighlight` The most basic macro is a style macro - to format the typeset text, indicating that it has been coded, and also to place the codes themselves in the margin.

```
⟨*package⟩
\newcommand{\ulqdaHighlight}[2]{%
```

```
    \hl{\protect\ul{#2}}%
    \marginpar%
    {\raggedright\hbadness=10000\tiny\it%
      \hl{#1}
  \par}%
  %\par%
  }

⟨/package⟩
```

We'll also create `\ulQda`, a vanity macro to typeset the ulqda package name, in the TEX tradition.

`\ulQda`

```
⟨*package⟩
\newcommand{\ulQda}{\textsf{ul\kern -.075em\lower .3ex\hbox {\protect\emph{q}}da}}

⟨/package⟩
```

Next, we need to determine if the package is intended to be active for this LATEX processing run or not. This is essentially a big switch around the majority of the package definitions.

```
⟨*package⟩
\ifUlQda@active
⟨/package⟩
```

`\ulqdaCode` We now create a macro, `\ulqdaCode` to perform the actual coding of the raw text. This macro, when invoked, will invoke the highlighting macro `\ulqdaHighlight` and also conditionally invoke the package private macro `\UlQda@ListIt` to output coded text to a comma separate values (.csv) cache file.

This is hooked (presently) to `\begin{document}`, and contains some conditional code to decide if caching is enabled, and if so, if the cache is present or not.

```
⟨*package⟩
%
%
  \AtBeginDocument{%
    \typeout{ulqda: Loaded - 2009/06/11 v1.1 Qualitative Data Analysis package}
```

¡/package¿

If caching is enabled, the .csv file will only be generated if necessary. This is because the .csv generation can be quick slow - particularly when dealing with a number of large portions of text, each having multiple codes.

```
%
⟨*package⟩
    \ifUlQda@cache
      \IfFileExists{\jobname.csv} %
      {
        \ifUlQda@debug
```

```
          \typeout{ulqda: QDA cache file \jobname.csv found}
        \fi
        \UlQda@cachepresenttrue
      }
      {
        \ifUlQda@debug
          \typeout{ulqda: QDA cache file \jobname.csv not found}
        \fi
        \UlQda@cachepresentfalse
      }
    \else
      \ifUlQda@debug
        \typeout{ulqda: caching disabled}
      \fi
      \UlQda@cachepresentfalse
    \fi
⟨/package⟩
```

Without caching enabled, the .csv file will be generated every run.

If a cache file is detected and shell escape is enabled, the .csv cache will be processed on demand: by `\ulqdaGraph` to generate GraphViz .dot file outputs, by `\ulqdaCloud` to generate tag cloud style maps, and by `\ulqdaTable` to generate a multicolumn list of codes.

In this case, the `\ulqdaCode` macro will not cause the cache file to update, but instead will only perform a typesetting function.

```
⟨*package⟩

  % Code macro
    \ifUlQda@cachepresent
      \newcommand{\ulqdaCode}[2]{\ulqdaHighlight{#1}{#2}}
⟨/package⟩
```

Otherwise, any occurrence of the `\ulqdaCode` macro will update the cache file for the run.

```
⟨*package⟩
    \else
      \ifUlQda@debug
        \typeout{ulqda: Creating QDA cache file \jobname.csv} %
      \fi
      \newwrite\ulqdaCodeFile %
      \immediate\openout\ulqdaCodeFile=\jobname.csv %
      \immediate\write\ulqdaCodeFile{Page Number, Section, Code, Text} %

⟨/package⟩
```

The following macro outputs the coding to the code file.

```
⟨*package⟩
      \def\UlQda@ListIt#1[#2,{%
        \ifUlQda@debug %
```

```
         \typeout{ulqda: Coding "#2" as "#1" on page \thepage, section \thesection}
        \fi %
        \immediate\write\ulqdaCodeFile{\thepage, \thesection, #2, "#1"}
⟨/package⟩
```

It also causes the code to be added to the index for the document, which is useful.

```
⟨*package⟩
        \index{#2} %
        \@ifnextchar]%             Look ahead one token.
          {\eatthesquarebracket}%  End of list.
          {\UlQda@ListIt{#1}[]%     Process rest of list.
      }
      \def\eatthesquarebracket]{}  % Gobble the square bracket.
      %
      % Coding macro
      \newcommand{\ulqdaCode}[2]{\ulqdaHighlight{#1}{#2}\UlQda@ListIt{#2}[#1,]} %
    \fi
  } % end of \AtBeginDocument

⟨/package⟩
```

`\ulqdaSetSectFilter`

`\ulqdaSetSectFilter` enables filtering of CSV processing by section label.

```
⟨*package⟩
\newcommand{\UlQda@FirstOfTwo}[1]{
  \ifx#1\UlQda@MyUndefinedMacro
    ?\typeout{ulqda: undefined reference, please re-run}
  \else
    \expandafter\@firstoftwo#1
  \fi}
\newcommand{\UlQda@RefToSectNum}[1]{
  \expandafter \ifx\csname r@#1\endcsname\relax
    ?\typeout{ulqda: undefined reference, please re-run}
  \else
    \expandafter\UlQda@FirstOfTwo\csname r@#1\endcsname
  \fi}
⟨/package⟩
```

Now we start the actual filtering work. First, we delete any old files from previous builds. Next, we create a macro which will be used to pass a command line argument selecting the appropriate filtering to `ulqda.pl`.

```
⟨*package⟩
  \def\UlQda@filter{}
  \newcommand{\ulqdaSetSectFilter}[1]{
      \ifUlQda@shellescape
        \immediate\write18{\ifUlQda@MiKTeX del \else rm -f -- \fi \jobname_net.dot}
        \immediate\write18{\ifUlQda@MiKTeX del \else rm -f -- \fi \jobname_flat.dot}
        \immediate\write18{\ifUlQda@MiKTeX del \else rm -f -- \fi \jobname_table.tex}
      \fi
      \def\UlQda@filter{--filter \UlQda@RefToSectNum{#1}}
  }
⟨/package⟩
```

`\ulqdaClearSectFilter` We also need to be able to clear any previously configured filter, and this is what the following macro does for us.

```
⟨*package⟩
  \newcommand{\ulqdaClearSectFilter}{\def\UlQda@filter{}}
⟨/package⟩
```

`\ulqdaGraph` It is typical to want to present your coded data visually in a number of different ways, perhaps focusing on a particular sub-theme if the entire ontology is too cumbersome. However, I have provided a sample macro, `\ulqdaGraph`, which will support the generation of an overall ontology graph through the use of `dot2texi.sty`.

`\ulqdaGraph` uses the power of `\csname` to expand to either `\UlQda@GraphNet` or `\UlQda@GraphFlat`, depending on its first argument.

```
⟨*package⟩
  \newcommand{\ulqdaGraph}[2]{\expandafter\csname UlQda@Graph#1\endcsname{#2}}
  \newcommand\UlQda@Graphflat[1]{\UlQda@GraphFlat{#1}}
  \newcommand\UlQda@Graphnet[1]{\UlQda@GraphNet{#1}}
  \newcommand{\UlQda@GraphVizFileName}{}
  \newsavebox{\UlQda@GraphSaveBox}
  \newcommand{\UlQda@GraphNet}[1]{%
    \renewcommand{\UlQda@GraphVizFileName}{\jobname_net.dot}%
⟨/package⟩
```

If a cache file is detected and shell escape is enabled, the .csv cache will be processed on demand by `\UlQda@GraphNet` to generate GraphViz .dot file output.

```
⟨*package⟩
    \ifUlQda@cachepresent
      \ifUlQda@shellescape
        \ifUlQda@debug
          \typeout{ulqda: Converting .csv to hierarchical GraphViz dot file}
        \fi
        \immediate\write18{ulqda.pl --graphnet \UlQda@filter \UlQda@counts
                              -- \jobname.csv \jobname_net.dot}
      \fi
    \fi

    \UlQda@DoGraph{#1}%
  }
  \newcommand{\UlQda@GraphFlat}[1]{%
    \renewcommand{\UlQda@GraphVizFileName}{\jobname_flat.dot}%
⟨/package⟩
```

If a cache file is detected and shell escape is enabled, the .csv cache will be processed on demand by `\UlQda@GraphFlat` to generate GraphViz .dot file output.

```
⟨*package⟩
    \ifUlQda@cachepresent
```

```
   \ifUlQda@shellescape
    \ifUlQda@debug
     \typeout{ulqda: Converting .csv to flat GraphViz dot file}
    \fi
    \immediate\write18{ulqda.pl --graphflat \UlQda@filter \UlQda@counts
              -- \jobname.csv \jobname_flat.dot}
   \fi
  \fi

  \UlQda@DoGraph{#1}%
 }

⟨/package⟩
```

The following package internal macro, `\UlQda@DoGraph`, actually enacts the graph generation.

```
⟨*package⟩
 \newcommand{\UlQda@DoGraph}[1]{
  \IfFileExists{\UlQda@GraphVizFileName}{
   \ifUlQda@shellescape
    \begin{lrbox}{\UlQda@GraphSaveBox}
⟨/package⟩
```

The next few lines are a hack to enable `dot2texi.sty` to work with an external .dot file[1].

```
⟨*package⟩
     \stepcounter{dtt@fignum}
     \setkeys{dtt}{#1}
     \immediate\write18{cp "\UlQda@GraphVizFileName" "\dtt@figname.dot"}
     \dottotexgraphicsinclude
⟨/package⟩
```

Now we finish the `\ulqdaGraph` command.

```
⟨*package⟩
    \end{lrbox}
    \framebox{\usebox{\UlQda@GraphSaveBox}} \par
   \else
    \typeout{ulqda: shell escape not enabled}
    \typeout{ulqda: unable to process \UlQda@GraphVizFileName}
   \fi
  }
 }
⟨/package⟩
```

We now create a `\ulqdaTable` macro – a command to process the table of codes. It doesn't do terribly much, but it is there because it is useful and conceptually consistent with the graph macros.

---

[1] I have suggested this to Kjell Magne Fauskes, the `dot2texi.sty` author, and he intends to include such a feature natively in a future version.

`\ulqdaTable`

```
⟨*package⟩
  \newcommand{\ulqdaTable}{
    \IfFileExists{\jobname_table.tex}{
      \input{\jobname_table.tex}
    }{
⟨/package⟩
%   \end{macrocode}
% If a cache file is detected and shell escape is enabled, the~.csv cache
% will be processed on demand by |\ulqdaTable| to generate
% a multicolumn list of codes.
%   \begin{macrocode}
⟨*package⟩
      \ifUlQda@cachepresent
        \ifUlQda@shellescape
          \ifUlQda@debug
            \typeout{ulqda: Converting .csv to TeX table}
          \fi
          \immediate\write18{ulqda.pl --list \UlQda@filter \UlQda@counts
                                -- \jobname.csv \jobname_table.tex}
        \fi
      \fi
      \IfFileExists{\jobname_table.tex}{
        \input{\jobname_table.tex}
      }
    }
  }
⟨/package⟩
```

Next, we create a `\ulqdaCloud` macro – a command to process the table of codes and create a tag cloud style visualisation.

`\ulqdaCloud`

```
⟨*package⟩
  \newcommand{\ulqdaCloud}{
    \IfFileExists{\jobname_cloud.tex}{
      \input{\jobname_cloud.tex}
    }{
⟨/package⟩
%   \end{macrocode}
% If a cache file is detected and shell escape is enabled, the~.csv cache
% will be processed on demand by |\ulqdaCloud| to generate the cloud.
%   \begin{macrocode}
⟨*package⟩
      \ifUlQda@cachepresent
        \ifUlQda@shellescape
          \ifUlQda@debug
            \typeout{ulqda: Converting .csv to TeX cloud}
          \fi
          \immediate\write18{ulqda.pl --cloud \UlQda@filter \UlQda@counts
```

```
                               -- \jobname.csv \jobname_cloud.tex}
        \fi
      \fi
      \IfFileExists{\jobname_cloud.tex}{
        \input{\jobname_cloud.tex}
      }
    }
  }
⟨/package⟩
```

7.6 Inactive Macro Stubs

If the package is not intended to be active, we need to create stub definitions for the macros that the package provides, so that document runs where the package is not active will succeed.

```
⟨*package⟩
\else % UlQda@activefalse
⟨/package⟩
```

`\ulqdaTable`

```
⟨*package⟩
  \newcommand{\ulqdaTable}{}
⟨/package⟩
```

`\ulqdaCloud`

```
⟨*package⟩
  \newcommand{\ulqdaCloud}{}
⟨/package⟩
```

`\ulqdaGraph`

```
⟨*package⟩
  \newcommand{\ulqdaGraph}[2]{}
⟨/package⟩
```

`\ulqdaCode`

```
⟨*package⟩
  \newcommand{\ulqdaCode}[2]{#2}
⟨/package⟩
```

`\ulqdaSetSectFilter`

```
⟨*package⟩
  \newcommand{\ulqdaSetSectFilter}[1]{}
⟨/package⟩
```

`\ulqdaSetSectFilter`

```
⟨*package⟩
  \newcommand{\ulqdaClearSectFilter}{}
⟨/package⟩
```

And finally close the conditional switch on whether active or not.

```
⟨*package⟩
\fi
⟨/package⟩
```

References

[1] M. B. Miles and A. M. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook*. 2455 Teller Road, Thousand Oaks, California 91320, USA: Sage Publications, Inc., 1994. ISBN-10: 0-8039-5540-5.

[2] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. 200 Saw Mill River Road, Hawthorne, New York 10532, USA: Aldine De Gruyter, 1967. ISBN-10: 0-202-30260-1.

[3] A. Strauss and J. Corbin, eds., *Grounded Theory in Practice*. 2455 Teller Road, Thousand Oaks, California 91320, USA: Sage Publications, 1997. ISBN-10: 0-7619-0748-3.

[4] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software - Practice and Experience*, vol. 30, pp. 1203–1233, 1999.

[5] K. M. Fauske, "dot2text – A GraphViz to LaTeX converter," 2006. available: `http://www.fauskes.net/code/dot2tex/` [accessed 2009-03-02 17h31.

[6] B. Appleton, "The LoRD Principle – Locality breeds Maintainability," *Portland Patterns Repository wiki*, 1997. available: `http://c2.com/cgi/wiki?LocalityOfReferenceDocumentation` [accessed 2009-05-14 10h03].

[7] J. Hefferon, "LaTeX goes with the flow," *The PracTeX Journal*, no. 1, 2008. available `http://www.tug.org/pracjourn/2008-1/hefferon/` [accessed 2009-03-02 17h51].

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

`\@firstoftwo` 161
`\@ifnextchar` 144

A

`\AtBeginDocument` 94, 153

C

`\CurrentOption` 39

D

`\DeclareOption` 29–36, 38
`\definecolor` 15
`\dottotexgraphicsinclude` 234
`\dtt@figname` 233

E

`\eatthesquarebracket` 145, 148
`\end` 237, 252, 279
`\ExecuteOptions` 42

F

`\framebox` 238

I

`\IfFileExists` 57, 99, 226, 248, 267, 275, 293
`\ifUlQda@active` 26, 89
`\ifUlQda@cache` 22, 98
`\ifUlQda@cachepresent` 23, 122, 194, 210, 258, 284
`\ifUlQda@counts` 27, 45
`\ifUlQda@debug` 21, 64, 101, 107, 113, 127, 137, 196, 212, 260, 286
`\ifUlQda@MiKTeX` 25, 56, 59, 174–176
`\ifUlQda@shellescape` 24, 54, 63, 173, 195, 211, 227, 259, 285

M

`\marginpar` 76
`\MessageBreak` 68

N

`\newcommand` 74, 85, 123, 151, 157, 163, 172, 182, 185–188, 190, 206, 225, 247, 274, 303, 306, 309, 312, 315, 318
`\newif` 21–27
`\newsavebox` 189
`\newwrite` 130

P

`\PackageInfo` 65
`\PackageWarning` 39
`\PackageWarningNoLine` 68
`\ProcessOptions` 43

R

`\raggedright` 77
`\renewcommand` 191, 207

T

`\textsf` 85

U

`\ulQda` 84
`\UlQda@activefalse` 26
`\UlQda@activetrue` 29
`\UlQda@cachefalse` 22, 32
`\UlQda@cachepresentfalse` 23, 110, 116
`\UlQda@cachepresenttrue` 104
`\UlQda@cachetrue` 31
`\UlQda@counts` 46, 48, 199, 215, 263, 289
`\UlQda@countsfalse` 27
`\UlQda@countstrue` 36
`\UlQda@debugfalse` 21
`\UlQda@debugtrue` 30
`\UlQda@DoGraph` 204, 220, 225
`\UlQda@filter` 171, 178, 182, 199, 215, 263, 289
`\UlQda@FirstOfTwo` 157, 167
`\UlQda@GraphFlat` 186, 206
`\UlQda@Graphflat` 186
`\UlQda@GraphNet` 187, 190
`\UlQda@Graphnet` 187
`\UlQda@GraphSaveBox` 189, 228, 238
`\UlQda@GraphVizFileName` 188, 191, 207, 226, 233, 241
`\UlQda@ListIt` 136, 146, 151
`\UlQda@MiKTeXfalse` 25
`\UlQda@MiKTeXtrue` 35
`\UlQda@MyUndefinedMacro` 158
`\UlQda@RefToSectNum` 163, 178
`\UlQda@shellescapefalse` 34, 60
`\UlQda@shellescapetrue` 24, 33, 58
`\ulqdaClearSectFilter` 181, 318
`\ulqdaCloud` 273, 305
`\ulqdaCode` 91, 311
`\ulqdaCodeFile` 130–132, 140
`\ulqdaGraph` 184, 308
`\ulqdaHighlight` 73, 123, 151
`\ulqdaSetSectFilter` 156, 314, 317
`\ulqdaTable` 246, 302
`\usebox` 238
`\usetikzlibrary` 10

Change History

v1.0
    General: Initial Version. 1