User's Guide to the CAMEL Citator*

Frank G. Bennett, Jr.†

July 12, 1995

Abstract

This is the core documentation for CAMEL, a LaTeX macro package that drastically simplifies the typesetting of citations and bibliographies in a wide variety of styles and formats. CAMEL is similar in look and feel to standard BibTeX, but introduces a number of long-desired enhancements. When the document is printed, CAMEL allows the generation of subdivided bibliographies, separate bibliographies for each subdivision of a larger work, in-footnote cross-referenced citations, bibliographies indexed to the pages where citations to the relevant work occur, and much else. This is the only fully supported bibliography package for LaTeX that provides full support for legal citations. The package also provides genuinely helpful error messages and an extremely simple syntax for adding details like page numbers and the like to citation tags. CAMEL provides the logical engine and skeleton macros that make these things possible; a separate style package is required to actually typeset documents. For an example of a working style package, see `law.dtx` in the `camel` subdirectory on CTAN.

Contents

*This file is version number 1.0m. The documentation was last revised on 95/06/24.

†The text and source code contained in this file are © 1992–1995 by Frank G. Bennett, Jr. This file can be freely distributed; the sole condition of use is that acknowledgement to the author be made in any published work produced using CAMEL. The author wishes to thank Pedro Aphalo, Steve Bellovin, George Greenwade and Boris Pevzner and David Rhead for their helpful comments on versions of LexiTeX, the predecessor to CAMEL. In addition, David Rhead provided a mass of information on bibliographic conventions, which was used in shaping the options supported by this package. Gratitude is also owed to Marcia Elliot and Yoshiko Makino, LL.B. finalists in the Law Department of the School of Oriental and African Studies in 1993 and 1994 respectively, who prepared extended essays using early releases of LexiTeX.

"Now comes the fun part."
—Donald Knuth[1].

Preface to release 1.0

Welcome to CAMEL, a comprehensive bibliography manager developed as a prototype citation engine for the LaTeX3 project. The CAMEL engine offers a highly flexible yet straightforward syntax for entering citation keys into a LaTeX document. The CAMEL engine is designed to support a wide range of citation styles. Documents prepared for CAMEL can be converted into virtually any supported bibliography or citation style, usually by editing only a few toggles at the top of the document.[2]

Within the LaTeX3 project, an extensive survey of bibliographic styles was carried out by David Rhead. Pedro Aphalo, Task Coordinator for bibliographic support on the LaTeX3 Project, first proposed adopting a unified syntax for all bibliographic styles. CAMEL itself is the brainchild of Frank Bennett, and is based upon his earlier work on the LexiTeX style engine for legal materials. This package completely supercedes LexiTeX, and that earlier style package will no longer be supported. The new package should be referred to as "the CAMEL citator".[3]

The core CAMEL package provides hooks for the development of style options that satisfy all bibliographic requirements that have come to the attention of the LaTeX3 bibliography team. The foundation has been laid, but much work remains to be done in drafting CAMEL modules to meet all known bibliographic requirements in all fields (and in all languages) for which LaTeX is used. This work will involve the drafting or modification of `.bst` files in the reverse polish notation used by BibTeX, and of CAMEL style modules written in TeX. If you want to contribute to a project that will dramatically enhance the utility of LaTeX as an author's productivity tool, get involved! The developers' email addresses are as follows:

- Frank Bennett: `fb@soas.ac.uk`.
- Pedro Aphalo: `aphalo@cc.joensuu.fi`.

[1] D. KNUTH, THE TEX BOOK 176 (1990).

[2] This is potentially exciting (he said), but as of this writing only one module — `law.dtx`, the legal citation module used for beta-testing the CAMEL engine — exists. Some followers of CAMEL development have expressed serious interest in developing other modules, however, so a claim to broadening support for this package is legitimate.

[3] Why CAMEL? I can offer several reasons, users may take their pick. The style operates by gathering citation details that it carries with it through the desert of plain text, until the bibliography engine reaches its oasis at document's end. Alternatively, in his daily work the author uses the multi-lingual enchancement of Emacs, which is called "Mule". It's nice for one critter to have the company of another in the stable. A third arbitrary explanation is that the Japanese word for "camel" is *rakuda*. These syllables, with a different intonation, mean "dead easy". Since the design goal of CAMEL is to make bibliography management as easy, as fool-proof and as robust as possible, this cross-lingual pun seems fitting.

- David Rhead: `David_Rhead@vme.ccc.nottingham.ac.uk`.

A programmer's reference guide for drafting CAMEL styles is included in this manual. Don't delay—start coding today.

1 Introduction

This guide explains the use of the CAMEL Citator. It covers all commands available in the style. This should be the only user manual that you need to prepare a document for use with any CAMEL bibliography style. The package provides eight commands, listed in the table at the bottom of this page. The opening commands may be combined in a variety of ways, but a typical document heading might look like this:

```
\documentclass{article}
\usepackage{camel}
\citationsubject{english}{Materials in English}
\citationsubject{german}{Materials in German}
\begin{document}
\citationstyle{law}
\citationdata{gorp,slush,fodder}
\forcefootnotes
```

You should be careful to decide on the subject classification, if any, that you wish to apply to your citations, before you begin your writing. With that in hand, you can begin entering `\source` commands into the document, adjusting the style parameters at print time. A detailed description of command usage follows. Throughout this guide, it is assumed that the reader has a general familiarity with BIBTEX and its use in preparing bibliographies with standard-issue LATEX. If you need to learn more about BIBTEX, a number of excellent sources of information are available.[4] The treatment given to the programming language tends to be quite brief, but you should not be put off by this; the language is, in fact, quite simple to use, and the CAMEL programming library makes things even easier.

[4] L. LAMPORT, LATEX: A DOCUMENT PREPARATION SYSTEM (1994); M. GOOSENS, F. MITTELBACH, AND A. SAMARIN, THE LATEX COMPANION (1994); Oren Patashnik, BIBTEXing (CTAN document, 1988) *and* Oren Patashnik, Designing BIBTEX Styles (CTAN document, 1988).

CAMEL **commands**		
`\citationsubject[`⟨*options*⟩`]{`⟨*nickname*⟩`}{`⟨*heading text*⟩`}`	optional	preamble
`\citationstyle{`⟨*style package name*⟩`}`	required	doc top
`\citationdata{`⟨*bib1,bib2...*⟩`}`	alternative	doc top
`\bibliographymanager{`⟨*product name*⟩`}`	alternative	doc top
`\forcefootnotes`	optional	doc top
`\newinterword{`⟨*nickname*⟩`}{`⟨*text*⟩`}`	optional	doc top
`\printbibliography{`⟨*nickname*⟩`}`	optional	anywhere
`\source[`⟨*options*⟩`]{`⟨*nickname*⟩`}[`⟨*page numbers*⟩`]`	n/a	anywhere

1.1 Commands

`\citationstyle` The `\citationstyle` command is used to tell both BibTeX and the Camel engine how your citations should be formatted; its argument is the name of a Camel module (contained in a `.cst`, a `.cit` and a `.bst` file) such as '`law`'. In this respect, it is similar to a "document class", but for citations instead of whole documents. This declaration may be make more than once.[5]

`\citationdata` This command is used to indicate the `.bib` files from which citation entries should be drawn. As in the `\bibliography` command of standard LaTeX, the `\citationdata` command takes one option, which is a comma-separated list of bibliographies for use in the documents. Each refers to a `.bib` file, each of which should be in the path of your local BibTeX.

`\bibliographymanager` While we hope that you find BibTeX the most satisfactory means of formatting your citations, you might prefer to use some other software product for the separate task of managing a large citation database. Camel has been designed so that it can be used with a number of such utilities. If you are using an external bibiliography manager, you should declare the name of the package at the top of the document using the `\bibliographymanager` command. Supported managers, and their corresponding nicknames for this command, are:

- Reference Manager (`referencemanager`)
- Papyrus (`papyrus`)
- ProCite (`procite`)
- EndNote (`endnote`)

The details on how to join external bibliography managers with Camel are below at page 9.

If you use the `\bibliographymanager` command, you should not use the `\citationdata` command. This is not felt to be a serious limitation; if you use a bibliography manager, you might as well put *all* of your citations into it anyway, and use it as the sole source of citations in your documents.

`\citationsubject` You may wish to produce a bibliography classified into subheadings by subject or by type of material. To do this, place a set of `\citationsubject` commands at the top of your document, or after a `\printbibliography` command. Each command takes two mandatory arguments and one optional argument. The first mandatory argument indicates the nickname that you will use in your document to attach a citation to the category. The second is the section heading that you wish to see displayed for that section of the bibliography.

The optional argument may be used for two purposes. First, you may want to further divide your bibliography into sub-subheadings. You can specify a such a subcategory by putting a `2` as the optional argument to the `\citationsubject` command. Otherwise, a first-level header is used. Where

[5] This feature (support for multiple citation styles in a single document) will be added after the release of BibTeX 1.0.

the \printbibliography command is issued, the parts and sub-parts of the bibiliography will be produced in the order in which they were declared.

For some types of citation, you may wish to produce a table that shows more detail about citations made. With CAMEL, simply add an optional argument to the \citationsubject command applicable to your desired table which includes an o= argument of three letters (giving the extension of the raw data output file), and an i= argument of three letters (giving the extension of the makeindex-processed input file). Optionally, you may also include arguments of p or P. With p, the output file will be passed the page or section numbers given in pinpoint arguments to the \source command. With P, index-style page references to the main text are turned on. With or without the P or p,[6] the export file should be processed with makeindex using CAMEL's own camel.ist file to produce the import file. This table-creation feature exists principally for typesetting tables of statutes.

\forcefootnotes

If you are preparing a document that is to be, or may be, prepared with citations in footnotes at the bottom of the page, you can save yourself some typing (and make it easier to produce the same document in other formats) by using the \forcefootnotes command at the top of the document. In some styles this command has no effect since, for example, a footnote containing only "[2]" would look rather peculiar.

\newinterword

In a string citation, you may want to use a connecting word that is not already in CAMEL's internal list. You can add a word to this list with the \newinterword command. This command takes two mandatory arguments. The first is the nickname you wish to give to the word and its associated punctuation. The second is the text to be inserted when you use the new connecting word between two \source commands.

The connecting word may appear either at the beginning of a citation string (as the argument to a != option in the first \source command of a string), or in the middle, between two citations. Two special characters are used to control the appearance of the word in these two positions. A leading punctuation mark should be preceded by a _ character. This will suppress such a punctuation mark, and the space following it, if the connecting word is used as a prefix to an initial citation in a string. The ^ character should precede the first word of the phrase associated with the connecting word, and that character should be capitalized. The first character of the phrase will then be forced to lower case if the connecting word is used between two \source commands.

The following examples may be useful for guidance:

- \newinterword{cf}{_; ^C.f.~}
- \newinterword{see-generally}{_. ^See generally\ }

\printbibliography

Place a \printbibliography command in your document wherever you would like the bibliography to appear. If no \citationsubject commands are used at

[6] The syntax of these options is the same as for options to the \source command, discussed above.

the top of the document, `\printbibliography{all}` will produce a full bibliography of all references made in the document up to that point. If `\citationsubject` has been used, then `\printbibliography{all}` will produce the same bibliography, but classified by subject, with the headers specified in the `\citationsubject` commands at the top of the document.

You may also specify a subject to the `\printbibliography` command, instead of `all`. This will produce a bibliography formatted as requested in the `\citationsubject` command. Note that, if the `i=` and `o=` options are used with `\citationsubject`, `makeindex` must be used to generate the input file that will be expected for that subject. Note also that such external tables will be ignored by `\printbibliography{all}` where citation subjects are used.

`\source` The most common command for CAMEL users is the `\source` command, which replaces the `\cite` command of standard LaTeX. It is used to identify the citation in the `.bib` data files, and to provide a logical description of any 'fine-tuning' of the citation form that the user might desire. Because it is frequently used, and must be accessible to authors with more interest in the content of what they are writing than the inner workings of LaTeX, much effort has gone into making the command as simple to use as possible.

Options

The valid options that may be placed inside the first set of square braces are shown below.

Single character options

`a` — Supress printing of author's name in the document.

`t` — Suppress printing of the title in the document.

`l` — Force automatically generated abbreviations to lower case.

`n` — Suppress printing of the citation in the document (equivalent to `\nocite` in standard LaTeX).

`f` — Force the full form of the citation to be printed in the document.

`h` — Suppress automated footnote generation and the use of interwords for the current and following citations. This option should not be used in the middle of a string of citations linked with interwords.

String options

`s=` — Associate the citation with the specified subject. This option can be freely specified, but only has significance when a classified bibliography is being produced. When these subject tags are being used for that purpose, they must be declared at the top of the document, and a given source must be associated with the same subject throughout the document; specifying a different subject from that used in the first use of the citation will produce an error message from LaTeX.

`v=` — Give a volume number for use in the citation. `BOOK` entries in a BibTeX database may be given a volume number. Such citations are assumed to be multi-volume works, and a volume number should be supplied when citing them; the volume number given in a `BOOK` entry itself ignored, and an error message will be produced if an explicit volume number is not given using this option.

`!=` — Precede the citation with the interword given in the argument.

These options may be entered in the option braces in any order, with the single requirement that a comma must end a string option if it is followed by another option.

The number of possible citation forms that can be generated using these command options is very large, and varies from one style to another. The best way of familiarizing yourself with them is to finish reading the next section of this guide, and then do a little experimentation with the various options. This will serve you much better than a verbose set of illustrations here, which would not stick in the memory, would be difficult to use as a reference, and could not in any case cover all of the possibilities.

The following citations are all valid:

- `\source[ats=school]{ucla}`
- `\source[av=8,t]{harvard}`
- `\source[s=town,nat]{oxford}`

A string of citations applying to a particular statement may be entered as follows:

- `\source[`⟨*options*⟩`]{`⟨*nickname*⟩`}[`⟨*page numbers*⟩`]`
  ⟨*interword*⟩
  `\source[`⟨*options*⟩`]{`⟨*nickname*⟩`}[`⟨*page numbers*⟩`]`

Keys

The key given in the squiggly braces should correspond to the key that identifies the entry in the BibTeX or other database that you are using. If you have a series of citations to make, and do not need to give separate options to any of them, you can enter them as a comma-delimited string, in the following manner:

```
\source[s=stooges]{joe,curly,moe}
```

Pinpoints

For most citations, pinpoint references are simply typed between the second pair of square braces, using normal punctuation marks (`, & -`). Spacing is irrelevant, so long as the punctuation marks are entered. A valid simple pinpoint citation might look as follows:

```
\source{major}[2, 7, 9 & 23]
```

In some very special situations, Camel needs more information to handle pinpoint references correctly. Suppose you are entering a citation for a statute, and you intend to produce a table showing where references to each cited section of the statute are made in the text. If your citation includes references to more than one section of the statute, Camel will produce a table entry for each cited section automatically. But what if you are citing a *range* of sections? Camel cannot know whether the range is continuous, or has gaps in it where provisions have been repealed. You therefore need to tell Camel explicitly about each section number—but you don't want the numbers to be printed in the text. To feed citation numbers to the table, but suppress printing in the document, separate the numbers to "hidden" from the rest with a "fence" in the form of the `|` character. A typical citation of this kind might look like this:

```
\source{SaleofGoodsAct}[19-25 | 20, 21, 22, 23, 24]
```

There are two other situations, also related only to the production of tables, where special markings may need to be included in the pinpointing string. These are (a) where you want material to appear only in the text, and (b) where you want material to appear only in the table. The first case may arise where you are citing subsections of a statute. In this situation, you can enclose in `^` characters any text that you want to withhold from the table. A simple citation of this kind might look as follows:

```
\source{VehicleCode}[23(A)^(3)(a)(i)^]
```

Where material should appear only in the table, enclose that pinpoint material in _ characters. This might be necessary where, for example, you are citing two subsections of the same statute. An example might look as follows:

```
\source{LawofPropertyAct}[70(1)(f) & _70(1)_(g)]
```

Finally, there is one more feature that applies only to 'parallel citations' to law cases. If a law case to which you are referring is reported in more than one location (see below), and you want to give a pinpoint to each location, you can do so by using the the = sign between the pinpoint strings. For example:

```
\source{bloggvkermit}[20-25 = 125-26 ]
```

If more parallel pinpoints are given than exist in the BibTeX entry for the case nicknames `bloggvkermit`, the extra pinpoints will be ignored. Similarly, if you give fewer pinpoints than there are parallels in the BibTeX entry, the portions of the entry with no matching pinpoints will be omitted. No fuss, no muss.

1.2 BibTeX features

A number of enhancements for processing BibTeX entries have been incorporated into the Camel BibTeX programmer's library. These enhancements are drawn upon by the `law.bst` style, and should also be used by subsequent Camel style modules. These features are described briefly here. Please see the BibTeX documentation for more information on the format of BibTeX entries et cetera.

Cases

A new entry type, `@CASE`, has been added. This entry type should be used for reported law cases. Although citation forms vary widely between jurisdictions, Camel bases the formatting of the citation on the information contained in the citation, rather than a tag indicating the jurisdiction.

A result of this approach is that there are no 'required' fields in the usual sense that BibTeX will complain if something is missing. Instead, is a set of 'core' fields for each of three different formatting styles. For someone familiar with legal resources, this is actually quite intuitively straightforward.

For reported U.S. law cases, the core fields are `title`, `volume`, `journal`, `pages` and `year`. In addition, you may wish to specify `court`.[7] Procedural histories may not be represented in the BibTeX entry.

For reported Commonwealth cases, use `number` instead of `volume`. The effect of this will be to place the year at the front of the citation in square braces, with the `number` and `journal` following it. Again, you may specify `court` optionally.

For cases reported in ephemeral media such as newspapers, leave out `volume` and `number`, and give the full date in the `year` field instead (see below for the formatting of dates). The formatting of the citation will adjust accordingly.[8]

[7] This is not yet implemented, but can be and should be.

[8] I'm not sure whether this works with parallels yet. If it doesn't but you need it to, let me know and I'll fix it.

For cases reported in jurisdictions such as Japan that refer to cases by date rather than title, use `casedate`, `court`, `journal`, `volume`, `pages` and `year`. Optionally, you may also wish to include `divno` and `division`, to specify the exact identity of the deciding court.

Statutes

Support for statutes is still in its infancy. You need, at minimum, to enter `title`, `year` and `jurisdiction`. Supported jurisdictions are `japan`, `singapore` and `england`.

Dates

The entry of dates has been considerably simplified in CAMEL BIBTEX entries. Always use the `year` field (or, if appropriate, the `casedate` field). Months may be entered as numbers or as a three-character string. The following date forms are all valid:

```
year = {10 jun 1995},
year = {jun 10 1995},
year = {jun 1995},
year = {1995},
year = {10/06/95},
```

Parallel citations

For `@CASE` entries, it is possible to enter multiple citations in a single BIBTEX entry using a short form of citation in a field called `cites`, separating entries with a `=` character.[9] There are several valid syntaxes for this field, which closely follow the citation format used in the Blue Book for the relevant sources. They are:

```
cites = {[1995] 1 All ER 25 = [1995] 2 WLR 125},
cites = {123 Cal.3d 237 (1995) = 124 S.W.2d 235 (1995)},
```

You can create an arbitrary number of parallel citations in this way.[10] Be sure to read the notes on pinpointing, above, for details on how to specify pinpoint references to parallel citations.

1.3 Processing documents

Ordinary processing

For a simple document containing no cross-references other than those produced by CAMEL, printing requires one run of LATEX, one run of BIBTEX, and a second

[9] It actually does work. *Halsall v Brizell*, [1957] Ch 169; [1957] 1 All ER 371.

[10] Cross-references are complex to format, but they work correctly. *Id.* In a competition for speed and accuracy between typists equipped with LATEX/CAMEL and with WordPerfect, who do you reckon would win?

run of LaTeX. Because Camel absorbs all citation information at the start of the document (at the point of the `\citationdata` or `\bibliographymanager` commands), there is no need for the third LaTeX run required in the standard LaTeX setup. You may, however, need additional runs of LaTeX to resolve other cross-references in the document, or to work out misspellings and other errors.

Producing tables

When the `o=` and `i=` options are used for one or more declared citation subjects, those subjects will be excluded from the bibliography. Instead, Camel will write an external table file according to the specifications given in the `\citationsubject` command. Any such files must then be processed into properly formatted lists using `makeindex` before they will appear in the printed document.[11] On a UNIX or DOS system you would type the following for a table declared with the export extension `.atb` and the input extension `.art`:

`makeindex -s camel.ist -o` ⟨*file name*⟩`.art` ⟨*file name*⟩`.atb`

If the subject associated with this table is `article`, the table can then be placed in the document for the next LaTeX run using the command:

- `\printcitationtable{article}`.

When the option `p` is used with the `\citationsubject` command and the `i=` and `o=` options, the table will include any section or other numbers given as optional arguments to `\source` commands in the document. Because `makeindex` sorts alphabetically, not numerically, the section numbers may be out of their proper order in the `makeindex`-processed file. This must be corrected by hand.

[11] The documentation for `makeindex` will explain how it should be set up for your system. On a UNIX or DOS installation, the basic requirements are that the `makeindex` program be in your search path, and that the appropriate environment variable (usually `indexstyle`) be set—this tells `makeindex` where to find the style files that tell it exactly how to format the finished table lists.

2 Notes for Implementors

2.1 Getting started

The purpose of this section is to provide style designers with all of the information required to write a new bibliography style from scratch, drawing on the powerful facilities available in BibTeX and the Camel style engine. Before you start, you should have a sound knowledge of BibTeX primitives, and of the programmer's function library contained in the file `camel.dtx`. A bibliography style consists of a `.bst` file, a `.cst` file and a `.cit` file. The `.bst` file should produce entries suitable for digestion by Camel for all of the BibTeX entry types supported by the style.

The modules of the Camel system work together in an integrated fashion to format citations. Raw citation details are kept in `.bib` databases. The function of the *Camel* system is to massage those details into formatted citations, and include them in appropriate locations in typeset pages. When a document invoking the Camel package encounters a `\citationstyle` command, the relevant `.cst` and `.cit` files are loaded. A corresponding `\bibstyle` entry is also written on the `.aux` file. A `\citationdata` command causes a relevant `\bibdata` command to be written there as well. If a `\bibliographymanager` declaration is used instead, a `\bibdata` entry calling on the file `camel.bib` is written to `.aux`. When a `\source` command is encountered during processing, code contained in `camel.sty` writes a `\citation` entry for that citation on the `.aux` file. In this way, an orthodox set of entries and files for use with BibTeX is generated.

If an external bibliography manager is used, the file `camel.bib` must be converted into a valid `.bib` file, using whatever tools for finding and replacing key entries the database manager has to offer.

BibTeX must then be run on the document. This reads the entries on the `.aux` file, opens the `.bib` databases, reads them, and writes the result of its ruminations on a `.bbl` file. At this point, the operation of Camel diverges sharply from that of conventional LaTeX bibliography tools. With Camel, BibTeX is used to generate `\lexibib` entries on the `.bbl` file. Such entries parse the citation into logical typesetting units, but the typefaces used within the units, and the punctuation that appears between them, is not determined by BibTeX. The tasks of punctuation and typeface selection are carried out on the second run of LaTeX on the document, using the details specified in the `.cit` file for the Camel citaton style being used.

It is worth spending some time pondering this programming flow; there are good reasons for spreading the formatting work over a number of different modules in this way, and it is important to understand the design before embarking on the drafting of a style.

2.2 The `.cst` file

The `.cst` file should contain, at least, the following macros:

`\@law@print:` This is the macro that actually prints the citations in the document and in the bibliography. Its structure is discussed below in the commented

code listing of the default `\@law@print` command, which draws on most of the conditional handles provided by CAMEL.

`\@law@wordlist:` This is a list macro consisting of paired interword nicknames and corresponding punctuation/text sets. Possible changes you may wish to make to this file are discussed in the commented code description of this macro.

Additional macros (including re-definitions of the macros contained in the CAMEL engine) may be included in the `.cst` file. Be sure to document your work, however; CAMEL is among the more complex packages written for use with TEX, and documentation is essential if problems that arise are to be sorted out quickly and efficiently.

Concerning the `\@law@wordlist` macro, it is best to examine the existing macro in `law.dtx`, and read the user documentation on the `\newinterword` command. With this information, you should be able to make any changes you require in the behaviour of the list to suit your new style. You should not omit items from the list—this is important to maintain compatibility between styles. You may want to add to the list, and this is welcomed—drop a note to Frank Bennett on `fb@soas.ac.uk` to assure that your new addition is incorporated into the `law.dtx` list, which is used as a model, so others will include it in their styles too.

Drafting the `\@law@print` macro may be difficult to approach at first. If you examine the example in the `law.dtx` file, you may be somewhat intimidated by the size and complexity of the macro listed there. But in fact for most styles this macro should be reasonably simple to write; the `law.dtx` example is complex because it automates a very complex set of style rules applying to legal materials.

Essentially, what `\@law@print` does is print a citation in a particular form according to a set of conditions that can be recognized from toggles in effect when the macro is run by the CAMEL engine. The toggles that may be drawn upon are described here. For further information, study the example in `law.dtx`—bearing in mind that your own macro will probably be simpler to draft.

2.3 Toggles and hooks for the `.cit` file

2.3.1 Citation history

`\if@law@firstuseofcite` This is `true` during the first expansion of a citation nickname in the text.

`\if@justabove` This is set to `true` where (a) the author, title, source, page and citation trailer are identical in the last citation used in a footnote and in the current citation, (b) the last citation used in a footnote was the only citation in that footnote, and (c) a pinpoint was not used in the previous citation or, if there was such a pinpoint, a pinpoint is also used in the current citation.

`\if@l@quiteexact` This is set to `true` where (a) `\@law@justabove` is set `true` and (b) a pinpoint was used in both citations, and (c) the pinpoints match exactly.

2.3.2 Formatting toggles

`\if@law@intextcite` This is set `true` by the `h` option to the `\source` command. Use it to select a

special in-text form for the citation.

`\if@law@multipages` This is set `true` if more than one pinpoint reference is used (i.e. any of `,` `&` or `-` occur in the pinpoint argument).

`\if@law@printcite` If this is `true` the citation should print. Otherwise it should not. It should also trigger suppression of footnote forcing.

`\if@law@printauthor` If this is `true` the name of the author prints.

`\if@law@printtitle` If this is `true` the title of the work prints.

`\if@law@longcite` If this is `true` the citation should be printed in its full form. This is set `true` by the `f` option to the `\source` command.

2.3.3 typefaces

`\@ltok@authormainface` This is the typeface set in the `.cit` file for author names. It, and all other `\@ltok@` control strings are token registers that must be expanded using `\the`.

`\@ltok@authoroptionface` This is the typeface selected by enclosing text in the author portion of the citation within `\\` commands in the current citation's type (i.e. as defined in its `\newcitestyle` entry).

`\@ltok@titlemainface` The typeface used for the title portion of the citation.

`\@ltok@titleoptionface` The alternative `\\` typeface within the title argument to `\lexibib`.

`\@ltok@citefirstmainface` The typeface used for the source portion of the citation.

`\@ltok@citefirstoptionface` The alternative for the source portion of the citation.

`\@law@firstslash` `\@law@secondslash` These two commands should be defined as follows in your `\@law@print` command before citation elements are expanded. This could be done within CAMEL itself, but this gives you a little more hands-on control in style design maybe.

```
\def\@law@firstslash{\begingroup\def\\{\@law@secondslash}%
\the\ltokspecialface}%
\def\@law@secondslash{\endgroup\def\\{\@law@firstslash}}%
\def\\{\@law@firstslash}%
```

2.3.4 Citation details

`\@ltok@whereitsat` This actually contains the text that is used to flag either a note or a page number (i.e. `{\ note\ }` or `{\ page\ }`). This is used only where citation cross-referencing is required.

`\@ltok@stylename` This stores the citation type used to declare the citation with `\lexibib`. This is mainly used for internal purposes by CAMEL, but if you're printing a style or proof sheet, or are just trying to debug a style, you may have a use for it.

`\@ltok@nickname` This stores the nickname of the citation.

`\@ltok@author` This stores the author portion of the citation.

`\@ltok@name` This stores the title portion of the citation.

`\@ltok@citefirst` This stores the source portion of the citation.

`\@ltok@citepage` This stores the page number (*not* the pinpoint!) of the citation.

`\@ltok@citelast` This stores the citation trailer.

`\@ltok@pageorfootno` This stores either the page or the footnote number at which the citation first occurred.

2.3.5 Bridges

`\@ltok@atot` This contains the punctuation bridge, declared in the `.cit` file for the current citation type, that is to be placed between the author and the title elements of the citation.

`\@ltok@ttocone` This contains the bridge that is to be placed between the title and the source.

`\@ltok@conetop` This contains the (singular) bridge that goes between the source and the page number.

`\@ltok@conetopplural` This contains the alternative plural bridge that goes between the source and the page number.

`\@ltok@ptop` This contains the bridge that goes between a page reference giving the location of an article, for example, and the pinpoint reference.

`\@ltok@ptoctwo` This contains the bridge that goes between any page numbers and the citation trailer.

`\@ltok@atbridge` This contains the bridge that goes between an abbreviated citation (i.e. *Id.* or somesuch) and a pinpoint.

`\@ltok@atbridgeplural` This contains the plural bridge for abbreviated citations.

2.3.6 Pinpoint printing

`\@law@barkill` The pinpoint information is contained in the token register `\@ltok@argtwo`. To print the pinpoint, you should always use the following form, which will induce the correct behaviour of the string expansion engine within CAMEL.

```
\@law@barkill\the\@ltok@argtwo\relax
```

2.4 The `.cit` file

The `.cit` file should contain instances of the `\newcitestyle` command and associated arguments. This defines a CAMEL style instance for each of the entry types supported by the style's `.bst` file. If you find it necessary to add a new entry type to the style defined in the `.bst` file for your style, you must also define a CAMEL style instance in the `.cit` file applying to this new entry type.

The arguments to the `\newcitestyle` command have the functions described in the following footnoted illustration:

```
\newcitestyle{article[a]}%
 {rsirsr[b]A[c]}%
 {[a],\ [d][t],\ [d] [c]\ [d] [p],\ [d] (pl),\ [d] [rp]\ [d] [e]:[id]\ at~[d]
(pl)\ at~[d] [xrf]}%
 {6}[e]%
{{#1}[f]{#2}[f] {#3}[f]{#4}[f] {#5}[f]{(#6)}[f]}%
```

[a] This gives the name of the citation template to be generated.

[b] The first part of this list of arguments consists always of exactly six letters: the first three pairs set the main and alternative typefaces for the author, title and first-cite-part portions of the citation. `r`, `s`, `b` and `i` select roman, small caps, boldface and italic type. These letter flags are not case sensitive. The alternative typefaces may then be specially selected in the text arguments to a citation using `\\...\\`.

[c] The last letter in this list sets the classification of citations declared with the resulting style macro. `a`, `b`, `c` and `s` will classify all citations generated by this citation template as being to articles, books, cases or statutes, respectively. This letter flag is not case sensitive.

[d] These are the citation bridges that will be placed between the portions of the citation indicated. Two bridges, singular and plural, are given between `[p]` (page) and `[rp]` (reference page), and between `[id]` (*id.* or *supra*) and `[xrf]` (cross-reference).

[e] This argument states how many arguments the finished citation template will accept. It must be a number between `1` and `6`.

[f] Six pairs of matched braces must appear inside this argument. Any other than the first (which represents the citation nickname) may be left empty, but the bridges must of course take this into account. The number of arguments inserted here must correspond with the number stated on the line above.

2.5 The `.bst` file

2.5.1 Entry types

Most of the work in drafting a new style is in writing entry type functions for inclusions in the `.bst` file. This is, inevitably, an arduous task, because of the number of possibilities that must be covered. The task is made easier, however, by the `.bst` file programming library contained in the file `camel.dtx`. This library provides a set of functions for the manipulation of fields pushed onto the stack in the entry type functions themselves. This approach increases the transparency of your code, because you can see in the entry type function itself exactly what fields are being used, and what logical and formatting operations are being carried out on them.

Using the library is highly recommended. If you find that you require functions that are not presently available in the library, you can make your solutions available to others by adhering to a few design guidelines, and passing the function on to `fb@soas.ac.uk` for inclusion in the Camel distribution. The guidelines are actually quite simple. No library function should directly manipulate a field, but should instead expect to find the field on the stack when it is invoked. No library function should make assumptions about the setting of variables; any parameters should also be passed to the stack before the library function is invoked. When

finished, a library function should always leave a fixed number of items on the stack, in a fixed order. And a library function should not pass data to subsequent functions in the form of variables. Finally, a function should be documented concerning (a) the number of items it expects to find on the stack, and in what order, (b) the number of items it leaves on the stack, and in what order, and (c) the effect of any toggle strings that may be fed to the function.

You may find it necessary, in drafting a function, to divide its operation into sub-functions. As between sub-functions, the ban on passing data through variables need not be followed, so long as the set of functions adheres to the rule.

2.5.2 About cross-referencing

About cross-referencing, Oren Patashnik wrote:

> Now come the cross-referencing functions (these are invoked because one entry in the database file(s) cross-references another, by giving the other entry's database key in a `crossref` field). This feature allows one or more titled things that are part of a larger titled thing to cross-reference the larger thing. These styles allow for five posibilities:
>
> 1. an **article** may cross-reference an **article**;
> 2. a cross-reference to a **book** may be made in an entry for:
>   (a) a **book**;
>   (b) an **inbook**; or
>   (c) an **incollection**; and
> 3. an **inproceedings** may cross-reference a **proceedings**.
>
> Each of these is explained in more detail later.

This is the authoritative statement on the proper use of cross-referencing within a `.bib` file. The further comments I have to offer here are a non-authoritative gloss, but do accurately reflect the assumptions I made in writing the `.bst` code for the `law.dtx` file.

`article`**:** A cross-reference from one `article` entry to another may be used for special issues of a journal, such as those dedicated to a single topic. The referenced entry should include all details relevant to the issue as a whole, including `volume`, `journal` and `booktitle` (this last for the name or topic description of a special issue), and `type` (this defaults to "`Special issue`").

`book`**:** This cross-reference is not recognized by the `law.dtx` style. It could logically be used for a series, but this is little trouble to type into the original `@book` entry itself, and a series is not something that should appear separately on the reference list.

`inbook`**:** This is used for cross-referencing a chapter or other subdivision within a work by a single author to that work.

`incollection`: This is used for cross-referencing a work by multiple authors. The typeface conventions used in the Blue Book style differ for `inbook` and `incollection` citations, as defined here, so this distinction should be honored in building a database.

`inproceedings`: This is used only cross-referencing articles appearing to a proceedings volume to that volume.

3 The Citation Engine: `camel.sty`

3.1 Initialization

3.1.1 Hello!

To start things off on the right foot, we tell the world who we are.

```
⟨*lexitex⟩
\NeedsTeXFormat{LaTeX2e}[1994/12/01] \ProvidesPackage{camel}
[1995/13/01 v1.0 General bibliography engine (Frank Bennett)]
\@ifclassloaded{ltxdoc}{}{\RequirePackage{indexdmj}}
```

3.1.2 Customizable text elements

`\supra` The following definition specifies the string used to refer to a cite that occurs in a previous footnote, or in a preceding citation to the same source within the same footnote. It is not used in cross-referencing cases and statutes. The extra internal braces only serve to limit the scope of `\em`. An italic space (`\/`) is not added, since this might be followed by a period.

```
\def\supra {, {\em supra}}%
```

The `\Id` macro defined here is used in referring to immediately preceding works, in the same or in a preceding footnote. If the context demands that *Id.* be lower case when it occurs, the user can switch it to lower case by entering `\lowcase` immediately before an in-text citation declaration or citation nickname. The lower case macro thus invoked turns itself off after *id.* has printed, so no grouping is required.

```
\def\Id{{\em\@law@lettercase Id.}}%
```

These two tokens hold the text used to indicate the note or page number where a previous reference occurs. These are only used in short-form citations.

```
\newtoks\@ltok@userconetop%
\newtoks\@ltok@userconetopplural%
\newtoks\@ltok@userptop%
\newtoks\@ltok@userptoctwo%
\newtoks\@ltok@useratbridge%
\newtoks\@ltok@useratbridgeplural%
\newtoks\@ltok@usercitefirst
\newtoks\@ltok@usercitepage
\newtoks\@ltok@usercitelast
```

```
\newtoks\@ltok@infoot \@ltok@infoot={\ note\ }
\newtoks\@ltok@onpage
\@ltok@onpage={\ page\ }
```

The following tokens are used in making proof sheets. Change the content of these to taste.

```
\newtoks\@ltok@proofcitelast
 \@ltok@proofcitelast{{Editor, publisher and stuff, 1994}}
\newtoks\@ltok@proofpage
 \@ltok@proofpage{{12345}}
\newtoks\@ltok@proofciteone
 \@ltok@proofciteone{{Source}}
\newtoks\@ltok@prooftitle
 \@ltok@prooftitle{{Title of work}}
\newtoks\@ltok@proofauthor
 \@ltok@proofauthor{{Author's name}}
```

3.1.3 User selectable switches

The Blue Book[12] allows us to refer to statutes in a long form or a short form in subsequent references. The long form consists of the name of the Act (if any) plus its source details and the section number. The short form consists of the source details alone.[13] I'm all in favour of saving ink; the default is for short form statute citations. You can have long form subsequent cites by setting `\statuteverboseon` at the top of your document. If you need some other form, let me know; statutes are complex, and rather than trying to do everything from the get-go, this really has to be demand led.

```
\global\let\@law@delim\relax
\global\let\@law@pingroup\relax
\global\let\@law@pinstart\relax
\global\let\@law@pinend\relax
\newif\if@law@statuteverbose
\def\statuteverboseon{\@law@statuteverbosetrue}
\def\statuteverboseoff{\@law@statuteverbosefalse}
\statuteverboseoff
```

`\forcefootnotes` This fun little item is used to toggle the `\lexicite` macros to create a footnote if none was typed literally into the text. Great finger-saver for shortie footnotes. Conceived on a whim, implementing this turned out to be non-trivial. I hope you find it useful!

```
\newif\if@law@forcefootnotes
\def\forcefootnotes{\global\@law@forcefootnotestrue}
```

3.1.4 Shorthanding essentials

The following definitions are established as a means of cutting text using the `_`, `^` and `|` characters. Although we make them active here, they are *not* read as

[12] A Uniform System of Citation (19th ed. 1989).
[13] *See Id.* at 25.

active characters. CAMEL should in no way interfere with styles that require these characters to be set active. The active characters created here are used only for "internal purposes".

```
{
 \catcode`\_=13
 \catcode`\^=13
 \catcode`\|=13
 \gdef\@law@barkill{\def\@law@comma{, }%
                    \def\@law@ampersand{\ \& }%
                    \def\@law@dash{--}%
                    \def\@law@delim{}\def_##1_{}\def|##1\relax{}\def^{}}%
 \gdef\@law@barnil{\def_{}\def^##1^{}\def|{}}
 \gdef\@law@barinfull{\def^{}}
 \gdef\@law@barinshort{\def^##1^{}}
}
```

```
\def\bridges#1#2#3#4#5#6{%
 \@ltok@userconetop{#1}%
 \@ltok@userconetopplural{#2}%
 \@ltok@userptop{#3}%
 \@ltok@userptoctwo{#4}%
 \@ltok@useratbridge{#5}%
 \@ltok@useratbridgeplural{#6}%
}
```

3.1.5 Miscellaneous other macros

Sometimes it might be nice to avoid using `--` for some reason, so a macro is defined here. Same thing for exclamation points. One possible use for the latter is to avoid errors when an exported table is sorted by `makeindex`.

```
\def\dash{--}
\def\exclaim{!}%
```

The `\Id` macro contains its own period. Therefore a period supplied by the user becomes redundant. This macro is redefined to gobble this extra period when necessary.

```
\def\@law@gobble{}
```

If you are into software history, you might have cause to refer to LEXITEX, so its logo and the LEXIBIBlogo (also R.I.P.) are defined here. They are `\protect`ed just like the LATEX logo.

```
\def\p@LexiTeX{{\reset@font\rm\ \unskip\hbox{L%
 \kern-.06em\hbox{\sc e}%
 \kern-.05em\hbox{\sc x}%
 \kern-.05em\lower.65ex\hbox{I}%
 \kern-.18emT%
 \kern-.1667em\lower.65ex\hbox{E}%
 \kern-.125emX}}}%
\def\LexiTeX{\protect\p@LexiTeX}%
\def\p@LexiBib{{\reset@font\rm\ \unskip\hbox{L%
```

```
 \kern-.06em\hbox{\sc e}%
 \kern-.05em\hbox{\sc x}%
 \kern-.05em\hbox{\sc i}%
 \kern-.06em{\sc B}%
 \kern-.05em\hbox{\sc i}%
 \kern-.05em{\sc b}}}}%
\def\LexiBib{\protect\p@LexiBib}%
```

3.1.6 New environment

There is just one new environment in CAMEL, and all it does is set the list-input toggle to true, so that input citations will feed into memory but not print.

```
\newenvironment{lexilist}%
{\message{Loading citation data}\@law@listinputtingtrue}%
 {\@law@listinputtingfalse}
```

3.1.7 Token registers

The following are used as temporary token registers in various places.

```
\newtoks\@ltok@a
\newtoks\@ltok@b
\newtoks\@ltok@argtwo
\newtoks\@ltok@pagesstring
```

The following token registers are used for temporary storage of typeface information.

```
\newtoks\@ltok@authormainface
\newtoks\@ltok@authoroptionface
\newtoks\@ltok@titlemainface
\newtoks\@ltok@titleoptionface
\newtoks\@ltok@citefirstmainface
\newtoks\@ltok@citefirstoptionface
\newtoks\ltokspecialface
```

One token register is needed to store the name of the current cite type (as declared using `\newcitestyle` in the `.cit` file). With this string in hand, all of the typeface and punctuation bridge information can be recovered for that type of citation.

```
\newtoks\@ltok@citetype%
```

Tokens to store info on whether a cite is in a footnote or in the main text, and the name of the cite style, in case we need it.

```
\newtoks\@ltok@whereitsat
\newtoks\@ltok@stylename
```

Ho-hum. Now we're going to need a bunch of token registers to stash the information for a cite. Token registers have to be used in processing, in order to provide expansion control.

```
\newtoks\@ltok@nickname
\newtoks\@ltok@author
\newtoks\@ltok@name
```

```
\newtoks\@ltok@citefirst
\newtoks\@ltok@citepage
\newtoks\@ltok@citelast
\newtoks\@ltok@pageorfootno
\newtoks\@ltok@hereinafter
\newtoks\@ltok@bigsourcecite
\newtoks\@ltok@bigsourcepage
```

We also need token registers for citation bridge information.

```
\newtoks\@ltok@atot
\newtoks\@ltok@ttocone
\newtoks\@ltok@conetop
\newtoks\@ltok@conetopplural
\newtoks\@ltok@ptop
\newtoks\@ltok@ptoctwo
\newtoks\@ltok@atbridge
\newtoks\@ltok@atbridgeplural

\newtoks\@ltok@plistmacro
```

A counter is required to keep track of the number of cites in a footnote, and to store the number that occurred in the previous footnote.

```
\newcounter{citesinfoot}
\newcounter{citeslastfoot}
\newcounter{law@paranormal}
\newcounter{law@parapin}
\newcounter{law@paracounter}
```

3.1.8 Peek-word-ahead macro

This macro will not work if it is placed after the string of `\newif`s below. I haven't a clue as to why. I'm leery of whether this will cause problems with other styles. If you have problems, contact me on `fb@soas.ac.uk`.

`\@ifoverword` To provide inter-cite nicknames and to handle the bracketing or forcing of multiple citations, we need a macro that can peek one word ahead, as opposed to one character. All we need is something to fetch the next word, with leading and trailing spaces cut off, then call the LaTeX `\@ifnch` routine that does the work of `\@ifnextchar`. After execution, the next word is stored in `\@law@word`, and any space immediately preceding the word is stored in `\@law@space`. These can be used in the two alternative arguments to the macro to restore the *status quo ante* if desired. During the debugging of this, I noticed that a preceding space was never found, even when it existed. I'm puzzled by this, but solving any problems it might give rise to will have to wait for another day.

```
\long\def\@ifoverword#1#2#3{\let\reserved@e=#1%
  \def\reserved@a{#2}\def\reserved@b{#3}%
  \gdef\@preoverwordspace{}%
  \futurelet\reserved@c\@ifowd}
\def\@ifowd{\ifx\reserved@c\@sptoken%
      \gdef\@preoverwordspace{ }%
```

```
      \let\reserved@d\@xifowd%
      \else\long\def\reserved@d##1 {\long\def\@overword{##1}\@@ifowd}%
      \fi \reserved@d}
\def\@@ifowd{\futurelet\reserved@c\@ifnch}
\def\:{\@xifowd} \expandafter\def\: {\futurelet\reserved@c\@ifowd}
```

3.1.9 If initializations

We need a toggle so we can turn the print routine on and off; this is necessary for list inputting, and for making phantom references to sources in order to handle weird citation forms.

```
\newif\if@law@usepages
\newif\if@law@table
\newif\if@law@usepinpoints
\newif\if@law@maketable
\newif\if@law@printcite
```

The next one is used by the list-input environment.

```
\newif\if@law@listinputting
\newif\if@law@bibentry
```

We also need to know whether a citation is being used for the first time. In some styles, the first citation is treated specially.

```
\newif\if@law@firstuseofcite
```

The next if will be used to toggle the print routine between long and short citation forms.

```
\newif\if@law@longcite
```

Similar to the above is the following toggle, used to force a long citation used in the text. This should prevent the creation of footnotes if they are being forced.

```
\newif\if@law@intextcite
```

This will tell us whether we are in a footnote or not.

```
\newif\if@law@infoot
```

The next `\if` will let us know if the same work was cited immediately before the current cite, and if so, whether we should pay any attention to a page reference, if given. These are adjusted by `\law@justabovecheck`.

```
\newif\if@justabove
\newif\if@l@quiteexact
```

The next condition will control the way print output is done in short citation forms; it should set to false for articles and books, and to true for cases and statutes.

```
\newif\if@nosupra
```

The next is used to signal the presence of `++` as the argument linking a reference to a page/section number argument. This toggles plural bridges on, and tells the file output routine that, for statutes, each section number should be attached to a complete copy of the citation information. These copies are later reassembled in a set of table entries by `makeindex`.

```
\newif\if@law@multipages
\newif\if@law@printauthor%
\newif\if@law@printtitle%
\newif\if@law@requiresubjects%
\newif\if@law@subjectfound%
```

3.1.10 Macro initializations

The following is a grab-bag of small macros with simple functions. There is more to say about what other macros use them for than about they themselves, so I've put them in the front matter of the code.

First is a little something to force a lowercase *id.* if necessary. The operation of `\normcase` is obvious enough; it defines the case-switching macro to a no-op. `\lowcase` works by setting the case toggle to force a lowercase letter. The use of `\aftergroup` allows everything to be turned back on without losing the lowercase letter. This was designed by trial and error; there may be a better way.

```
\def\normcase{\def\@law@lettercase{}}
\def\lowcase{\def\@law@lettercase##1{%
  {\aftergroup\normcase\lowercase{##1}}}}
\normcase
```

The CAMEL nickname macros are now kept out of the name space by executing them via `\csname\endcsname`, and adding a prefix that makes the macro inaccessible by direct reference within the document. This change was recommended by a user—sorry out there, I can't remember who! Note that any nickname can now contain numbers and other stuff, which is handy for cases, which should be organized according to deciding court and date. If in-text references are being `forced` into footnotes, the footnote toggles are set to true here; the footnote itself is created later, after any optional arguments have been gobbled. Note that footnote forcing will only take effect for `\lexicite` citations, not for in-text declarations. We need a new counter for footnotes, so that we can keep track of footnote numbers while processing forced footnotes outside of the footnote environment.

```
\newcount\c@law@footnote%
\def\volno{}%
```

`\@law@getsubjectheader` This macro is used to grab the header appropriate to a given declared subject. The subjects, headers and other related information are kept in a single list macro for quick access.

```
\def\@law@headersearchend{\camelrefname}
\def\@law@abandonheadersearch#1\@law@headersearchend{}
\def\@law@headersearch#1#2#3#4{%
  \def\@law@temptwo{#1}%
  \ifx\@law@temp\@law@temptwo%
   \let\@law@headersearchend\relax%
   \let\\=\@law@abandonheadersearch%
   {}#3%
  \fi}
\def\@law@getsubjectheader#1{%
```

```
  {\def\@law@temp{#1}%
  \let\\=\@law@headersearch\@law@subjectlist\@law@headersearchend}}
\let\@law@bibformatsearchend\relax
\def\@law@abandonbibformatsearch#1\@law@bibformatsearchend{}
\def\@law@bibformatsearch#1#2#3#4{%
  \def\@law@temptwo{#1}%
  \ifx\@law@temp\@law@temptwo%
   \let\\=\@law@abandonbibformatsearch%
   {}#4%
  \fi}
\def\@law@getbibformat#1{%
  {\def\@law@temp{#1}%
  \let\\=\@law@bibformatsearch\@law@subjectlist\@law@bibformatsearchend}}
```

`\@law@confirmsubject` This parser checks whether a given subject exists in the list of declared subjects. It takes no action, only issues a warning.

```
\def\@law@subjectsearchend{%
  \ifx\@law@citesubject\empty%
   \@camel@error{No subject declared.^^J
    After declaring a subject or subjects using
    \string\citationsubject,^^J
    you have neglected to give a subject (using the s= option)^^J
    for one of your \string\source\space commands}\@ehc%
  \else%
   \@camel@error{Undeclared subject `\@law@citesubject'.^^J
    An undeclared subject has been given as an argument to
    a \string\source\space command.^^J  You must first %
    declare subjects using \string\citationsubject}\@ehc%
  \fi}
\def\@law@abandonsubjectsearch#1\@law@subjectsearchend{}
\def\@law@subjectsearch#1#2#3#4{%
  \def\@law@temptwo{#1}%
  \ifx\@law@citesubject\@law@temptwo%
   \let\@law@subjectsearchend\relax%
   \let\\=\@law@abandonsubjectsearch%
  \fi}
\def\@law@confirmsubject{%
  {\let\\=\@law@subjectsearch\@law@subjectlist\@law@subjectsearchend}}
```

`\@law@maybeaddcitesubject` This macro adds the nickname of the current citation to the list of nicknames under the current citation subject. It will only do this if the nickname does not already exist in the list. The effect is to keep the lists in citation order. Note that a citation may be listed under more than one citation subject. This may be useful, so it is not prevented.

```
\def\@law@citesubjectcheckend{%
  \expandafter\let\expandafter\@law@temptwo%
   \csname @law@\@law@citesubject @citelist\endcsname%
  \@ltok@a=\expandafter{\@law@temptwo}%
  \expandafter\xdef%
```

```
  \csname @law@\@law@citesubject @citelist\endcsname{%
    \the\@ltok@a\noexpand\\%
  \noexpand{\the\@ltok@nickname\noexpand}}}
\def\@law@abandoncitesubjectcheck#1\@law@citesubjectcheckend{}
\def\@law@citesubjectcheck#1{%
  \def\@law@temptwo{#1}%
  \ifx\@law@nickname\@law@temptwo%
   \let\@law@citesubjectcheckend\relax%
   \let\\=\@law@abandoncitesubjectcheck%
  \fi}
\def\@law@maybeaddcitesubject{%
  {\let\\=\@law@citesubjectcheck%
  \csname @law@\@law@citesubject @citelist\endcsname%
   \@law@citesubjectcheckend}}
```

There are to be three sets of specialized parsing macros in LexiTeX, all three of which are used on the options to the `\lexicite` command. One parser reads a list of options. The second reads a comma, `&`, and `-` delimited list and writes the contents as a macro argument. The third reads the same comma, `&` and `-` delimited list and prints it directly on the output after a slight amount of formatting work.

`\@law@plone` Does a comparison, executes the associated option if there is a match.

```
\def\@law@plone#1#2{\def\@law@listitem{#1}%
  \ifx\@law@optionitem\@law@listitem #2%
  \let\\=\@law@finish\fi}
```

`\@law@end` The internal ending character is also common to all parsers.

```
\def\@law@end{,}
```

`\@law@finish` If there is a match, we want to ignore the rest of the list. This is macro is defined and "kept on the shelf" until such time as we need to cut off the rest of the executable list macro below.

```
\def\@law@finish#1\@law@nomatch{}
```

`\@law@plnomatch` If when we execute the list of options we do not find a match, we want an error message to go to the terminal. This is the macro that gives the appropriate error message.

```
\def\@law@plnomatch{\@camel@error{Citation Warning: Invalid
    option `\@law@optionitem'
    given to \string\source.^^J%
   Valid options are:^^J
    a (supresses printing of author's name in main text)^^J
    t (suppresses printing of title of work in main text)^^J
    n (suppresses printing of citation in main text)^^J
    l (forces some short forms to lower case)^^J
    f (forces printing of full cite in text)^^J
    s=<subject> (associates a subject declared using
     `\string\bibsubject'^^J
     with the citation; if used once, the same `s=' option must^^J
```

```
     be used consistently for that key throughout the document)^^J
    v=<volume>  (associates a volume number with the current^^J
                citation, in the main text only; this is used for^^J
                multi-volume works)^^J%
To avoid errors in formatting, use commas in the option string^^J%
only where they are logically necessary to the sense of the^^J%
string.  Therefore \string\cite[ats=birds]{key} and
\string\cite[s=birds,at]{key}^^J%
are correct, while \string\cite[a,t,s=birds]{key} is NOT
correct}\@ehc}
```

\@law@sourceoptionlist

The list of options is simple and expansible. The syntax of the last two lines is required if everything is to work as expected. The items in the help message should probably be drawn from here, to provide a hook for expansion if other styles want to tinker with the options to `\cite`.

```
\def\@law@sourceoptionlist{%
 \\{a}{\@law@printauthorfalse}%
 \\{t}{\@law@printtitlefalse}%
 \\{n}{\global\@law@printcitefalse}%
 \\{l}{\message{\lowcase}}%
 \\{f}{\global\@law@longcitetrue}%
 \\{h}{\global\@law@intextcitetrue}%
 \\{b}{\global\@law@bibentrytrue\global\@law@longcitetrue}%
 \\{Z}{\gdef\@law@parsemacro##1{}%
       \let\@law@parse\@law@parsecomma}%
 \\{s}{\if@law@requiresubjects%
        \gdef\@law@parsemacro##1{\gdef\@law@citesubject{##1}}%
        \let\@law@parse\@law@parsecomma%
       \else%
        \def\@law@parsemacro##1{}%
       \fi}%
 \\{v}{\gdef\@law@parsemacro##1{\gdef\volno{##1}}%
    \let\@law@parse\@law@parsecomma}%
 \\{=}{\let\@law@parse\@law@parsebumequal}%
 \\{,}{}%
 \\{\end}{\let\@law@nomatch\relax\let\@law@parse\relax}%%
    \@law@nomatch}
```

\@law@parselastcheck
\@law@parsecomma
\@law@parselastcheck
\@law@parseplain

This is fun. Explanations will be added later.

```
\def\@law@parsecomma=#1,{%
  \@law@parsemacro{#1}\let\@law@parse\@law@parseplain%
  \@@parse}

\def\@law@parsebumequal#1,{\let\@law@parse\@law@parseplain\@@parse}

\def\@law@parseplain#1{%
  \let\@law@nomatch=\@law@plnomatch%
  \def\@law@optionitem{#1}\let\\=\@law@plone\@law@sourceoptionlist\@@parse}
```

\@@parse
\@law@parse

Now comes the fun bit. Control is passed back and forth between `\@law@parse`

and \@@parse, with the behaviour of those functions altering depending on what is chewed up into them. Amazingly enough, it works. Whatta concept.

```
\def\@@parse{\@law@parse}
\let\@law@parse\@law@parseplain
```

\@law@alone A copy of \@law@plone. Serves same function.

```
\def\@law@alone#1#2{\def\@law@allistitem{#1}%
  \ifx\@law@alitem\@law@allistitem #2%
  \let\\=\@law@alfinish\fi}
```

\@law@alnomatch We don't care what goes into this field, so there is no error message from the parser.

```
\def\@law@alfinish#1\@law@alnomatch{\@law@alnomatch}
\def\@law@alnomatch{%
  \global\@ltok@a=\expandafter{\@law@alitem}%
  \xdef\@law@temp{\the\@ltok@argtwo\the\@ltok@a}%
  \global\@ltok@argtwo=\expandafter{\@law@temp}}
```

\@law@allist The list of options is simple and expansible. The syntax of the last two lines is required if everything is to work as expected. The items in the help message should probably be drawn from here, to provide a hook for expansion if other styles want to tinker with the options to \cite.

```
\gdef\@law@allist{%
 \\{&}{\gdef\@law@alitem{\@law@delim%
                          \global\@law@multipagestrue%
                          \@law@ampersand}}%
 \\{=}{\gdef\@law@alitem{\@law@delim%
                             \@law@pinend%
                             \@law@pingroup%
                             \@law@pinstart%
                             \global\@law@multipagesfalse}
       \addtocounter{law@parapin}{1}}%
 \\{-}{\gdef\@law@alitem{\@law@delim
                          \global\@law@multipagestrue%
                          \@law@dash}}%
 \\{,}{\gdef\@law@alitem{}%
    \let\@law@parse\@law@alparsesavecomma}%
 \\{_}{\@law@get@ul}%
 \\{^}{\@law@get@carat}%
 \\{|}{\@law@get@bar}%
    \@law@alnomatch}
```

\@law@get@ul The purpose of the following is to place *active* versions of the special marking
\@law@getcarat characters on the stack. Note that the characters need not be active when read;
\@law@getbar they are replaced with active versions of the characters by this routine. Active characters are sent to the stack rather than control strings because they do not swallow following space (possibly not important), and because this made the whole thing easier to follow in the debugging process.

```
{\catcode`\_=13\catcode`\^=13\catcode`\|=13%
 \gdef\@law@get@ul{\gdef\@law@alitem{_}}%
 \gdef\@law@get@carat{\gdef\@law@alitem{^}}%
 \gdef\@law@get@bar{\gdef\@law@alitem{\@law@delim|}}%
}
\gdef\@law@finishargtwo{%
  \global\@ltok@a={\@law@pingroup\@law@pinstart}%
  \xdef\@law@temp{\the\@ltok@a\the\@ltok@argtwo}%
  \global\@ltok@argtwo=\expandafter{\@law@temp}}
```

`\@law@alparsesavecomma` `\@law@alparseplain` These parsers are switched in as appropriate. The first checks to see if the end of the string for parsing has been reached. The second cycles a comparison.

```
\gdef\@law@alparsesavecomma{\@ifnextchar,%
  {\gdef\@law@alitem{\@law@delim\@law@pinend}\@law@alnomatch%
   \expandafter\@law@finishargtwo%
   \@gobble}%
  {\gdef\@law@alitem{\@law@delim%
                     \@law@multipagestrue%
                     \@law@comma}%
   \@law@alnomatch%
   \let\@law@parse=\@law@alparseplain\@@parse}}
\def\@law@alparseplain#1{%
  \let\@law@nomatch=\@law@alnomatch%
  \gdef\@law@alitem{#1}\let\\=\@law@alone\@law@allist\@@parse}
```

`\@law@scanlist` This macro is defined and kept on the shelf as a definition of `\\` that checks through the list of paired aliases and associated text. If it finds a match, it gives the text one level of expansion, to fix the current definition of `_` and `^`. This allows one entry to serve both between citations, and at the start of a citation string. In addition to these macros, a `\@law@wordlist` macro must be defined in the `.cst` file used with CAMEL.

```
\def\@law@scanlist#1#2{\long\def\@law@temp{#1}%
  \@ltok@a={#2}%
  \ifx\@law@temp\@overword\xdef\@SBSword{{\noexpand\em \the\@ltok@a}}%
  \let\\=\@law@finish\fi}

\def\@law@wlnomatch{\message{^^J^^J%
 There is an unregistered inter-citation word (\@overword) on
 line no \the\inputlineno.  See the LexiTeX documentation
 for more information.  Please email fb@soas.ac.uk if
 you want a word added to the inter-citation list.}}
```

`\@law@checkins` This macro works with the `\@ifoverword` macro; it checks the contents of the `\@overword` macro defined during the leap, to see if it corresponds to anything in a list of aliases. The alias list could be defined as a set of prefixed macros (to speed things up), or as a single list macro (to conserve string space). I have adopted the latter strategy. If anyone feels strongly that speed is more important, or that somehow the integrity of TEX demands that macros be used, it's not much of a job to change things; be my guest.

```
{\catcode`\_=\active%
 \catcode`\^=\active%
\gdef\@law@checkins{%
  \def^##1{\lowercase{##1}}\def_##1{{\em{##1}}}%
  \let\@law@tempthree=\\%
  \let\@law@nomatch=\@law@wlnomatch%
  \let\\=\@law@scanlist\@law@wordlist%
  \let\\=\@law@tempthree}
\gdef\@law@checkpre{%
  \let\@law@tempthree=\\%
  \let^\relax\let_\@gobble%
  \let\@law@nomatch=\@law@wlnomatch%
  \let\\=\@law@scanlist\@law@wordlist%
  \let\\=\@law@tempthree}
}
```

`\source` The front end of the `\cite` command is very much *à la* typical LaTeX optional command definition stuff. We just carry things a little further because there are two possible options to the `\cite` command.

```
\gdef\source{%
  \if@law@requiresubjects%
    \gdef\@law@citesubject{}%
  \fi%
  \@law@printcitetrue%
  \@law@printauthortrue%
  \@law@printtitletrue%
  \@law@intextcitefalse%
  \@law@multipagesfalse%
  \setcounter{law@parapin}{0}%
  \setcounter{law@paracounter}{0}%
  \@ifnextchar[{\@lexicite}{\@lexicite[]}}
\def\@lexicite[#1]#2{\@ifnextcharcareful[{%
  \addtocounter{law@parapin}{1}\@@lexicite{#1}{#2}}%
  {\@@lexicitenobrace{#1}{#2}}}
```

Once we've gathered up any optional arguments, it's time to use the `\@ifoverword` command. If we find a `\cite` command after the next word, we stash the current cite, we check the word separating the two `\cite` commands against an internal list, return the result to `\@SBSword`, and push the raw citation details and the `@SBSword` bridge onto a temporary stack. Otherwise, we clear the `\@SBSword`, push the current cite details, dump the stack, and put the word we found, together with any separating space, back on the output stream.

```
\def\@ifnextcharcareful#1#2#3{%
  \gdef\@prenextcharspace{}%
  \let\reserved@e=#1\def\reserved@a{#2}\def\reserved@b{#3}\futurelet
    \reserved@c\@ifnchcareful}
\def\@ifnchcareful{\ifx \reserved@c \@sptoken \let\reserved@d\@xifnchcareful
      \else \ifx \reserved@c \reserved@e\let\reserved@d\reserved@a\else
          \let\reserved@d\reserved@b\fi
```

```
      \fi \reserved@d}
\def\:{\@xifnchcareful}
\expandafter\def\: {%
 \gdef\@prenextcharspace{ }\futurelet\reserved@c\@ifnch}
\def\@@lexicite#1#2[#3]{%
  \@ifoverword\cite{\@law@checkins\@@@lexicite%
      {#1}{#2}{#3}}%
    {\gdef\@SBSword{}\@@@lexicite%
      {#1}{#2}{#3}%
    \@law@citedump{}%
    \@preoverwordspace%
    \expandafter\@law@gobble%
    \@overword{} }}
\def\@@lexicitenobrace#1#2{%
  \@ifoverword\cite{\@law@checkins\@@@lexicite%
      {#1}{#2}{}}%
    {\gdef\@SBSword{}\@@@lexicite%
      {#1}{#2}{}%
    \@law@citedump{}%
    \@prenextcharspace%
    \expandafter\@law@gobble%
    \@overword{} }}
```

To push citation details, we use a token assignment with `\expandafter`, then an `\edef` of the register contents to get one level of expansion into a macro.

```
\def\@law@citestack{}%
\gdef\@SBSword{}%
\def\@@@lexicite#1#2#3{%
 \@ltok@a=\expandafter{\@law@citestack}%
 \@ltok@b={\@realcite{#1}{#2}{#3}}%
 \@ltok@c=\expandafter{\@SBSword}%
 \edef\@law@citestack{\the\@ltok@a\the\@ltok@b\the\@ltok@c}}
```

`\footnote` We need a flag to tell us whether we are in a footnote. This allows us to prevent footnote forcing if we're *already* in a footnote. LaTeX itself should probably take care of this, but for now we need to give it a helping hand by reinvoking the critical command with Apologies to the LaTeX3 team and everything, but I really truly do need these hooks.

Some thoughts on the tracking of previous citations. This is hardly transparent, and even I the author have trouble following it. Something for tidying up someday. Currentcite is set only at the top of a footnote. The idea apparently is that we need to remember the last cite in the last footnote always. Problems to watch here are (a) what happens for the very first footnote, and (b) what happens in the main text. There should probably be complete isolation of these context records for main text references and footnote references.

```
\long\def\@footnotetext#1{\insert\footins{%
 \@law@infoottrue%                % Hooks for citation manager
 \ifnum\the\c@citesinfoot=1\relax%             %
  \global\let\@law@lastcite\@law@currentcite%  %
```

```
\else%                                          %
 \gdef\@law@lastcite{\@dummy}%                  %
\fi%                                            %
\global\setcounter{citesinfoot}{0}% End of hooks
   \reset@font\footnotesize
   \interlinepenalty\interfootnotelinepenalty
   \splittopskip\footnotesep
   \splitmaxdepth \dp\strutbox \floatingpenalty \@MM
   \hsize\columnwidth \@parboxrestore
   \let\@tempa\protect
   \def\protect{\noexpand\protect\noexpand}%
   \edef\@currentlabel{\csname p@footnote\endcsname\@thefnmark}%
   \let\protect\@tempa
   \color@begingroup
     \normalcolor
     \@makefntext{%
       \rule\z@\footnotesep\ignorespaces#1%
   \@finalstrut\strutbox}
   \color@endgroup}}
```

We require a simple parser to allow multiple citation keys in the scope of a single `\source` command. We ignore any pinpoint given after the `\source` command; if you're using pinpoints, write individual `\source` commands into your text. This keeps things relatively clean and simple, for future linking to intelligent editors. The voiding of `\@ltok@argtwo` is going to be redundant if there are more than two keys in the scope, but what the heck, it's simple this way. Intervening punctuation is assumed to be the "default", which is signified by the `;` character. Note that any leading options will carry through to *all* of the citations in the key string! If you want to suppress the author, say, for only one citation, you'd better use a separate `\source` command for that citation, and link it to the string with an inter-word.

```
\def\@law@clparseplain#1,{%
 \@ifnextchar,%
   {\def\@law@temp{\@law@onerealcite{#1}}%
    \expandafter\@law@temp\@gobble}%
   {\@law@onerealcite{#1}%
   {\long\def\@overword{;}\@law@checkins%
    \@SBSword\gdef\@SBSword{}}%
   \@@parse}}
```

When citations are dumped, we need to decide what sort of delimiters they receive. If footnote forcing is in effect, we need to protect against the possibility that we're already in a footnote. The `\@citedump` command should be redefined for styles that bracket citation strings in, say, square brackets. The following code, the `\@law@printcite` command and the contents of the `lexicite.tex` file are the only chunks of the style that need modification to generate different—perhaps radically different—styles. The strategy used here to force footnotes could equally well be used to place the citation string in a `\marginpar`, in a floating box, on a facing page or what have you. So many possibilities, so little time ...

```
\def\@law@citedump{%
  \if@law@infoot%
    \begingroup\@law@citestack\endgroup%
  \else%
    \if@law@forcefootnotes%
      \footnote{\begingroup\@law@citestack\@law@gobble.\endgroup}%
    \else%
      \begingroup\@law@citestack\endgroup%
    \fi%
  \fi%
\def\@law@citestack{}}
```

We might as well have a couple of aliases for the `\cite` command.

```
\let\cite=\source
\let\lexicite=\source
```

The actual cites should be expanded only after it has been determined that the end of a citation string has been reached. The expansion sequence for individual citations begins here.

```
\def\@realcite#1#2#3{%
  \ifcat$#1$\else%
    {\let\@law@parse=\@law@parseplain\@law@parse #1,\end}%
  \fi%
  \if@law@requiresubjects%
   \@law@confirmsubject%
  \fi%
 \global\@ltok@argtwo{}%
 \ifcat$#3$\else%
   {\let\@law@parse=\@law@alparseplain\@law@parse #3,,}%
 \fi%
  {\let\@law@parse=\@law@clparseplain\@law@parse #2,,}%
  }
```

The `\@law@onerealcite` macro writes the citation key on the `.aux` file for onward use by BibTeX, and initiates citation expansion.

```
\def\@law@onerealcite#1{%
  \@ifundefined{@lnick@#1}%
   {\expandafter\def\csname @lnick@#1\endcsname{%
    {\small\bf\tt <undefined: #1>}
    \@latex@warning
     {Citation `#1' on page \thepage \space undefined}}}%
    {\relax}%
  \if@filesw%
   \immediate\write\@auxout{\string\citation{#1}}%
   \@ifundefined{@law@managerouthook}%
    \relax%
    {\immediate\write\@bibout{\@law@managerouthook{#1}}}%
  \fi%
\global\@law@firstuseofcitefalse%
\csname @lnick@#1\endcsname}
```

`\bibitem` The bibitem subcommands are redefined, since we don't need to export the nickname a second time when using CAMEL.

```
\def\@lbibitem[#1]{\item[\@biblabel{#1}\hfill]\ignorespaces}
\def\@bibitem{\item\ignorespaces}
```

`\bibliographymanager` This macro provides support for external bibliography managers. If a known manager is declared, the key is written on an ephemeral file with the name `camel.bib`. If the manager is then set up to replace keys with valid BIBTEX `.bib` entries, and is run over this file, the resulting file can be used as an input file for BIBTEX. Simple. Clean. LATEX.

```
\def\@law@subjectlist{}
\def\citationstyle#1{%
  \ifx\@law@subjectlist\empty%
    \def\@law@citesubject{all}
    \expandafter\gdef\csname @law@all@citelist\endcsname{}%
    \def\@law@subjectlist{\\{all}{}{\camelrefname}{}}%
  \fi%
  \if@filesw%
   \immediate\write\@auxout{\string\bibstyle{#1}}%
  \fi%
  \def\@law@savecat{\catcode`@}%
   \makeatletter%
   \input{#1.cst}%
   \input{#1.cit}%
   \catcode`\@=\@law@savecat%
  \@ifundefined{@law@bblfile}%
    {\relax}%
    {\begin{lexilist}
       \@input{\@law@bblfile}
     \end{lexilist}}}
\def\citationdata#1{%
  \def\bibliographymanager{\@camel@error{\string\bibliographymanager\space
     following \string\citationdata.^^J
    You can use only one of \string\citationdata\space and
    \string\bibliographymanager\space^^J at the start of a document
    or after a
    \string\printthebibliography\space command}\@ehc}%
  \if@filesw%
   \immediate\write\@auxout{\string\bibdata{#1}}%
  \fi%
  \@ifundefined{@law@wordlist}%
    {\gdef\@law@bblfile{\jobname.bbl}}%
    {\begin{lexilist}
       \@input {\jobname.bbl}
     \end{lexilist}}}
\def\@camel@error#1#2{%
   \GenericError{%
      \space\space\space\@spaces\@spaces\@spaces
   }{%
```

```
      Camel Error: #1%
    }{%
      See the Camel manual for explanation.%
    }{#2}%
}
\def\bibliographymanager#1{%
  \def\citationdata{\@camel@error{\string\citationdata\space
      following \string\bibliographymanager.^^J
    You can use only one of \string\citationdata\space and
    \string\bibliographymanager\space^^J at the start of a
    document or after a
    \string\printthebibliography\space command.}\@ehc}%
  \if@filesw%
   \immediate\write\@auxout{\string\bibdata{camel}}%
  \fi%
  \@ifundefined{@law@wordlist}%
    {\gdef\@law@bibfile{\jobname.bbl}}%
    {\begin{lexilist}%
        \@input {\jobname.bbl}
     \end{lexilist}}%
  \newwrite\@bibout%
  \immediate\openout\@bibout camel.bib%
  \gdef\@law@temp{#1}%
  \gdef\@law@temptwo{procite}%
  \ifx\@law@temp\@law@temptwo%
    \def\@law@managerouthook##1{(##1)}%
  \fi%
  \gdef\@law@temptwo{endnote}%
  \ifx\@law@temp\@law@temptwo%
    \def\@law@managerouthook##1{[##1]}%
  \fi%
  \gdef\@law@temptwo{papyrus}%
  \ifx\@law@temp\@law@temptwo%
    \def\@law@managerouthook##1{\%\%##1\%\%}%
  \fi%
  \gdef\@law@temptwo{referencemanager}%
  \ifx\@law@temp\@law@temptwo%
    \def\@law@managerouthook##1{\{##1\}}%
  \fi%
  \gdef\@law@temptwo{tib}%
  \ifx\@law@temp\@law@temptwo%
    \def\@law@managerouthook##1{[.##1.]}%
  \fi}
```

`\lexibib` This is used by LexiBib to declare citations in a `lexilist` environment. Use of a single macro permits `\theholding` and its friends to work more simply.

```
\def\lexibib#1{\csname new#1\endcsname}%
```

`\@law@showholding` `\@law@hideholding` `\holding` `\theholding` These macros allow database storage of itemized details of a case holding, and their recall using `\theholding`.

```
\def\@law@showholding#1{\def\holding{\@law@hideholding}
  \begin{enumerate}#1\end{enumerate}}
\def\@law@hideholding#1{}
\def\holding{\@law@hideholding}
\def\theholding{\def\lexibib##1##2##3##4##5##6##7{%
 \edef\@law@temp{\the\@ltok@nickname}
 \def\@law@temptwo{##2}
 \ifx\@law@temp\@law@temptwo\def\holding{\@law@showholding}
 \else\def\holding{\@law@hideholding}\fi}
\@input {\jobname.bbl}}
```

`\@law@showcomment` `\@law@hidecomment` `\comment` `\thecomment`

These macros allow database storage of comments on a case, and their recall using `\thecomment`.

```
\def\@law@showcomment#1{\def\comment{\@law@hidecomment}
  #1}
\def\@law@hidecomment#1{}
\def\comment{\@law@hidecomment}
\def\thecomment{\def\lexibib##1##2##3##4##5##6##7{%
 \edef\@law@temp{\the\@ltok@nickname}
 \def\@law@temptwo{##2}
 \ifx\@law@temp\@law@temptwo\def\comment{\@law@showcomment}
 \else\def\comment{\@law@hidecomment}\fi}
\@input {\jobname.bbl}}
```

`\@law@showquestions` `\@law@hidequestions` `\questions` `\thequestions`

These macros allow database storage of itemized questions on a case, and their recall using `\thequestions`.

```
%    \begin{macrocode}
\def\@law@showquestions#1{\def\questions{\@law@hidequestions}
  \begin{enumerate}#1\end{enumerate}}
\def\@law@hidequestions#1{}
\def\questions{\@law@hidequestions}
\def\thequestions{\def\lexibib##1##2##3##4##5##6##7{%
 \edef\@law@temp{\the\@ltok@nickname}
 \def\@law@temptwo{##2}
 \ifx\@law@temp\@law@temptwo\def\questions{\@law@showquestions}
 \else\def\questions{\@law@hidequestions}\fi}
\@input {\jobname.bbl}}
```

`\@law@showfacts` `\@law@hidefacts` `\facts` `\thefacts`

These macros allow database storage of the facts of a case, and their recall using `\thefacts`.

```
\def\@law@showfacts#1{\def\facts{\@law@hidefacts}
  #1}
\def\@law@hidefacts#1{}
\def\facts{\@law@hidefacts}
```

```
\def\thefacts{\def\lexibib##1##2##3##4##5##6##7{%
 \edef\@law@temp{\the\@ltok@nickname}
 \def\@law@temptwo{##2}
 \ifx\@law@temp\@law@temptwo\def\facts{\@law@showfacts}
 \else\def\facts{\@law@hidefacts}\fi}
\@input {\jobname.bbl}}
```

We need to initialize the list macro used as a workspace by the list macro handlers.

```
\def\@law@templistmacro{}
\def\@law@tempplistmacro{}
```

These macros are used for `\if`-branching in various formatting routines.

```
\def\@law@case{case}
\def\@law@statute{statute}
\def\@law@article{article}
\def\@law@book{book}
```

3.2 Main macros

3.2.1 Utility macros

This section contains larger macros that perform major tasks and smaller bits of code that are complex in their operation.

`\@law@unstashparas` When this is implemented, the above chain of `futurelet` macros will be linked to this `\@law@unstashparas` routine, which fetches any parallel items, and then (in `\@law@setup`) performs the restash of the citation if necessary.

`\@law@clean` `\@law@cleanup` This is a rough set of routines; if anyone has a suggestion on how to more elegantly clean most of LaTeX's control sequences for export, I would be most grateful to hear from them. The effect of these routines is to convert control sequences that might be sent to the output files into harmless strings. This is necessary because the file cannot be written at the time `\@law@cleanup` is called; the page references would be incorrect. The definitions are local; bracketing the clean command and its arguments will leave the cleaned text in the macro, but restore command strings to operation when LaTeX passes out of the current group. The `\ifcat` condition at the start (this method of token register checking was suggested by Bernd Raichle—who scolded me for bothering the LaTeX3 list with this question!) is meant to catch empty strings fed to the routine in the fastest possible way. Without any checking, things can blow up. The use of `\@law@barnil` (somewhere in the middle) makes the vertical-bar character turn wimpish and disappear, which is what we want. Otherwise it prints weird stuff when it is fed back in through the external table.

```
{\catcode`\_=13\catcode`\^=13\catcode`\|=13%
\gdef\@law@clean#1#2{%
 \def\protect##1{\string##1\space}%
 \ifcat$\the#1$%
   \edef#2{}%
  \else%
 \def^{\string^}%
```

```
\def_{\string_}%
\@law@barnil%
\def\exclaim{\string\exclaim}%
\def\hbox{\string\hbox}%
\def\&{\string\&}\def\%{\string\%}%
\def\hskip{\string\hskip}%
\def\jintercharskip{\string\jintercharskip}%
\def\char{\string\char}%
\def~{\string~}\def\/{\string\/}%
\def\\{\string\\}\def\ {\string\ }\def\sc{\string\sc\space}%
\def\rm{\string\rm\space}\def\bf{\string\bf\space}%
\def\em{\string\em\space}%
\def\={\string\=}%
\def\‘{\string\‘}\def\’{\string\’}%
\def\^{\string\^}\def\"{\string\"}\def\~{\string\~}%
\def\.{\string\.}\def\u{\string\u}\def\v{\string\v}%
\def\H{\string\H\space}\def\t{\string\t\space}%
\def\c{\string\c\space}%
\def\d{\string\d\space}\def\b{\string\b\space}%
\def\oe{\string\oe\space}%
\def\ae{\string\ae\space}\def\aa{\string\aa\space}%
\def\o{\string\o\space}%
\def\l{\string\l\space}\def\ss{\string\ss\space}%
\def\OE{\string\OE\space}\def\AE{\string\AE\space}%
\def\AA{\string\AA\space}%
\def\O{\string\O\space}\def\L{\string\L\space}%
\def\dag{\string\dag\space}\def\ddag{\string\ddag\space}%
\def\S{\string\S\space}%
\def\P{\string\P\space}%
\def\TeX{\string\TeX\space}\def\LaTeX{\string\LaTeX\space}%
\def\LexiTeX{\string\LexiTeX\space}%
\def\BibTeX{\string\BibTeX\space}\def\-{\string\-}%
\xdef\@law@temp{\the#1}%
\xdef#2{\expandafter\expandafter\expandafter\@gobble%
\expandafter\string\csname\@law@temp\endcsname}%
\fi}
}
```

This routine runs the cleaning routine on every token register that we want to send to the external files. The result is stored in a macro, since we need the registers for the next citation, and because it will be safe to fully expand the contents after cleaning.

```
\def\@law@cleanup{%
\@law@clean\@ltok@authormainface\@law@authormainfacetemp%
\@law@clean\@ltok@authoroptionface\@law@authoroptionfacetemp%
\@law@clean\@ltok@author\@law@authortemp%
\@law@clean\@ltok@titlemainface\@law@titlemainfacetemp%
\@law@clean\@ltok@titleoptionface\@law@titleoptionfacetemp%
\@law@clean\@ltok@argtwo\@law@argtwotemp%
\@law@clean\@ltok@name\@law@nametemp%
```

```
\@law@clean\@ltok@citepage\@law@citepagetemp%
\@law@clean\@ltok@citefirstmainface\@law@citefirstmainfacetemp%
\@law@clean\@ltok@citefirstoptionface\@law@citefirstoptionfacetemp%
\@law@clean\@ltok@citefirst\@law@citefirsttemp%
\@law@clean\@ltok@conetop\@law@conetoptemp%
\@law@clean\@ltok@citelast\@law@citelasttemp%
\@law@clean\@ltok@ptop\@law@ptoptemp}
```

3.2.2 Declaration of citation types

This section contains macros used in creating the citation declaration macros actually entered in the document by the user.

`@newcite` and its friends The following macros are used to define the macro suites that relate to particular citation types. These are normally invoked by entries in the `lexicite.tex` file, but the `\newcitestyle` macro is accessible within the document as well. The `\newcitestyle` macro expects the following arguments:

1. the nickname of the citation style to be created;
2. a list of typeface and cite type options, in the syntax required by `\@newciteoptions`;
3. a list of cite bridges, in the syntax required by `\@newcitebridges`;
4. an integer giving the number of arguments the finished cite-generation macro will accept; and
5. an argument consisting of a balanced list of six arguments indicating which of the six possible CAMEL arguments will be used.

The last two of these are fed to `\newcommand` as formatting arguments in the creation of citation style macros. In operation, the token register assignments performed by `\@newcitebridges` and`\@law@parsefaces` are memorized by freezing them in macros whose names are derived from the name of the citation style. The freezing operation is carried out by a list of token assignments within the storage macros. The macros are `\xdef`ed, but the expansion of everything except the existing contents of the registers (inserted at one level of expansion using `\the`) is prevented with `\noexpand`s. The macro containing the faces is created inside `\newcitestyle` itself to save an argument position in the definition of `\@law@parsefaces`; we've used up seven, and there are only two to go. This could be done with list macros as well, but this method probably runs a little faster because it does the job directly. The trade-off is that the ⟨*cite type name*⟩`faces` macros contain a lot of non-informative text (the names of the token registers); but there are not many citation types, so it is probably worth the cost. in memory.

Note that as of this release, citation declarations defined with `\newcitestyle` *always* begin with `\new`. This will help simplify the integration of BIBTEX and CAMEL.

```
\def\newcitestyle#1#2#3{%
```

```
\@ifundefined{@law@citeload}%
 {\def\@law@citeload{Loading citation classes: #1 ... }}%
 {\message{\@law@citeload#1}\def\@law@citeload{...\space}}%
 \@newcitebridges#1#3%
  \expandafter\@law@parsefaces#2%
  \expandafter\xdef\csname#1faces\endcsname{%
   \noexpand\global\noexpand\@ltok@authormainface\noexpand{%
    \the\@ltok@authormainface\noexpand}%
   \noexpand\global\noexpand\@ltok@authoroptionface\noexpand{%
    \the\@ltok@authoroptionface\noexpand}%
   \noexpand\global\noexpand\@ltok@titlemainface\noexpand{%
    \the\@ltok@titlemainface\noexpand}%
   \noexpand\global\noexpand\@ltok@titleoptionface\noexpand{%
    \the\@ltok@titleoptionface\noexpand}%
   \noexpand\global\noexpand\@ltok@citefirstmainface\noexpand{%
    \the\@ltok@citefirstmainface\noexpand}%
   \noexpand\global\noexpand\@ltok@citefirstoptionface\noexpand{%
    \the\@ltok@citefirstoptionface\noexpand}%
   \noexpand\global\noexpand\@ltok@citetype\noexpand{%
    \the\@ltok@citetype\noexpand}}%
 \expandafter\gdef\csname new#1\endcsname##1##2##3##4##5##6##7%
  {\@newcite{#1}{##1}{##2}{##3}{##4}{##5}{##6}{##7}}}
```

The following are parsing routines that are called by `\newcitestyle`. In `\@law@parsefaces`, the seven arguments are the string of letters in the second argument of `\newcitestyle`. This function has been drastically simplified over the first release. I should have done it this way in the first place and saved myself a lot of aggravation. The code should run a lot faster to boot.

```
\def\@law@parsefaces#1#2#3#4#5#6#7{%
 \@law@parseoneoption\@ltok@authormainface#1%
 \@law@parseoneoption\@ltok@authoroptionface#2%
 \@law@parseoneoption\@ltok@titlemainface#3%
 \@law@parseoneoption\@ltok@titleoptionface#4%
 \@law@parseoneoption\@ltok@citefirstmainface#5%
 \@law@parseoneoption\@ltok@citefirstoptionface#6%
 \@law@parselastoption\@ltok@citetype#7}
%
\def\@law@parseoneoption#1#2{%
  \if#2s\global#1{\sc}\fi%
  \if#2i\global#1{\em}\fi%
  \if#2b\global#1{\bf}\fi%
  \if#2r\global#1{\rm}\fi%
  \if#2S\global#1{\sc}\fi%
  \if#2I\global#1{\em}\fi%
  \if#2B\global#1{\bf}\fi%
  \if#2R\global#1{\rm}\fi}
%
\def\@law@parselastoption#1#2{%
  \if#2a\global#1{article}\fi%
  \if#2b\global#1{book}\fi%
```

```
  \if#2c\global#1{case}\fi%
  \if#2s\global#1{statute}\fi%
  \if#2A\global#1{article}\fi%
  \if#2B\global#1{book}\fi%
  \if#2C\global#1{case}\fi%
  \if#2S\global#1{statute}\fi}
```

Bridges are stored literally, without parsing, so it's easy to stash them directly in a single macro that is fed the arguments. The assignments here are stored by the same method outlined in the description of `\newcitestyle`, above.

```
\newtoks\@ltok@c%
\newtoks\@ltok@d%
\newtoks\@ltok@e%
\newtoks\@ltok@f%
\newtoks\@ltok@g%
\newtoks\@ltok@h%
\newtoks\@ltok@i%
\newtoks\@ltok@j%
\def\@newcitebridges#1[a]#2[t]#3[c]#4[p]#5(pl)#6[rp]#7[e]:[id]#8(pl)#9[xrf]{%
\@ltok@c{#2}%
\@ltok@d{#3}%
\@ltok@e{#4}%
\@ltok@f{#5}%
\@ltok@g{#6}%
\@ltok@h{#7}%
\@ltok@i{#8}%
\@ltok@j{#9}%
  \expandafter\xdef\csname#1bridges\endcsname{%
   \noexpand\global\noexpand\@ltok@atot\noexpand{\the\@ltok@c\noexpand}%
   \noexpand\global\noexpand\@ltok@ttocone\noexpand{\the\@ltok@d\noexpand}%
   \noexpand\global\noexpand\@ltok@conetop\noexpand{\the\@ltok@e\noexpand}%
   \noexpand\global\noexpand\@ltok@conetopplural\noexpand{\the\@ltok@g\noexpand}%
   \noexpand\global\noexpand\@ltok@ptop\noexpand{\the\@ltok@f\noexpand}%
   \noexpand\global\noexpand\@ltok@ptoctwo\noexpand{\the\@ltok@h\noexpand}%
   \noexpand\global\noexpand\@ltok@atbridge\noexpand{\the\@ltok@i\noexpand}%
   \global\@ltok@atbridgeplural\noexpand{\the\@ltok@j\noexpand}}}%
```

3.2.3 Declaration of citation nicknames

The following macros are called by the macros generated by the `\newcitestyle` macro and its arguments. Their effect is to create nickname macros which can then be called by the user with the `\source` command, with the nickname as a single argument in braces. This macro is called by `\new`⟨*citation type name*⟩. While this macro always takes seven arguments, some of these may be masked off from the user, if fewer than the full possible six arguments were called for in the final argument fed to `\newcitestyle` in creating the particular `\new`⟨*citation type name*⟩ macro that is doing the calling. The arguments are there, the user just can't put anything into them.

The `\@law@authortracing` macro is actually just a temporary macro; I gave it

a descriptive name because it was a struggle to keep my head around what it does as I was programming this. What happens is that a token assignment of the author's name followed by a trip to the cleaning routine is used to store a harmless string consisting of the entire contents of the author's name field in the `\@law@authortracing` macro. Then this *name* is defined as a macro, expanding to `1` or `2`, depending on whether it has been defined once already. The macro may look something like `\Jayeff Huggins~Jr.`. Drastic, but effective. Try doing that with a WordPerfect macro ...

```
\newif\if@law@specialbridges%
\def\@newcite#1#2#3#4#5#6#7#8{%
\message{=}%
  {\@ltok@a={#4}\@law@clean\@ltok@a\@law@temp%
\expandafter\ifx\csname\@law@temp\endcsname\relax%
\expandafter\expandafter\expandafter\xdef\expandafter%
 \csname\@law@temp\endcsname{1}%
\else%
  \expandafter\expandafter\expandafter\xdef\expandafter%
  \csname\@law@temp\endcsname{2}%
\fi}%
```

The `\@law@stasheverything` macro creates the list macro that can be used to retrieve all citation details at a later point in the document. `\@law@makecitenick` creates the short macro that knows how to unstash the stored information. Note that this is where things end if we are inside the `lexilist` environment.[14]

```
\def\@law@templistmacro{}%
\def\@law@tempplistmacro{}%
\setcounter{law@paranormal}{0}
  \ifcat$\the\@ltok@useratbridgeplural$%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
  \else%
   \@law@addtoken\@ltok@userconetop\tocitelist%
   \@law@addtoken\@ltok@userconetopplural\tocitelist%
```

[14] The previous release used a `\@law@stashinfo` macro that took nine arguments. The last two were just toggles, in effect, for the creation of a long- and a short-form nickname macro. Passing the information in this way required the definition of two separate invocation macros, one for full-form citing and one for short-form citing. This was messy. We now do all of this with true toggles that change the behaviour of a single set of routines. This means:

- We don't need two macros; and
- The stash routines can be included in `\@newcite` itself—information needn't be passed through macro arguments at all, which helps speed things up.

We have to start by emptying the contents of the temporary list macro; a beta version didn't do this, and loading cites took an amazingly long time. I had problems with "`TeX capacity exceeded`" messages, and `\tracingmacros=2` showed that every cite was a stack containing the desired cite—and every preceding citation as well!

```
    \@law@addtoken\@ltok@userptop\tocitelist%
    \@law@addtoken\@ltok@userptoctwo\tocitelist%
    \@law@addtoken\@ltok@useratbridge\tocitelist%
    \@law@addtoken\@ltok@useratbridgeplural\tocitelist%
   \fi%
\@law@addargument{#1}\tocitelist%
\@law@addtoken\@ltok@bigsourcepage\tocitelist%  (not yet implemented)
\@law@addtoken\@ltok@bigsourcecite\tocitelist%  (not yet implemented)
\@law@addtoken\@ltok@hereinafter\tocitelist%    (not yet implemented)
```

Two fields are made nil during stash; they will be filled when the macro is first unpacked.

```
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
\@law@addtoken\@ltok@citetype\tocitelist%     cite type
\@law@addargument{#8}\tocitelist%             cite last part
\@law@addargument{#7}\tocitelist%             cite page
\@law@addargument{#6}\tocitelist%             cite first part
\@law@addargument{#5}\tocitelist%             name of work
\@law@addargument{#4}\tocitelist%             author
\expandafter\gdef\csname @law@#2@lab\endcsname{#3}% label
\@law@addargument{#2}\tocitelist%             nickname
\def\@law@tempplistmacro{}%
\@law@paracheckone{#2}}%
```

`\@law@paracheckone`
`\@law@parachecktwo`
`\@law@getpara`

The following three macros look ahead (after the initial arguments to a cite declaration have been digested but *before* the cite information has been stored to the list macro) for a `=` sign, which signals parallel citation details. If a `=` sign is found, a further check is made for a `[`, which signals a set of special bridges for the upcoming parallel citation. Note the use of two `=` signs in braces as the first argument to `\@ifnextchar`. This is required because of the internal syntax of the `\@ifnextchar` macro.

```
\def\@law@paracheckone#1{%
 \def\@law@temp{{#1}}%
 \@ifnextchar{=}%
 {\expandafter\expandafter\expandafter\@law@parachecktwo%
   \expandafter\@law@temp\@gobble}%
  {\@@newcite{#1}}}%

\def\@law@parachecktwo#1{\@ifnextchar[%
  {\@law@getpara{#1}}%
  {\@law@getpara{#1}[{}{}{}{}{}]}}
```

The following add a single set of parallel citation details into a list macro used as a holding area for this purpose.

```
\def\@law@getpara#1[#2#3#4#5#6]#7#8#9{%
  \addtocounter{law@parapin}{1}
  \def\@law@temp{{#1}}%
  \@law@addpargument{{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}\tolist%
  \@ifnextchar{=}%
```

```
799    {\expandafter\expandafter\expandafter%
800     \@law@parachecktwo\expandafter\@law@temp\@gobble}%
801    {\@@newcite{#1}}}
802 \def\@@newcite#1{%
```

A counter indicating how many parallel citations are stored for the current citation, and the list macro of parallel citation details built by the routines above is stored as a single item in the general list macro of citation details.

```
803 \@law@addtoken\c@law@paranormal\tocitelist
804 \@ltok@a\expandafter{\@law@tempplistmacro}%
805 \@law@addtoken\@ltok@a\tocitelist%
806 \expandafter\global\expandafter\let\csname @ldata@#1\endcsname=\@law@templistmacro%
807 \@ltok@userconetop{}%
808 \@ltok@userconetopplural{}%
809 \@ltok@userptop{}%
810 \@ltok@userptoctwo{}%
811 \@ltok@useratbridge{}%
812 \@ltok@useratbridgeplural{}%
```

Having stashed everything, we're ready to make the nickname macro itself, and print the long form of the cite. We set `\@law@temp` equal to the nickname macro, and expand after the condition so that arguments can be examined—otherwise the first thing the nickname sees will be the `\fi`.

```
813 \@law@makecitenick{#1}%
814   \if@law@listinputting\def\@law@nick{}\else%
815   \def\@law@nick{\source[f]{#1}}\fi\@law@nick}%
```

This macro adds one argument item `#1` to the end of list macro `\@law@templistmacro`. This and the other list handlers are modelled on the examples in Knuth's TEXbook.[15]+378 This and the other macros in this set do their work on one list macro, `\@law@templistmacro`, in order to avoid the need for lots of `\expandafter`s.

```
816 \def\@law@addargument#1\tocitelist{%
817   \@law@leftappendargument#1\to\@law@templistmacro}
818 \long\def\@law@leftappendargument#1\to#2{\@ltok@a={\\{#1}}%
819   \@ltok@b=\expandafter{#2}%
820   \xdef#2{\the\@ltok@a\the\@ltok@b}}
```

This macro adds the contents of one token register to the temporary list macro.

```
821 \def\@law@addtoken#1\tocitelist{\@law@leftappendtoken#1\to\@law@templistmacro}
822 \long\def\@law@leftappendtoken#1\to#2{%
823   \xdef\@law@temp{\noexpand\\\noexpand{\the#1\noexpand}}%
824    \@ltok@a=\expandafter{\@law@temp}%
825   \@ltok@b=\expandafter{#2}%
826   \xdef#2{\the\@ltok@a\the\@ltok@b}}
```

This macro cuts one token-register item `#1` from the end of list macro `\@law@templistmacro`. The lopoff portion is almost straight out of the TeXbook, except that it stores the lopped element in a token register instead of a macro.

[15] D. Knuth, *supra* note 1.

```
\def\@law@cut#1\fromcitelist{\@law@lop\@law@templistmacro\to#1}
\def\@law@lop#1\to#2{\expandafter\@law@lopoff#1\@law@lopoff#1#2}%
\long\def\@law@lopoff\\#1#2\@law@lopoff#3#4{\global#4={#1}\gdef#3{#2}}
\def\@law@pincut#1\frompinlist{\@law@pinlop\@law@argtwolist\to#1}
\def\@law@pinlop#1\to#2{\expandafter\@law@pinlopoff#1\@law@pinlopoff#1#2}%
\long\def\@law@pinlopoff\@law@pingroup\@law@pinstart#1\@law@pinend#2\@law@pinlopoff#3#4{\global#
```

`\@law@addpargument` `\@law@addptoken` `\@law@pcut`

The following nine macros are used to manipulate a list macro containing parallel citation details, and sets of token registers into which the list is extracted.

```
\def\@law@fetchparas{%
  \ifx\@law@tempplistmacro\empty%
    \gdef\@law@tempplist{{}{}{}{}{}{}{}{}}%
  \else%
    \@law@pcut\@law@tempplist\fromlist%
  \fi%
  \expandafter\@law@extractparas\@law@tempplist}
\def\@law@extractparas#1#2#3#4#5#6#7#8#9{%
  \ifcat$\the\@ltok@useratbridge$%
  \else%
    \@ltok@userconetop={#1}%
    \@ltok@userconetopplural={#2}%
    \@ltok@userptop={#3}%
    \@ltok@useratbridge={#4}%
    \@ltok@useratbridgeplural={#5}%
  \fi%
  \global\@ltok@usercitefirst={#6}%
  \global\@ltok@usercitepage={#7}%
  \global\@ltok@usercitelast={#8}}
\def\@law@shiftparas{%
  \ifcat$\the\@ltok@useratbridge$%
  \else%
    \global\@ltok@conetop\@ltok@userconetop%
    \global\@ltok@conetopplural\@ltok@userconetopplural%
    \global\@ltok@ptop\@ltok@userptop%
    \global\@ltok@ptoctwo\@ltok@userptoctwo%
    \global\@ltok@atbridge\@ltok@useratbridge%
    \global\@ltok@atbridgeplural\@ltok@useratbridgeplural%
  \fi%
    \global\@ltok@citefirst\@ltok@usercitefirst%
    \global\@ltok@citepage\@ltok@usercitepage%
    \global\@ltok@citelast\@ltok@usercitelast}
\def\@law@addpargument#1\tolist{%
  \addtocounter{law@paranormal}{1}%
  \@law@rightappendpargument#1\to\@law@tempplistmacro}
\long\def\@law@rightappendpargument#1\to#2{\@ltok@a={\\{#1}}%
  \@ltok@b=\expandafter{#2}%
  \xdef#2{\the\@ltok@b\the\@ltok@a}}
\def\@law@addptoken#1\tolist{\@law@leftappendptoken#1\to\@law@tempplistmacro}
\long\def\@law@leftappendptoken#1\to#2{%
```

```
  \xdef\@law@temp{\noexpand\\\noexpand{\the#1\noexpand}}%
   \@ltok@a=\expandafter{\@law@temp}%
  \@ltok@b=\expandafter{#2}%
  \xdef#2{\the\@ltok@a\the\@ltok@b}}
\def\@law@pcut#1\fromlist{\@law@plop\@law@tempplistmacro\to#1}
\def\@law@plop#1\to#2{\expandafter\@law@plopoff#1\@law@plopoff#1#2}%
\long\def\@law@plopoff\\#1#2\@law@plopoff#3#4{\gdef#4{#1}\gdef#3{#2}}
```

This macro turns off the gobbling feature (it may be turned on during execution, if *Id.* is used), erases any pinpoint reference information that is hanging around, and if we are in a footnote, it increments the counter that was set to zero at the start of the footnote. It then unpacks all of the info for the cite whose nickname is given in `#1`, and puts it into token registers for further processing. After it unpacks the list macro `\@ldata@`⟨*nickname*⟩, it unpacks the formatting details appropriate to the style in which it was declared. After the list is unpacked, *if* there is no record of a footnote or page number, then this is the first printing of the cite. The note/page and footnote/page number details need to be stored, and then everything has to be *repacked* for future reference under the appropriate list macro name. The list macro strategy allows on-the-fly editing of macros which adds greatly to the flexibility of the style.
Note the `parabeta` guard below. When we turn on support for parallel citations, we will need to do the restash work later, after checking for any attached items.

```
\def\@law@unstasheverything#1{%
\gdef\@law@gobble{}%
\if@law@infoot%
  \addtocounter{citesinfoot}{1}%
\fi%
\expandafter\let\expandafter\@law@templistmacro\csname @ldata@#1\endcsname%
\global\c@law@footnote\c@footnote%
\@law@cut\@ltok@plistmacro\fromcitelist%
\xdef\@law@tempplistmacro{\the\@ltok@plistmacro}%
\@law@cut\@ltok@a\fromcitelist%
\expandafter\c@law@paranormal\the\@ltok@a%
\@law@cut\@ltok@nickname\fromcitelist%
\@law@cut\@ltok@author\fromcitelist%
\@law@cut\@ltok@name\fromcitelist%
\@law@cut\@ltok@citefirst\fromcitelist%
\@law@cut\@ltok@citepage\fromcitelist%
\@law@cut\@ltok@citelast\fromcitelist%
\@law@cut\@ltok@citetype\fromcitelist%
\@law@cut\@ltok@whereitsat\fromcitelist%
\@law@cut\@ltok@pageorfootno\fromcitelist%
\@law@cut\@ltok@hereinafter\fromcitelist%
\@law@cut\@ltok@bigsourcecite\fromcitelist%(not yet implemented)
\@law@cut\@ltok@bigsourcepage\fromcitelist%(not yet implemented)
\@law@cut\@ltok@stylename\fromcitelist%
\csname \the\@ltok@stylename bridges\endcsname%
\csname \the\@ltok@stylename faces\endcsname%
```

The following lines check one of the special bridge fields for content. If it is empty, we leave the bridges alone. Otherwise, we replace the default bridges with those selected by the user for this citation.

```
\@law@cut\@ltok@a\fromcitelist%
  \ifcat$\the\@ltok@a$%
  \else%
   \@ltok@atbridgeplural\@ltok@a%
   \@law@cut\@ltok@atbridge\fromcitelist%
   \@law@cut\@ltok@ptoctwo\fromcitelist%
   \@law@cut\@ltok@ptop\fromcitelist%
   \@law@cut\@ltok@conetopplural\fromcitelist%
   \@law@cut\@ltok@conetop\fromcitelist%
   \@law@specialbridgestrue%
  \fi%
\ifcat$\the\@ltok@whereitsat$%
  \if@law@requiresubjects%
   \@ifundefined{@law@\@law@citesubject @citelist}%
    {\expandafter\gdef\csname @law@\@law@citesubject @citelist\endcsname{}}%
    {\relax}%
  \fi%
  \@law@maybeaddcitesubject
  \global\@law@firstuseofcitetrue%
  \if@law@infoot%
   \global\@ltok@whereitsat\@ltok@infoot%
   \global\@ltok@pageorfootno=\expandafter{\the\c@law@footnote}%
  \else%
   \global\@ltok@whereitsat\@ltok@onpage%
   \global\@ltok@pageorfootno=\expandafter{\the\c@page}%
  \fi%
  \if@law@specialbridges%
   \@law@addtoken\@ltok@conetop\tocitelist%
   \@law@addtoken\@ltok@conetopplural\tocitelist%
   \@law@addtoken\@ltok@ptop\tocitelist%
   \@law@addtoken\@ltok@ptoctwo\tocitelist%
   \@law@addtoken\@ltok@atbridge\tocitelist%
   \@law@addtoken\@ltok@atbridgeplural\tocitelist%
  \else%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
   \@law@addargument{}\tocitelist%
  \fi%
\@law@addtoken\@ltok@stylename\tocitelist%                style name
\@law@addtoken\@ltok@bigsourcepage\tocitelist%(not yet implemented)
\@law@addtoken\@ltok@bigsourcecite\tocitelist%(not yet implemented)
\@law@addtoken\@ltok@hereinafter\tocitelist%  (not yet implemented)
\@law@addtoken\@ltok@pageorfootno\tocitelist%   (1)(b) stash the footnote bridge
```

```
\@law@addtoken\@ltok@whereitsat\tocitelist%     (2)(b) footnote number
\@law@addtoken\@ltok@citetype\tocitelist%   cite type
\@law@addtoken\@ltok@citelast\tocitelist%   cite last part
\@law@addtoken\@ltok@citepage\tocitelist%   cite page
\@law@addtoken\@ltok@citefirst\tocitelist%   cite first part
\@law@addtoken\@ltok@name\tocitelist%   name of work
\@law@addtoken\@ltok@author\tocitelist%   author
\@law@addtoken\@ltok@nickname\tocitelist%   nickname
\@law@addtoken\c@law@paranormal\tocitelist% parallels counter
\@law@addtoken\@ltok@plistmacro\tocitelist% parallels
\global\expandafter\let\csname @ldata@#1\endcsname\@law@templistmacro%
\fi%
}
```

3.2.4 Calling citation nicknames

The nickname macro defined by this routine (`\@lnick@`⟨*nickname*⟩), is immensely more compact than in the first release. `\@law@argscheck` eventually leads to `\@law@setup` after a series of checks for optional arguments. For the longest time I used a `\the` statement for the contents of `\@law@argscheck`; `\noexpand` works just as well and is much simpler to arrange.

```
\def\@law@makecitenick#1{%
    \expandafter\xdef\csname @lnick@#1\endcsname{%
    \noexpand\@law@unstasheverything\noexpand{#1\noexpand}%
  \noexpand\@law@setup}}
```

`\@law@newcitefirst` The following code will be used to kick off the existing `\@newcite` routine when hereinafter-style references come to be supported.

```
⟨*hereinafters⟩
\def\@law@newcitefirst{\@ifnextchar[{\catcode`\[=??
\catcode`\]=??
\expandafter\@law@newcitesecond\@law@grabhereinafter}
{\@law@newcite}}
\def\@law@grabhereinafter#1{\@ltok@hereinafter={#1}
⟨/hereinafters⟩
```

3.2.5 Table writes and preliminary formatting

Once the printing information has been stashed, the `\@lnick@`⟨*nickname*⟩ macro can be used to extract and print it. After extracting the information and storing it in token registers, this looks forward to gather any pinpointing argument that has been appended to the macro in the text. Once all the the necessary information has been collected, the `\@law@setup` macro tidies up the formatting of the cite and writes it to the export file, if the creation of tables for that type of citation has been toggled on. It then invokes the print routine.

This macro assumes that all citation information stored for the cite has already been retrieved to appropriate token registers. The redefinition of the backslash character here allows it to be used to select the alternate typeface within a citation

argument. The alternate typeface can be changed for all documents by editing the relevant `.cst` file. Grouping braces enclosing all operations within `\@law@setup` mean than weird stuff turned on the sole purpose of exporting to tables and printing the citation will turn itself off when the print routine is finished. A routine (`\@law@justabovecheck`) is run to figure out what the citation context is—how a cross-reference should be formatted—and once it's found out what it needs to know about the last citation, information on the current citation is tucked away for reference by this the context-checking routine next time around. A series of checks is performed so that superfluous bridges for which there is no corresponding argument can be erased. Then a file write is performed to the appropriate export file, if creation of that table has been toggled on. And finally the print routine is called. The `\@law@gobble` that follows the end of the group is used to eat a superfluous period if necessary. It is brought to life, if appropriate, by the print routine. Note the special catcode of the `|` character during this definition.

```
{\catcode`\|=13%
\long\gdef\@law@setup{%
```

The following operations restash the citation details if necessary; by this point we will have grabbed any parallel citation details that we needed. The backslash character is redefined as a self-resetting font-switching macro.

```
{\def\@law@firstslash{\begingroup\def\\{\@law@secondslash}%
   \the\ltokspecialface}%
  \def\@law@secondslash{\endgroup\def\\{\@law@firstslash}}%
  \def\\{\@law@firstslash}%
```

This carries out the necessary checks on the context of the current citation.

```
\@law@justabovecheck%
```

Now that the `\@law@justabovecheck` macro has been run, it is time to set up for the next comparison. Take note that this source was cited, and of any specific page reference it contains, IF we are in a footnote

```
\if@law@infoot%
   \xdef\@law@currentcite{\the\@ltok@author\the\@ltok@name\the\@ltok@citefirst%
   \the\@ltok@citepage\the\@ltok@citelast}%
   \xdef\@law@lastpage{\the\@ltok@argtwo}%
\fi%
  \begingroup%
  \@law@tidybridges%
  \ifx\@law@citesubject\empty%
   \relax%
  \else%
    \if@law@bibentry%
    \else%
      \ifnum\the\c@law@paranormal=0\relax%
        \csname write@one@\@law@citesubject @entry\endcsname%
      \fi%
    \fi%
  \fi%
  \endgroup%
```

```
  \ifcat$\the\@ltok@argtwo$%
  \else%
    \xdef\@law@argtwolist{\the\@ltok@argtwo}%
    \@law@pincut\@ltok@argtwo\frompinlist%
  \fi%
\@law@print}}}
\def\@law@tidybridges{%
```

A number of further tidying-up operations are appropriate regardless of the form in which the citation will be printed. If certain elements are missing, their related bridges must be erased if the citation is not going to look awful.

```
  \ifcat$\the\@ltok@author$%
   \ifcat$\the\@ltok@citefirst$%
    \global\@ltok@ttocone{}%
   \fi%
   \ifcat$\the\@ltok@name$%
     \global\@ltok@ttocone{}%
   \fi%
   \global\@ltok@atot{}%
  \fi%
  \xdef\@law@temp{\the\@ltok@citelast}%
  \ifx\@law@temp\empty%
   \global\@ltok@ptoctwo{}%
  \fi%
  \def\@law@temptwo{()}%
  \ifx\@law@temp\@law@temptwo%
   \global\@ltok@ptoctwo{}%
   \global\@ltok@citelast{}%
  \fi%
   \ifcat$\the\@ltok@citepage$%
     \global\@ltok@ptop{}%
   \fi%
   \ifcat$\the\@ltok@argtwo$%
      \global\@ltok@ptop{}%
      \ifcat$\the\@ltok@citepage$%
       \global\@ltok@conetop{}%
      \fi%
   \fi%
```

Now we run a series of checks to determine the class of citation we are dealing with, and if tables for that type of citation have been turned on, an export is performed; if a cite for export is to a statute, then the special routines for handling page references is brought into play.

```
  \if@law@multipages%
    \global\@ltok@conetop\@ltok@conetopplural%
    \global\@ltok@atbridge\@ltok@atbridgeplural%
  \fi%
  \ifcat$\the\@ltok@argtwo$%
   \global\@ltok@ptop{}%
   \global\@ltok@atbridge{}%
```

```
  \ifcat$\the\@ltok@citepage$%
    \global\@ltok@conetop{}%
   \fi%
```

And here the `\@law@setup` routine comes to an end; punt to the print routine, possibly followed by a gobble, if *Id.* is printed.

```
 \fi}
```

3.3 Output routines

Once the citation has been unpacked and the basic tidying up appropriate to all appearances of the citation has been carried out, it is time to ship its contents to the various output routines.

3.3.1 The print macro

This section contains the code for the print routine and its supporting macros. We begin with the supporting stuff.

`\@law@justabovecheck` Check if immediately preceding cite is the same as this one. This macro is called immediately before the citation is printed. There are lots of conditions, but the structure is just a straightforward nesting of `\if` statements, each of which sets a toggle or toggles appropriately. In the end, this macro tells us:

- Whether the immediately-preceding citation is to the same work; and if it is
- Whether the preceding cite had no pinpoint page reference; and if it had one
- Whether the pages referred to by the two cites are exactly identical.

Try *that* with WordPerfect! If Camel isn't getting the decisions right, let me know.

```
\def\@law@justabovecheck{%
  \xdef\@law@temptwo{\the\@ltok@author\the\@ltok@name%
                     \the\@ltok@citefirst\the\@ltok@citepage%
                     \the\@ltok@citelast}%
  \ifx\@law@lastcite\@law@temptwo%
    \ifnum\the\c@citesinfoot=1\relax%
     \@law@justabovecheckdetails%
    \else%
      \ifx\@law@currentcite\@law@temptwo%
       \@law@justabovecheckdetails%
      \else%
       \@justabovefalse%
      \fi%
    \fi%
  \else%
    \ifnum\the\c@citesinfoot>1\relax%
      \ifx\@law@currentcite\@law@temptwo%
       \@law@justabovecheckdetails%
```

```
      \else%
       \@justabovefalse%
      \fi%
    \fi%
  \fi}
\def\@law@justabovecheckdetails{%
    \ifx\@law@lastpage\empty%
     \@justabovetrue%
     \@l@quiteexactfalse%
    \else%
     \ifcat$\the\@ltok@argtwo$%
       \@justabovefalse%
      \else%
       \@justabovetrue%
        \xdef\@law@temp{\the\@ltok@argtwo}%
        \ifx\@law@temp\@law@lastpage%
         \@l@quiteexacttrue%
        \else%
         \@l@quiteexactfalse%
        \fi%
      \fi%
    \fi}
⟨/lexitex⟩
```

3.3.2 Proof sheet master document

The following short document will generate proof sheets for all of the primary styles on the system. With the standard distribution, this produces twelve pages of output.

3.4 Proof sheets

The following macros produce a proof sheet that shows all of the permutations of a citation that I can think of. This is useful for those who have to design new citation styles. This is a pretty straightforward exercise, except for the task of feeding a cite declaration exactly the right arguments; this requires re-definition of `\newcitestyle` and the re-parsing of the `lexicite.tex` file.

If you find permutations that are not represented here, let me know and I'll add them.

3.5 Macros for data export

`\citationsubject` This macro toggles on the requirement that every citation be assigned to a subject category, and adds a subject to the subject list, with all of the information needed to create a classified bibliography. This information consists of:

1. The nickname for the category, used as the argument to the `s=` option in each citation;

2. The title for use in the bibliography.

An optional argument of `t` creates a table, and must be accompanied by `o=` and `i=` options giving the extensions of the input and output files, and by `h=` and optionally may be accompanied by an option of `p` to include poinpoints in the table.

```
⟨*lexitex⟩
\def\@law@slone#1#2{\def\@law@listitem{#1}%
  \ifx\@law@optionitem\@law@listitem #2%
  \let\\=\@law@finish\fi}
```

`\@law@end` The internal ending character is also common to all parsers.

```
\def\@law@end{,}
```

`\@law@slnomatch` If when we execute the list of options we do not find a match, we want an error message to go to the terminal. This is the macro that gives the appropriate error message.

```
\def\@law@slnomatch{\@camel@error{Citation Warning: Invalid
    option `\@law@optionitem'
    given to \string\citationsubject.^^J%
   Valid options are:^^J
    2 (force a second-level header for a subject)^^J
    p (to include pinpoints in an exported table)^^J
    i=<extension> (use the three-letter extension in^^J
                   exporting a table)^^J
    o=<volume>  (use the three-letter extension in^^J
                  importing a table)^^J%
To avoid errors in formatting, use commas in the option string^^J%
only where they are logically necessary to the sense of the^^J%
string.}\@ehc}
```

`\longestlabelfor` This macro set returns a box the exact width of the longest string given as a label to citations of a specified subject class.

```
\newlength{\templen}
\newlength{\lawlengthone}
\newlength{\lawlengthtwo}
\def\@law@getlabel#1{\settowidth\lawlengthone{\csname @law@#1@lab\endcsname}%
  \ifnum\lawlengthone>\lawlengthtwo%
    \settowidth\lawlengthtwo{\csname @law@#1@lab\endcsname}%
  \fi}
\def\longestlabelfor#1{%
   \setlength\lawlengthone{0pt}%
   \setlength\lawlengthtwo{0pt}%
   \let\\=\@law@getlabel%
   \csname @law@#1@citelist\endcsname}
```

`\@law@sourceoptionlist` The list of options is simple and expansible. The syntax of the last two lines is required if everything is to work as expected. The items in the help message should

probably be drawn from here, to provide a hook for expansion if other styles want to tinker with the options to `\cite`.

```
\def\@law@subjectoptionlist{%
 \\{1}{\gdef\@law@level{1}}%
 \\{2}{\gdef\@law@level{2}}%
 \\{p}{\global\@law@usepinpointstrue}%
 \\{P}{\gdef\@law@bibformat{\global\@law@usepagestrue}}%
 \\{o}{\gdef\@law@parsemacro##1{\gdef\@law@outputext{##1}}%
       \let\@law@parse\@law@slparsecomma}%
 \\{i}{\gdef\@law@parsemacro##1{%
          \ifcat$##1$\else%
          \global\@law@maketabletrue%
          \expandafter\gdef\csname\@law@temp inputext\endcsname{##1}%
          \fi}%
        \let\@law@parse\@law@slparsecomma}%
 \\{=}{\let\@law@parse\@law@slparsebumequal}%
 \\{,}{}%
 \\{\end}{\let\@law@nomatch\relax\let\@law@parse\relax}%
    \@law@nomatch}
```

`\@law@parselastcheck` `\@law@parsecomma` `\@law@parselastcheck` `\@law@parseplain`

We switch between three possible forms for the `\@law@parse` macro, and we also need to have a special way of ending when the delimiter is found in the list.

```
\def\@law@slparsecomma=#1,{%
  \@law@parsemacro{#1}\let\@law@parse\@law@slparseplain%
  \@@parse}

\def\@law@slparsebumequal#1,{\let\@law@parse\@law@slparseplain\@@parse}

\def\@law@slparseplain#1{%
  \let\@law@nomatch=\@law@slnomatch%
  \def\@law@optionitem{#1}\let\\=\@law@slone\@law@subjectoptionlist\@@parse}

\def\citationsubject{%
  \@ifnextchar[{\@citationsubject}{\@citationsubject[1]}}
{\catcode`\|=13%
\gdef\@citationsubject[#1]#2#3{%
  \global\let\@law@level\relax%
  \global\@law@usepinpointsfalse%
  \global\@law@maketablefalse%
  \global\@law@requiresubjectstrue%
  \gdef\@law@bibformat{}%
  \def\@law@temp{#2}%
  {\let\@law@parse=\@law@slparseplain\@law@parse #1,\end}%
  \expandafter\gdef\csname write@one@#2@entry\endcsname{}%
  \if@law@maketable%
  \expandafter\def\csname the#2table\endcsname{%
    \@restonecoltrue%
    \if@twocolumn\@restonecolfalse\fi%
    \section*{#3}%
    \@mkboth%
      {\uppercase{#3}}%
```

```
    {\uppercase{#3}}%
  \thispagestyle{plain}%
  \parindent\z@\parskip\z@ plus.3\p@\relax%
  \let\item\citationtableitem}
\expandafter\def\csname endthe#2table\endcsname{%
  \if@restonecol\onecolumn%
  \else\clearpage%
  \fi}%
\def\itemspace{%
  \par\vskip 10\p@ plus5\p@ minus3\p@\relax}
\newindex{#2}{\@law@outputext}{\csname#2inputext\endcsname}{#3}%
\if@law@usepinpoints%
    \expandafter\gdef\csname write@one@#2@entry\endcsname{%
      {\ifcat$\the\@ltok@argtwo$\@ltok@argtwo{\@law@delim}\fi%
        \@ltok@pagesstring\@ltok@argtwo%
        \def\@law@comma{\@law@exportapage}%
        \def\@law@ampersand{\@law@exportapage}%
        \def\@law@dash{\@law@exportapage}%
        \def|{\@law@exportapage}%
        \def\@law@exportapage####1\@law@delim{%
          \@ltok@argtwo{####1}%
          \def\@law@temp{\index[#2]}%
          \edef\@law@temptwo{%
           \noexpand{\string\bibitem %
       \string\source[bZ=\the\@ltok@nickname]%
       \noexpand{\the\@ltok@nickname\noexpand}%
        \ifcat$\the\@ltok@argtwo$\else>####1\fi\string|maybe\noexpand}}%
          \expandafter\@law@temp\@law@temptwo}
          \expandafter\@law@exportapage\the\@ltok@pagesstring}}%
\else%
    \expandafter\gdef\csname write@one@#2@entry\endcsname{%
     \def\@law@temp{\index[#2]}%
      \edef\@law@temptwo{%
        \noexpand{\string\bibitem %
        \string\source[bZ=\csname @law@\the\@ltok@nickname @lab\endcsname]%
       \string{\the\@ltok@nickname\string}\string|maybe\noexpand}}%
     \expandafter\@law@temp\@law@temptwo}%
\fi%
\fi%
 \@ltok@a=\expandafter{\@law@subjectlist}%
 \@ltok@c=\expandafter{\@law@bibformat}%
 \if@law@usepinpoints%
   \edef\@law@temp{\noexpand\\\noexpand{#2\noexpand}%
                   \noexpand{p\noexpand}%
                   \noexpand{#3\noexpand}
                   \noexpand{\the\@ltok@c\noexpand}}%
   \@ltok@b=\expandafter{\@law@temp}%
 \else%
   \edef\@law@temp{\noexpand\\\noexpand{#2\noexpand}
                   \noexpand{\@law@level\noexpand}
```

```
                    \noexpand{#3\noexpand}
                    \noexpand{\the\@ltok@c\noexpand}}%
    \@ltok@b=\expandafter{\@law@temp}%
  \fi%
  \edef\@law@subjectlist{\the\@ltok@a\the\@ltok@b}}}
```

`\printbibliography` To print bibliographies, we use a single command with two modes built into it.

```
\def\printbibliography#1{%
  \let\camelrefname\refname%
  \expandafter\ifx\csname#1inputext\endcsname\relax%
    \global\@law@tablefalse%
  \else%
    \global\@law@tabletrue%
  \fi%
  \begingroup%
    \def\refname{\@law@getsubjectheader{#1}}%
      \@law@usepagesfalse%
      \@law@getbibformat{#1}
      \let\maybe\@gobble%
      \let\camelfill\relax%
      \let\camelarrow\relax%
      \if@law@table%
        \advance\@totalleftmargin -\labelsep%
        \advance\linewidth \labelsep%
        \def\@biblabel##1{}%
        \if@law@usepages%
          \let\maybe\relax%
          \let\camelfill\dotfill%
          \let\camelarrow\rightarrow%
        \fi%
      \else%
        \longestlabelfor{#1}%
        \ifdim\lawlengthtwo=0pt%
          \advance\@totalleftmargin -\labelsep%
          \advance\linewidth \labelsep%
          \def\@biblabel##1{}%
        \fi%
      \fi%
    \if@law@requiresubjects%
      \@law@dobibliography{#1}%
    \else%
      \gdef\@law@temp{#1}\gdef\@law@temptwo{all}%
      \ifx\@law@temp\@law@temptwo%
        \@law@dobibliography{#1}%
      \else%
      \message{^^J==================================================^^J%
                  Camel bibliography ^^J
                   Document type: without-subjects ^^J
                   Bibliography type: subject-specific (#1) ^^J
                   Action: impossible task, ignoring ...^^J%
```

```
                  ==================================================^^J}
      \fi%
    \fi%
  \endgroup}

\def\@law@dobibliography#1{%
    \begin{thebibliography}{\longestlabelfor{#1}\rule{\lawlengthtwo}{5pt}}
      \if@law@table%
        \if@law@usepages%
          \addtolength{\parsep}{-.5\parsep}%
          \addtolength{\itemsep}{-.5\itemsep}%
        \fi%
      \fi%
      \if@law@requiresubjects%
      \@law@requiresubjectsfalse%
        \if@law@table
      \message{^^J==================================================^^J%
                  Camel bibliography ^^J
                   Document type: subject-classified cites ^^J
                   Bibliography type: subject-specific external (#1) ^^J
                   Action: generating^^J%
                  ==================================================^^J}
          \@input{\jobname.\csname#1inputext\endcsname}
        \else%
        \gdef\@law@temp{#1}\gdef\@law@temptwo{all}%
        \ifx\@law@temp\@law@temptwo%
      \message{^^J==================================================^^J%
                  Camel bibliography ^^J
                   Document type: subject-classified cites ^^J
                   Bibliography type: all citations ^^J
                   Action: header only (not yet implemented)^^J%
                  ==================================================^^J}
        \else%
      \message{^^J==================================================^^J%
                  Camel bibliography ^^J
                   Document type: subject-classified cites ^^J
                   Bibliography type: subject-specific internal (#1) ^^J
                   Action: generating^^J%
                  ==================================================^^J}
          \let\\\@law@bibslash%
          \csname @law@#1@citelist\endcsname
        \fi%
        \fi%
      \else%
      \message{^^J==================================================^^J%
                  Camel bibliography ^^J
                   Document type: without-subjects ^^J
                   Bibliography type: all citations ^^J
                   Action: generating^^J%
                  ==================================================^^J}
```

```
        \let\\\@law@bibslash%
        \csname @law@#1@citelist\endcsname%
      \fi%
    \end{thebibliography}}
\def\@law@bibslash#1{\bibitem [\csname @law@#1@lab\endcsname] \source[b]{#1}. }
```
⟨/lexitex⟩

3.5.1 Citation style definitions

No styles are included here. See the separate style files in the CAMEL archive for the matching `.bst`, `.cst` and `.cit` files that make up each CAMEL style variation. These may be distributed as `.doc` files, and should be available on CTAN under `/macros/latex/contrib/supported/camel`.

3.6 Index styles

This code, and the table writing code, is provisional; this should be made more general. Less of a hack than the original LEXITEX code, though!

⟨∗camelindex⟩
```
actual '#'
quote '!'
level '>'
preamble
"\n"
postamble
"\n"
item_0    "\n"
item_x1   "\n \\subitem "
item_x2   "\n \\subsubitem "
delim_n "\$\\rightarrow\$"
delim_0 "\~\\camelfill"
delim_1 "\$\\camelarrow\$"
% The next lines will produce some warnings when
% running Makeindex as they try to cover two different
% versions of the program:
lethead_prefix   "{\\bf\\hfil "
lethead_suffix   "\\hfil}\\nopagebreak\n"
lethead_flag       1
heading_prefix   "{\\bf\\hfil "
heading_suffix   "\\hfil}\\nopagebreak\n"
headings_flag       0
\ifx\on@line\undefined
  \def\on@line{ on input line \the\inputlineno}
  \errhelp{Support for input line numbers has been added
           to latex.tex <dec91>.^^J^^J%
           Please update to a newer LaTeX release.}
  \errmessage{Obsolete LaTeX release (older than Dec.91)}
\fi
```
⟨/camelindex⟩

4 A BibTEX Library

Considerable modification of an existing `.bst` file may be required to make it give output that can be digested by Camel. You may find that it is actually easier to draft you entry type functions from scratch, using the following library of BibTEX functions. See the file `law.dtx` for an example of entry types based on this library, and for an example of how to tie the library and your own functions together through a driver file entry. Each function defined using the `FUNCTION` operator is accompanied by a brief description of what it is intended to do, followed by a box in the following form:

Expected:	1.	What the function expects to find on the stack, numbered from the top (most recently pushed) to the bottom item.
Left:	1.	What the function will leave on the stack, numbered in the same fashion.
Toggles:	1.	**toggle name:** What the effect of any toggles is on the behaviour of the function.

Happy reading.

4.1 Hello!

As ever, we start by telling the world who we are.

```
⟨*bstheader⟩
FUNCTION { hello }
{ "     ============================================="
"     bibliography package, version 1.0k or later"
"     Use with LaTeX2e and the Camel prototype"
"     ---------------------------------------------"
"     documentation last updated: 5 December 1994"
"     code last updated: 22 March 1995"
"     style version: 1.0i"
⟨/bstheader⟩
⟨*bstlibrary⟩
"     ============================================="
stack$
}
```

4.2 Variable initializations

`ENTRY` This command takes three braced arguments, which in order are:

1. A list of field names;
2. A list of integer variables; and
3. A list of string variables.

The field names created here include some not found in the standard BibTeX styles. The use of both the standard and these new items is fully documented elsewhere, so no commentary is given here. Suffice to say that each item in the list creates a field that subsequent code can work on.

```
ENTRY
```

field names

```
{ address
annote
author
booktitle
chapter
edition
editor
howpublished
institution
journal
key
month
note
number
organization
pages
publisher
school
series
title
type
volume
year
date
jurisdiction
court
division
divno
casedate
translator
booktranslator
cites
}
```

entry integers

None are defined. Note that any integers or strings defined with the ENTRY command are created for *each* entry in the citation list.

```
{}
```

entry strings

There are no general string variables, either. Such variables might be defined for use in building key labels for use in sorting, or in an end-of-document bibliography style.

```
{ label extra.label sort.label }
```

STRINGS

A few string macros are needed to hold things temporarily removed from the

stack. BibTeX only allows ten such string variables; the named variables in this list will slowly be reduced in number through redesign of the functions that require them. Meanwhile, let's hope we don't find a sudden need for more ...

```
STRINGS { s t u v scrubdate
 volume.var year.var number.var journal.var pages.var }
```

INTEGERS The following list of integers is probably also larger than it needs to be. Again, optimization of stack usage will lead to the elimination of some of these variables.

```
INTEGERS { itemcount date.specials charcount a b c
           nameptr namesleft numnames a.logical }
```

FUNCTIONS These functions provide strings used in the parsing of Japanese legal citations. If the language of such citations is not English, these strings may need to be edited.

```
FUNCTION {dc..} {"District Court"}
FUNCTION {sc..} {"Supreme Court"}
```

4.3 Function definitions

4.3.1 Logic and measurement

not and or These have been copied verbatim from Oren Patashnik's 1988 release. His commentary on these functions reads: *These three functions pop one or two (integer) arguments from the stack and push a single one, either `0` or `1`. The `'skip$` in the `'and'` and `'or'` functions are used because the corresponding* `if$` *would be idempotent.*

Expected:	1.	The result of a logical test
	2.	The result of a second logical test
Left:	1.	A single logical test result
Toggles:	1.	none

```
FUNCTION {not}
{   { #0 }
    { #1 }
  if$
}
FUNCTION {and}
{   'skip$
    { pop$ #0 }
  if$
}
FUNCTION {or}
{   { pop$ #1 }
    'skip$
  if$
}
```

times.ten One of the date calculation routines requires string-to-integer conversion.

BibTeX primitives can only convert in this direction for single characters. Accordingly, we need to be able to raise by a power of ten.

Expected:	1.	A single integer.
Left:	1.	A single integer.
Toggles:	1.	none

```
1415 FUNCTION {times.ten}
1416 { duplicate$ duplicate$ duplicate$ duplicate$ duplicate$
1417   duplicate$ duplicate$ duplicate$ duplicate$
1418   + + + + + + + + +
1419 }
```

`character.length` BibTeX provides a built-in command for measuring the length of a string in "text units". Sometimes it's nice to know the length of a string in *characters*; this function provides that facility.

Expected:	1.	A string.
Left:	1.	An integer giving the number of characters in the string.
Toggles:	1.	none

```
1420 FUNCTION {character.length}
1421     { 'u :=
1422       #0 'charcount :=
1423       { u empty$ not }
1424         { u #2 global.max$ substring$ 'u :=
1425           charcount #1 + 'charcount :=
1426         }while$
1427       charcount
1428     }
```

`first.in.second` This function checks for a substring at the beginning or at the end of a given string. This facility is needed, for example, to provide an automated means of toggling the formatting of the court division string for Japanese cases. In the defintion below, `t` is the substring length, and `u` is the string length.

Expected:	1.	A string toggle.
	2.	A string (the string in which to look).
	3.	A string (the substring to look for).
Left:	1.	A single integer (`0` or `1`), indicating whether the substring was found at the specified location in the string for search.
Toggles:	1.	**end:** The function looks at the end of the given string.
	2.	**start:** The function looks at the start of the given string.

```
1429 FUNCTION {first.in.second}
1430     { 's :=
1431       duplicate$ empty$
```

```
    { pop$ pop$ #0 #0 }
    { swap$ duplicate$ character.length 'a :=
      swap$ duplicate$ character.length 'b :=
      b a <
      { pop$ pop$ #0 #0 }
      { s "end" =
        { b a - #1 + global.max$ substring$ =
          { b a - #1 + #1 }
          { #0 #0
          }if$
        }
        { s "start" =
          { #1 a substring$ =
            { #1 #1 }
            { #0
            }if$
          }
          { s "reverse" =
            { b a - #1 + 'b :=
              { b #0 > }
              { duplicate$
                b a substring$
                's :=
                swap$ duplicate$ s =
                  { pop$ pop$ b #0 'b := #1 }
                  { b #1 =
                    { pop$ pop$ #0 #0 'b := #0 }
                    { swap$ b #1 - 'b :=
                    }if$
                  }if$
              }while$
            }
            { b a - #1 + 'b := #1 'c :=
              { c b < }
              { duplicate$
                c a substring$
                's :=
                swap$ duplicate$ s =
                  { pop$ pop$ c b 'c := #1 }
                  { b c - #1 =
                    { pop$ pop$ #0 b 'c := #0 }
                    { swap$ c #1 + 'c :=
                    }if$
                  }if$
              }while$
            }if$
          }if$
        }if$
      }if$
    }if$
```

```
1482    }
```

`get.character.type` This function returns one of three strings, '`letter`', '`numeral`' or '`other`' to indicate the type of the character it finds on the stack. This was written for use in the `gather.chars` function, but may find other uses as well.

Expected:	1.	Either `*` or a single character to be scanned for.
	2.	A single character to be examined by the function.
Left:	1.	Either `letter`, `numeral` or `other`, the significance of which depends upon the toggle used.
Toggles:	1.	`*` causes the character to be identified as a numeral, and letter or something else.
	2.	Any character other than `*` causes that character to be identified as `other`, and any other characters to be identified as `letter`.

```
1483 FUNCTION { get.character.type }
1484   { duplicate$ "*" =
1485     { pop$ duplicate$ empty$
1486       { pop$ "other" }
1487       { chr.to.int$ duplicate$
1488         duplicate$ #47 > swap$ #58 < and
1489         { pop$ "numeral" }
1490         { duplicate$ #64 > swap$ duplicate$ #91 < swap$
1491           duplicate$ #96 > swap$ #123 < and 'a.logical :=
1492           and a.logical or
1493           { "letter" }
1494           { "other"
1495           }if$
1496         }if$
1497       }if$
1498     }
1499     { =
1500       { "other" }
1501       { "letter"
1502       }if$
1503     }if$
1504   }
```

`type.last.char` This was written in order to handle situations where trailing punctuation varies depending upon whether the text being punctuated ends in a numeral or not. The power of the library is (finally, in July of 1995) starting to show through.

Expected:	1.	A single string.
Left:	1.	A string, either `numeral`, `letter` or `other`.
Toggles:	1.	None.

```
1505 FUNCTION { type.last.char }
1506   { duplicate$ character.length
```

```
1507    #1 substring$
1508    "*" get.character.type
1509  }
```

4.3.2 Housekeeping

These functions are used to issue warning messages and avoid errors during processing.

`empty.to.null` This is simply the `field.or.null` function from the standard styles, renamed so that I could more easily remember what it does. Invoked only by the output routines, it assures that empty field markers are replaced with the null string, to prevent smash-ups.

Expected:	1.	The contents of a field, which may be an empty field marker.
Left:	1.	A string, possibly the null string.
Toggles:	1.	none

```
1510 FUNCTION {empty.to.null}
1511 { duplicate$ empty$
1512     { pop$ "" }
1513     'skip$
1514   if$
1515 }
```

`check` This check function isolates the production of warning messages from the output routines, to enhance transparency. Note that this will not convert an empty field marker to the null string; that is left to the output routines.

Expected:	1.	A string for use in an error message.
	2.	The contents of a field.
Left:	1.	The field contents.
Toggles:	1.	none

```
1516 FUNCTION { check }
1517 { 't :=
1518   duplicate$ empty$
1519   { "empty " t * " in " * cite$ * warning$}
1520   'skip$
1521   if$
1522 }
```

`either.or` This checks whether one of a two-item pair is empty. If at least one is empty, the other is returned to the stack. If both are non-empty, a warning is issued, and one item is arbitrarily selected.

Expected:	1.	A string.
	2.	A string.
Left:	1.	A single string.
Toggles:	1.	none

```
FUNCTION {either.or}
{ duplicate$ empty$
  { pop$ duplicate$ empty$
    { pop$ "" }
    'skip$
    if$
  }
  { swap$ duplicate$ empty$
    { pop$ }
    { "both items in an either.or pair are non-empty in " cite$ *
      warning$
      "  I'm using only ONE of these items (the second passed by the function)."
      warning$
      pop$
    }if$
  }if$
}
```

`either.or.nowarning` This does the same thing as `either.or`, but does not issue a warning if both items exist. As with that function, the second item pushed is preferred.

Expected:	1.	A string.
	2.	A string.
Left:	1.	A single string.
Toggles:	1.	none

```
FUNCTION {either.or.nowarning}
{ duplicate$ empty$
  { pop$ duplicate$ empty$
    { empty.to.null }
    'skip$
    if$
  }
  { swap$ pop$ }
  if$
}
```

4.3.3 Output

In the interest of transparency, I have tried to make the output functions as simple to use and as flexible in their operation as possible. Every output routine expects to find three items on the stack, and the name of each routine describes how it

will react to what it finds there. To keep things tidy, the empty field marker is replaced with the null string only by the output routines.
The names are sometimes coincidentally comical, but I hope informative as well.

`must.must.must` The simplest output routine is the `must.must.must` routine; it simply catenates the three strings, replacing any empty field markers it finds with the null string.

Expected:	1.	A string.
	2.	A string.
	3.	A string.
Left:	1.	none
Toggles:	1.	none

```
1550 FUNCTION {must.must.must}
1551     { empty.to.null 't :=
1552       empty.to.null swap$ empty.to.null swap$ t
1553       * *
1554       write$
1555     }
```

`might.ifone.must` This output routine is heavily used for conditional output. If the deepmost of the three stack items is an empty field marker, the only the topmost item is written to the output file. If this is not the case, then the three items are written in order.

Expected:	1.	A string.
	2.	A string.
	3.	A string.
Left:	1.	*none*
Toggles:	1.	none

```
1556 FUNCTION {might.ifone.must}
1557     { empty.to.null 't :=
1558       swap$ duplicate$ empty$
1559       { pop$ pop$ t }
1560       { swap$ empty.to.null t * * }
1561       if$
1562       write$
1563     }
```

`iftwo.might.iftwo` This function is typically used to output a formatted item in enclosing braces, where the entire item and its braces should be suppressed if the item is empty.

Expected:	1.	A string.
	2.	A string.
	3.	A string.
Left:	1.	none
Toggles:	1.	none

```
1564 FUNCTION {iftwo.might.iftwo}
1565     { 't :=
1566       duplicate$ empty$
1567        { pop$ pop$ }
1568        { empty.to.null swap$ empty.to.null swap$
1569          t empty.to.null
1570          * * write$ }
1571       if$
1572     }
```

`ifthree.ifthree.might` This outputs a pair of prefixes if the suffix to which they are to be attached exists. Otherwise it outputs a null string.

Expected:	1.	A string.
	2.	A string.
	3.	A string.
Left:	1.	none
Toggles:	1.	none

```
1573 FUNCTION {must.ifthree.might}
1574     { 't :=
1575       duplicate$ empty$
1576       { pop$ write$ }
1577       { swap$ empty.to.null
1578         swap$ empty.to.null
1579         t * * write$ }
1580       if$
1581     }
```

4.3.4 Parsing and conversion

`field.tag.no.combine` The `no` here stands for "number". This function is typically used to simplify the task of building a numbered label for something (e.g. a technical report). It provides facilities for handling blank entries, so that these error-handling routines do not need to be drafted from scratch for every such situation.

Expected:	1.	A string toggle.
	2.	An identifier (i.e. a report number).
	3.	A text prefix for the identifier.
	4.	A string label (i.e. "Technical Report" or somesuch).
Left:	1.	A single string.
Toggles:	1.	**endlabel:** Combines the text items in the same order in which they are found.
	2.	frontlabel: Places the label in front, followed by the identifier, followed by the first item pushed.

```
1582 FUNCTION {field.tag.no.combine}
1583  { "endlabel" =
1584    { duplicate$ empty$
1585      { pop$ pop$ empty.to.null }
1586      { empty.to.null 's := empty.to.null
1587      swap$ empty.to.null swap$ s * *
1588      }if$
1589    }
1590    { duplicate$ empty$
1591      { pop$ pop$ empty.to.null }
1592      { empty.to.null 's := empty.to.null
1593        swap$ empty.to.null s swap$ * *
1594      }if$
1595    }if$
1596  }
```

`change.letter.case` This is a simple front-end to the `change.case$` built-in function, used to change characters in a string from upper to lower case. It slightly enhances the built-in function by providing an option that leaves the string alone.

Expected:	1.	A single character—either an `n`, a `t`, an `l`, or a `u`.
	2.	A string for conversion.
Left:	1.	One string, converted according to the option toggle pushed after the string.
Toggles:	1.	**n:** yields the string unchanged.
	2.	**t:** yields all lower case letters except for the first.
	3.	l: yields all lower case letters.
	4.	u: yields all upper case letters.

```
1597 FUNCTION {change.letter.case}
1598    { 't :=
1599      duplicate$ empty$
1600        'skip$
1601        { t chr.to.int$ "n" chr.to.int$ =
1602          'skip$
1603          { t change.case$ }
1604          if$
```

```
1605          }
1606        if$
1607      }
```

`n.dashify` The `n.dashify` function makes each single `-` in a string a double `--`. if it's not already.[16]

Expected:	1.	A single string for conversion.
Left:	1.	A single converted string.
Toggles:	1.	none

```
1608 FUNCTION {n.dashify}
1609 { 't :=
1610   ""
1611     { t empty$ not }
1612     { t #1 #1 substring$ "-" =
1613      { t #1 #2 substring$ "--" = not
1614          { "--" *
1615            t #2 global.max$ substring$ 't :=
1616          }
1617          {   { t #1 #1 substring$ "-" = }
1618           { "-" *
1619             t #2 global.max$ substring$ 't :=
1620           }
1621           while$
1622          }
1623        if$
1624      }
1625      { t #1 #1 substring$ *
1626        t #2 global.max$ substring$ 't :=
1627      }
1628       if$
1629     }
1630   while$
1631 }
```

`gather.chars` The following function was designed for date parsing, but may find other uses as well. It finds the first letter or numeral in a given string, then proceeds to build a substring until it hits a different character type or the end of the string, at which point it stops parsing and terminates.

[16] This comment by Oren Patashnik.

Expected:	1.	A string for parsing.
Left:	1.	A string toggle showing the character type of the first homogenous substring in the string for parsing.
	2.	A string of characters from the front of the given string which are letters or numbers only.
	3.	The remainder of the string.
Toggles:	1.	none

```
1632 FUNCTION { gather.chars }
1633   {
1634     "forward" =
1635     { swap$ duplicate$ character.length 'a :=
1636       't :=
1637       duplicate$
1638       { t #1 #1 substring$ swap$ get.character.type
1639         "other" =
1640         t empty$ not and }
1641       { t #2 global.max$ substring$ 't := duplicate$
1642       }while$
1643       duplicate$ t #1 #1 substring$ swap$ get.character.type
1644       t #1 #1 substring$ swap$
1645       "" swap$ duplicate$ 'u :=
1646       t #2 global.max$ substring$ 't :=
1647       { u = }
1648       { * swap$
1649         u * 'u :=
1650         u #1 #1 substring$ swap$
1651         u #1 #1 substring$
1652         u #2 global.max$ substring$ 'u :=
1653         t #1 #1 substring$ swap$ get.character.type
1654         t #1 #1 substring$ swap$
1655         duplicate$ u =
1656         { t #2 global.max$ substring$ 't :=  }
1657         { swap$ pop$
1658         }if$
1659       }while$
1660       swap$ pop$ t swap$ u
1661     }
1662     { swap$ duplicate$ character.length 'a :=
1663       't :=
1664       duplicate$
1665       { t a #1 substring$ swap$ get.character.type
1666         "other" =
1667         t empty$ not and }
1668       { a #1 - 'a :=
1669         t #1 a substring$ 't :=
1670       }while$
1671       duplicate$ t a #1 substring$ swap$ get.character.type
```

```
      t a #1 substring$ swap$
      "" swap$ duplicate$ 'u :=
      a #1 - 'a :=
      t #1 a substring$ 't :=
      { u = }
      { swap$ * swap$
        u * 'u :=
        u #1 #1 substring$ swap$
        u #1 #1 substring$
        u #2 global.max$ substring$ 'u :=
        t a #1 substring$ swap$ get.character.type
        t a #1 substring$ swap$
        duplicate$ u =
        { a #1 - 'a := t #1 a substring$ 't :=  }
        { swap$ pop$
        }if$
      }while$
      swap$ pop$ t swap$ u
    }if$
  }
```

`tie.or.space.connect` We use this function from the standard styles. It adds a tie if the string it is applied to is three characters or less in length.

Expected:	1.	A string
	2.	A second string
Left:	1.	One string
Toggles:	1.	none

```
FUNCTION {tie.or.space.connect}
{ duplicate$ text.length$ #3 <
    { "~" }
    { " " }
  if$
  swap$ * *
}
```

`format.pages` This function accepts one toggle option currently, but it is open to expansion. If the `short` toggle is fed to this function, it reads through the field until it hits a non-integer character. Otherwise, it `n.dashify`s the whole field. This needs robustification; the page number might be a roman numeral. Might it be simplest to just drop the scan when we hit a `-`? Deserves some thought.

Expected:	1.	A toggle string.
	2.	page number or possibly a range of pages.
Left:	1.	One string, constituting a finished page number, or `empty$`.
Toggles:	1.	**short:** will return only the first number given in the string.
	2.	full: will return the page number, n-dashified.

```
1699 FUNCTION {format.pages}
1700     { swap$ duplicate$ empty$
1701       { pop$ pop$ "" }
1702       { swap$ duplicate$ "short" =
1703         { pop$
1704           's :=
1705           ""
1706           s #1 #1 substring$
1707           { "-" = not }
1708             { s #1 #1 substring$ *
1709               s #2 global.max$ substring$ 's :=
1710               s #1 #1 substring$
1711               duplicate$ "" =
1712                 { pop$ "-" }
1713                 'skip$
1714                 if$
1715             }
1716           while$
1717         }
1718       { "full" =
1719         { pages n.dashify }
1720         { "invalid switch fed to the format.pages function"
1721            warning$
1722         }if$
1723       }if$
1724     }if$
1725 }
```

Names The code for parsing and formatting names is extremely economical, thanks to the powerful built-in functions that BibTeX supplies for this purpose.

`format.names` This is based on Oren Patashnik's original function of the same name. His comment was as follows:

> The format.names function formats the argument (which should be in BibTeXname format) into "`First Von Last, Junior`", separated by commas and with an `and` before the last (but ending with `et~al.` if the last of multiple authors is `others`). This function's argument should always contain at least one name. The format.authors function returns the result of format.names(author) if the author is present, or else it returns the null string.

This function is used to format any name field that is thrown at it. It is based

on the 1988 release, but with modifications to permit toggling, since the Blue Book requires different author formats for different types of material. The toggling strategy should also make modification of this style a pretty simple matter.

Expected:	1.	The contents of one name field
	2.	string toggle (either `firstinitial`, `lastonly` or `full`)
Left:	1.	A single string, containing a formatted name or names, or `empty$`
Toggles:	1.	**firstinitial:** Yields the form "F. Bennett, Jr."
	2.	lastonly: Yields the form "Bennett"
	3.	**full:** yields "Frank Bennett, Jr."

```
FUNCTION {format.names}
    { swap$ duplicate$ empty$
        { swap$ pop$ }
        { 's :=
          'u :=
          #1 'nameptr :=
          s num.names$ 'numnames :=
          numnames 'namesleft :=
          { namesleft #0 > }
            { u "lastonly" =
              { s nameptr "{vv~}{ll}" format.name$ 't := }
              { u "firstinitial"  =
                { s nameptr "{f.~}{vv~}{ll}{, jj}" format.name$ 't := }
                { u "full" =
                  { s nameptr "{ff~}{vv~}{ll}{, jj}" format.name$ 't :=}
                  { "style error; invalid or non-existent toggle" warning$ }
                  if$
                }
                if$
              }
              if$
              nameptr #1 >
                { namesleft #1 >
                  { ", " * t * }
                  { numnames #2 >
                    { "," * }
                    'skip$
                    if$
                    t "others" =
                      { " et~al." * }
                      { " and " * t * }
                    if$
                  }
                if$
                }
                't
```

```
            if$
            nameptr #1 + 'nameptr :=
            namesleft #1 - 'namesleft :=
          }
        while$
      }
    if$
  }
```

Dates The style code for parsing and formatting dates is much more complex than that for names. This is due to the need to build the necessary tools out of BibTeX primitives, since no built-in tools for this task are supplied. There are two date formatting routines below. The simpler of the two is `format.month.year`. This is basically just the function supplied with the standard BibTeX styles, under a new name. The LexiBib version of `format.date` is underpinned by a whole set of new functions, and allows great flexibility in the syntax for entering dates.

`format.month.year`

Old format routine The `format.date` function is for the month and year, but we give a warning if there's an empty year but the month is there, and we return the empty string if they're both empty.[17]

This is not changed over the `format.date` function in the original standard BibTeX styles. It is retained, although the distributed LexiBib styles won't be making use of it.

Expected:	1.	none
Left:	1.	A single string.
Toggles:	1.	none

```
FUNCTION {format.month.year}
{ year empty$
    { month empty$
     { "" }
     { "there's a month but no year in " cite$ * warning$
       month
     }
      if$
    }
    { month empty$
     'year
     { month " " * year * }
      if$
    }
  if$
}
```

[17] This comment by Oren Patashnik.

New date parsing routines The `format.date` function depends upon a number of supporting functions. Some of these are of general utility, and are presented above. Those presented here are specific to this particular function.

`fillout.a.year` This adds a leading `19` to a year entered in two-digit form.

Expected:	1.	A single string.
Left:	1.	A single string.
Toggles:	1.	none

```
1786 FUNCTION {fillout.a.year}
1787 { duplicate$ character.length #2 =
1788   { "19" swap$ * }
1789   'skip$
1790   if$
1791 }
```

`parse.month` This function is a simple parser, used in converting database entries that have been identified as probable abbreviated month entries into numerical string form.

Expected:	1.	A single string, which should consist of exactly three alphabetic characters.
Left:	1.	A single string of numbers.
Toggles:	1.	none

```
1792 FUNCTION {parse.month}
1793 { duplicate$ "jan" =
1794   { pop$ "1" }
1795   { duplicate$ "feb" =
1796     { pop$ "2" }
1797     { duplicate$ "mar" =
1798       { pop$ "3" }
1799       { duplicate$ "apr" =
1800         { pop$ "4" }
1801         { duplicate$ "may" =
1802           { pop$ "5" }
1803           { duplicate$ "jun" =
1804             { pop$ "6" }
1805             { duplicate$ "jul" =
1806               { pop$ "7" }
1807               { duplicate$ "aug" =
1808                 { pop$ "8" }
1809                 { duplicate$ "sep" =
1810                   { pop$ "9" }
1811                   { duplicate$ "oct" =
1812                     { pop$ "10" }
1813                     { duplicate$ "nov" =
1814                       { pop$ "11" }
1815                       { duplicate$ "dec" =
```

```
                        { pop$ "12" }
                        { "invalid month in " cite$ * warning$
                          "passing text to Camel verbatim" warning$
                          "t" 'scrubdate :=
                        }if$
                      }if$
                    }if$
                  }if$
                }if$
              }if$
            }if$
          }if$
        }if$
      }if$
    }if$
  }if$
}
```

`format.month.name` This takes a numerical string and converts it to either an abbrviated or a spelled-out month name.

Expected:	1.	A toggle string.
	2.	A numerical string.
Left:	1.	A string.
Toggles:	1.	long: The month name placed on the stack will be spelled out.
	2.	short: The month name will be abbreviated.

```
FUNCTION { format.month.name }
{ swap$ duplicate$ empty$
  { pop$ pop$ "" }
  { swap$ "long" =
    { duplicate$ "1" =
      { "January" }
      { duplicate$  "2" =
        { "February" }
        { duplicate$  "3" =
          { "March" }
          { duplicate$ "4" =
            { "April" }
            { duplicate$  "5" =
              { "May" }
              { duplicate$ "6" =
                { "June" }
                { duplicate$  "7" =
                  { "July" }
                  { duplicate$  "8" =
                    { "August" }
                    { duplicate$  "9" =
                      { "September" }
```

```
                    { duplicate$  "10" =
                      { "October" }
                      { duplicate$  "11" =
                        { "November" }
                        { duplicate$  "12" =
                          { "December" }
                          { "invalid month in " cite$ * warning$ ""
                            "passing text to Camel verbatim" warning$
                            "t" 'scrubdate :=
                          }if$
                        }if$
                      }if$
                    }if$
                  }if$
                }if$
              }if$
            }if$
          }if$
        }if$
      }if$
    }
    { duplicate$ "1" =
      { "Jan." }
      { duplicate$  "2" =
        { "Feb." }
        { duplicate$  "3" =
          { "Mar." }
          { duplicate$ "4" =
            { "Apr." }
            { duplicate$  "5" =
              { "May" }
              { duplicate$ "6" =
                { "Jun." }
                { duplicate$  "7" =
                  { "Jul." }
                  { duplicate$  "8" =
                    { "Aug." }
                    { duplicate$  "9" =
                      { "Sept." }
                      { duplicate$  "10" =
                        { "Oct." }
                        { duplicate$  "11" =
                          { "Nov." }
                          { duplicate$  "12" =
                            { "Dec." }
                            { ""
                            }if$
                          }if$
                        }if$
```

```
                        }if$
                      }if$
                    }if$
                  }if$
                }if$
              }if$
            }if$
          }if$
        }if$
      }if$
swap$ pop$
    }if$
}
```

`extract.date` This function takes a labelled stack of numeric items, and places its contents in a formatted string on the stack. It is a subroutine called by `format.date`, and is not intended for use directly in entry type functions.

The behaviour of this function is a little too complex to describe its input and output behaviour using our usual description box, so we revert to prose here. The expectation of what will appear on the stack depends on the contents of the integer variable `itemcount`. For each date item counted, the function expects to find two items on the stack: a string possibly flagging the variable into which the item should be placed; and a numerical string.

If `itemcount` is `0`, we check to see whether the streamlined format for parallel citations has been used. If not, a warning of an empty date is issued.

If `itemcount` is `1`, the flag string is ignored, and the numerical item is assumed to be a year.

If `itemcount` is `2`, the flag strings are both ignored, and the numerical items are assumed to be a year and a month, in that order.

If `itemcount` is `3`, the first item is assumed to be a year. The assignment of the second two items depends first upon the contents of the integer variable `date.specials`. If this is `0`, then the flags are ignored, and the numerical items are assumed to be a day and a month, in that order. If `date.specials` is `1` ('true'), then the flag of the first item is checked to see whether it is "`month`". If so, the first item is assigned to the `themonth` variable, and the next to the `theday` variable. Otherwise the assignments are reversed.

```
FUNCTION {extract.date}
    { 'v := "" 's := "" 't := "" 'u :=
      itemcount #0 =
      { cites empty$
        { "some date or other is COMPLETELY empty in " cite$ *
          warning$ }
        'skip$
        if$
      }
      { itemcount #1 =
        { pop$ fillout.a.year 's := "" 'u := "" 't := }
        { itemcount #2 =
```

```
      { pop$ swap$ pop$ swap$ duplicate$ character.length #4 =
        { swap$ "--" swap$ * * 's := "" 't := }
        { swap$ fillout.a.year 's := 't :=
        }if$
        "" 'u :=
      }
      { itemcount #3 =
        { date.specials
          { pop$ fillout.a.year 's :=
            "month" =
            { 't :=
              pop$ 'u := }
            { 'u :=
              pop$ 't :=
            }if$
          }
          { pop$ fillout.a.year 's :=
            pop$ 'u :=
            pop$ 't :=
          }if$
        }
        { "too many items for date in " cite$ * warning$
        }if$
      }if$
    }if$
    v duplicate$ "month.dd.yy" =
    { pop$ s ", " u "\ " t "long" format.month.name }
    { duplicate$ "dd.month.yy" =
      { pop$ s "\ " t "long" format.month.name "\ " u }
      { duplicate$ "mo.dd.yy" =
        { pop$ s ", " u "\ " t "short" format.month.name }
        { duplicate$ "dd.mo.yy" =
          { pop$ s "\ " t "short" format.month.name "\ " u }
          { duplicate$ "dd.mm.yy" =
            { pop$ s "/" t "/" u }
            { duplicate$ "mm.dd.yy" =
              { s "/" u "/" t }
              { "yy" =
                { "" "" "" "" s }
                { "invalid date toggle in style file" warning$
                }if$
              }if$
            }if$
          }if$
        }if$
      }if$
    }if$
    "" 'v :=
    duplicate$ empty$
```

```
      { pop$ pop$ }
      { swap$ * 'v := }
      if$
      duplicate$ empty$
      { pop$ pop$ }
      { swap$ * v swap$ *  'v := }
      if$
      duplicate$ empty$
      { pop$ v }
      { v swap$ * }
      if$
}
```

`topup.date` This is used to perform an addition operation on a string of numerals.

Expected:	1.	An integer to be added.
	2.	A string consisting of numerals only.
Left:	1.	A string consisting of the sum of the integer and the numeric string.
Toggles:	1.	none

```
FUNCTION {topup.date}
{ 'a :=
  duplicate$ character.length #2 =
  { duplicate$ #1 #1 substring$ chr.to.int$ #48 -
    times.ten swap$ #2 #1 substring$ chr.to.int$ #48 -
    +
  }
  { duplicate$ character.length #1 =
    { chr.to.int$ #48 - }
    { pop$ #0 "I can't cope with more than two Japanese year digits in "
      cite$ * warning$
    }if$
  }if$
a + int.to.str$
}
```

`format.jdate` This rather specialized function converts a Japanese Imperial date written in a fixed syntax into the LexiBib internal date stack, for onward handling by `extract.date`.

Expected:	1.	A string in the form: `s57.9.27`, where the first letter indicates the Imperial reign of the year given, the first numeric item is the year, the second the month, and the third the day. Any non-alphabetic, non-numeric character may be used as a separator.
Left:	1.	`itemcount` is set to `2` (one more is added by the `format.date` function in which is function is nested).
	2.	Six stack items are output—see `extract.date` for details on stack syntax.
Toggles:	1.	none

```
FUNCTION {format.jdate}
{ duplicate$ #2 global.max$ substring$ "*" "forward" gather.chars
  pop$ 't := swap$
  #1 #1 substring$ duplicate$ "s" =
  { pop$ t #1925 topup.date }
  { duplicate$ "m" =
    { pop$ t #1867 topup.date }
    { duplicate$ "t" =
      { pop$ t #1911 topup.date }
      { "h" =
        { t #1988 topup.date }
        { "invalid Imperial calendar code in " cite$ * warning$
        }if$
      }if$
    }if$
  }if$
  swap$
  "*" "forward" gather.chars
  pop$ swap$
  "*" "forward" gather.chars
  pop$ swap$ pop$ 't := swap$ "default" swap$ t swap$ "default"
  swap$ "default"
  #2 'itemcount :=
}
```

`format.date` This function makes use of `gather.chars` and other supporting functions to build a stack of date items, which it then parses using `extract.date`. The syntax for entering dates is described in the user guide.

Expected:	1.	A single string in appropriate date syntax.
Left:	1.	none
Toggles:	1.	none

```
FUNCTION {format.date}
 { 'v :=
   empty.to.null 's :=
```

```
    #0 'charcount :=
    #0 'itemcount :=
    #0 'date.specials :=
    { s empty$ not }
      { s "*" "forward" gather.chars
        duplicate$ "letter" =
        { pop$ duplicate$ character.length #1 = itemcount not and
          { pop$ pop$ s format.jdate "" 's := }
          { duplicate$ character.length #3 =
            { swap$ 's :=
              parse.month "month" #1 'date.specials := }
            { swap$ 's :=
              pop$ "1" "invalid date in " cite$ * warning$
              "passing text to Camel verbatim" warning$
              "t" 'scrubdate :=
              "default"
            }if$
          }if$
        }
        { "numeral" =
          { duplicate$ character.length #1 =
            { swap$ 's := "default" }
            { duplicate$ character.length #2 =
              { swap$ 's := "default" }
              { duplicate$ character.length #4 =
                { swap$ 's := "default" }
                { swap$ 's := pop$ "1"
                  "invalid numerical element in date for " cite$ * warning$
                  "passing text to Camel verbatim" warning$
                  "t" 'scrubdate :=
                  "default"
                }if$
              }if$
            }if$
          }
          { swap$ 's := pop$ "1"
            "failed to parse date in " cite$ * warning$
            "default"
          }if$
        }if$
        itemcount #1 + 'itemcount :=
      }while$
    v extract.date
    scrubdate "t" =
    { pop$ year empty.to.null }
    'skip$
    if$
  }
FUNCTION {parse.one.cite}
{ duplicate$ "=" = not
```

```
{ "=" "forward" gather.chars pop$
  duplicate$ "[" swap$ "forward" first.in.second
  { swap$ duplicate$ "]" swap$ "forward" first.in.second
      { "" 'volume.var :=
        'b := swap$ 'a := a b <
          { duplicate$ a #1 + b a #1 + - substring$
            'year.var :=
            b #1 + global.max$ substring$
            " " "forward" gather.chars pop$
            duplicate$ "*" "forward" gather.chars
            "numeral" =
            { swap$ duplicate$ empty$
              { pop$ pop$ 'number.var := }
              { pop$ pop$ swap$ * "" 'number.var :=
              }if$
            }
            { pop$ pop$ " " swap$ * swap$ * "" 'number.var :=
            }if$
          }
          { "Weird syntax error in " cite$ * warning$
          }if$
      }
      { "Opening [ without closing ] in " cite$ * warning$
      }if$
  }
  { pop$ duplicate$ ")" swap$ "reverse" first.in.second
    { swap$ duplicate$ "(" swap$ "reverse" first.in.second
      { 'a := swap$ 'b := a b <
        { duplicate$ a #1 + b a #1 + - substring$
          'year.var :=
          #1 a #1 - substring$
          " " "forward" gather.chars pop$
          duplicate$ "/" swap$ "forward" first.in.second
            { pop$ "/" "forward" gather.chars pop$
              swap$ duplicate$ empty$
                { pop$ "" 'number.var := }
                { #2 global.max$ substring$
                  'number.var :=
                }if$
            }
            { pop$ "" 'number.var :=
            }if$
          duplicate$ empty$
            { pop$ "" 'volume.var := }
            { duplicate$ "*" "forward" gather.chars
              "numeral" =
              { swap$ duplicate$ empty$
                { pop$ pop$ 'volume.var := }
                { pop$ pop$ swap$ * "" 'volume.var :=
                }if$
```

```
                }
                { pop$ pop$ " " swap$ * swap$ * "" 'volume.var :=
                }if$
              }if$
          }
          { "Weird syntax error in " cite$ * warning$
          }if$
        }
        { "Closing ) without opening ( in " cite$ * warning$
        }if$
      }
      { pop$ "No recognizable date in following string:" warning$
      }if$
    }if$
    " " swap$ * " " "reverse" gather.chars pop$
    duplicate$ "*" "forward" gather.chars
    "numeral" =
    { swap$ duplicate$ empty$
      { pop$ pop$ 'pages.var := }
      { pop$ pop$ swap$ * "" 'pages.var :=
      }if$
    }
    { pop$ pop$ swap$ * "" 'pages.var :=
    }if$
   duplicate$ empty$
   { 'journal.var := }
   { duplicate$ character.length #2 -
     #3 swap$ substring$ 'journal.var :=
   }if$
 }
 'skip$
 if$
}
```

⟨/bstlibrary⟩

⟨*bsttrailer⟩

```
FUNCTION {sortify}
{ purify$
  "l" change.case$
}

INTEGERS { len }

FUNCTION {chop.word}
{ 's :=
  'len :=
  s #1 len substring$ =
    { s len #1 + global.max$ substring$ }
    's
  if$
```

```
}

INTEGERS { et.al.char.used }

FUNCTION {initialize.et.al.char.used}
{ #0 'et.al.char.used :=
}

EXECUTE {initialize.et.al.char.used}

FUNCTION {format.lab.names}
{ 's :=
  s num.names$ 'numnames :=
  numnames #1 >
    { numnames #4 >
        { #3 'namesleft := }
        { numnames 'namesleft := }
      if$
      #1 'nameptr :=
      ""
        { namesleft #0 > }
        { nameptr numnames =
            { s nameptr "{ff }{vv }{ll}{ jj}" format.name$ "others" =
                { "{\etalchar{+}}" *
                  #1 'et.al.char.used :=
                }
                { s nameptr "{v{}}{l{}}" format.name$ * }
              if$
            }
            { s nameptr "{v{}}{l{}}" format.name$ * }
          if$
          nameptr #1 + 'nameptr :=
          namesleft #1 - 'namesleft :=
        }
      while$
      numnames #4 >
        { "{\etalchar{+}}" *
          #1 'et.al.char.used :=
        }
        'skip$
      if$
    }
    { s #1 "{v{}}{l{}}" format.name$
      duplicate$ text.length$ #2 <
        { pop$ s #1 "{ll}" format.name$ #3 text.prefix$ }
        'skip$
      if$
    }
  if$
}
```

```
2233
2234 FUNCTION {author.key.label}
2235 { author empty$
2236     { key empty$
2237         { cite$ #1 #3 substring$ }
2238         { key #3 text.prefix$ }
2239       if$
2240     }
2241     { author format.lab.names }
2242   if$
2243 }
2244
2245 FUNCTION {author.editor.key.label}
2246 { author empty$
2247     { editor empty$
2248         { key empty$
2249             { cite$ #1 #3 substring$ }
2250             { key #3 text.prefix$ }
2251           if$
2252         }
2253         { editor format.lab.names }
2254       if$
2255     }
2256     { author format.lab.names }
2257   if$
2258 }
2259
2260 FUNCTION {author.key.organization.label}
2261 { author empty$
2262     { key empty$
2263         { organization empty$
2264             { cite$ #1 #3 substring$ }
2265             { "The " #4 organization chop.word #3 text.prefix$ }
2266           if$
2267         }
2268         { key #3 text.prefix$ }
2269       if$
2270     }
2271     { author format.lab.names }
2272   if$
2273 }
2274
2275 FUNCTION {editor.key.organization.label}
2276 { editor empty$
2277     { key empty$
2278         { organization empty$
2279             { cite$ #1 #3 substring$ }
2280             { "The " #4 organization chop.word #3 text.prefix$ }
2281           if$
2282         }
```

```
        { key #3 text.prefix$ }
      if$
    }
    { editor format.lab.names }
  if$
}

FUNCTION {calc.label}
{ type$ "book" =
  type$ "inbook" =
  or
    'author.editor.key.label
    { type$ "proceedings" =
        'editor.key.organization.label
        { type$ "manual" =
            'author.key.organization.label
            'author.key.label
          if$
        }
      if$
    }
  if$
  duplicate$
  year "yy" format.date purify$ #-1 #2 substring$
  *
  'label :=
  year "yy" format.date purify$ #-1 #4 substring$
  *
  sortify 'sort.label :=
}

FUNCTION {sort.format.names}
{ 's :=
  #1 'nameptr :=
  ""
  s num.names$ 'numnames :=
  numnames 'namesleft :=
    { namesleft #0 > }
    { nameptr #1 >
        { "   " * }
        'skip$
      if$
      s nameptr "{vv{ } }{ll{ }}{  ff{ }}{  jj{ }}" format.name$ 't :=
      nameptr numnames = t "others" = and
        { "et al" * }
        { t sortify * }
      if$
      nameptr #1 + 'nameptr :=
      namesleft #1 - 'namesleft :=
    }
```

```
  while$
}

FUNCTION {sort.format.title}
{ 't :=
  "A " #2
    "An " #3
      "The " #4 t chop.word
    chop.word
  chop.word
  sortify
  #1 global.max$ substring$
}

FUNCTION {author.sort}
{ author empty$
    { key empty$
        { "to sort, need author or key in " cite$ * warning$
          ""
        }
        { key sortify }
      if$
    }
    { author sort.format.names }
  if$
}

FUNCTION {case.sort}
{ title empty$
    { key empty$
        { "to sort, need title or key in " cite$ * warning$
          ""
        }
        { key sortify }
      if$
    }
    { title sort.format.title }
  if$
}

FUNCTION {author.editor.sort}
{ author empty$
    { editor empty$
        { key empty$
            { "to sort, need author, editor, or key in " cite$ * warning$
              ""
            }
            { key sortify }
          if$
        }
```

```
        { editor sort.format.names }
      if$
    }
    { author sort.format.names }
  if$
}

FUNCTION {author.organization.sort}
{ author empty$
    { organization empty$
        { key empty$
            { "to sort, need author, organization, or key in " cite$ * warning$
              ""
            }
            { key sortify }
          if$
        }
        { "The " #4 organization chop.word sortify }
      if$
    }
    { author sort.format.names }
  if$
}

FUNCTION {editor.organization.sort}
{ editor empty$
    { organization empty$
        { key empty$
            { "to sort, need editor, organization, or key in " cite$ * warning$
              ""
            }
            { key sortify }
          if$
        }
        { "The " #4 organization chop.word sortify }
      if$
    }
    { editor sort.format.names }
  if$
}

FUNCTION {presort}
{ type$ "book" =
  type$ "inbook" =
  or
    'author.editor.sort
    { type$ "proceedings" =
        'editor.organization.sort
        { type$ "manual" =
            'author.organization.sort
```

```
            { type$ "case" =
              'case.sort
              { type$ #2 global.max$ substring$ "statute" =
                'case.sort
                'author.sort
              if$
              }
            if$
            }
          if$
        }
      if$
    }
  if$
  "    "
  *
  year "yy" format.date sortify
  *
  "    "
  *
  title empty.to.null
  sort.format.title
  *
  #1 entry.max$ substring$
  'sort.key$ :=
}

ITERATE {presort}

SORT

FUNCTION {begin.bib}
{ et.al.char.used
    { "\newcommand{\etalchar}[1]{$^{#1}$}" write$ newline$ }
    'skip$
  if$
  preamble$ empty$
    'skip$
    { preamble$ write$ newline$ }
  if$
}

EXECUTE {begin.bib}

ITERATE {call.type$}
```
⟨/bsttrailer⟩

5 A .bib file

```
⟨*bib⟩
@book{bluebook,
 title = {A Uniform System of Citation},
 edition = {19th},
 year = 1989
}
@book{knuth,
 title = {The {\TeX} book},
 year = 1990,
 author = {D.E. Knuth}
}
@book{leunen,
 author = {M.-C. van Leunen},
 title = {A Handbook for Scholars},
 year = 1979
}
@book{latex-book,
 title = {\LaTeX: a document preparation system},
 year = 1994,
 author = {Leslie Lamport}
}
@book{latex-companion,
 author = {M. Goosens and Frank Mittelbach and A. Samarin},
 title = {The \LaTeX\ Companion},
 year = 1994
}
@techreport{oren-user,
 author = {Oren Patashnik},
 title = {\BibTeX ing},
 year = 1988,
 type = {CTAN document}
}
@techreport{oren-hackers,
 author = {Oren Patashnik},
 title = {Designing \BibTeX\ Styles},
 type = {CTAN document},
 year = 1988
}
@book{maki-constitution,
 author = {Maki, John McGilvrey},
 title = {Court and Constitution in Japan: Selected Supreme Court Decisions,
       1948-1960},
 publisher = {Washington University Press},
 year = 1964
}
@booklet{sansom-constitution,
 author = {Sansom, {Sir} George Bailey},
 title = {The First Japanese Constitution: A Lecture},
 note = {delivered before the
       Asiatic Society of Japan},
```

```
 year = 1938
}
@incollection{haley-land-lease,
 title = {Japan's New Land and House Lease Law},
 author = {John Owen Haley},
 crossref = {policy-failure},
 pages = 149,
}
@book{policy-failure,
 title = {Land Issues in Japan: A Policy Failure?},
 booktitle = {Land Issues in Japan: A Policy Failure?},
 year = 1992
}
@article{appleyard-heed,
 author = {Bryan Appleyard},
 title = {Why Major should heed the press},
 pages = 21,
 journal = {The Independent},
 date = {7 jul 1995},
 place = {London}
}
@statute{sga,
 title = {Sale of Goods Act},
 year = {1979},
 jurisdiction = {england}
}
@case{heap,
 title = {Heap v Motorists' Advisory Agency Ltd},
 year = 1923,
 journal = {KB},
 number = 1,
 pages = 577
}
@case{halsall-v-brizell,
 title = {Halsall v Brizell},
 cites = {[1957] Ch 169 = [1957] 1 All ER 371}
}
```
⟨/bib⟩

6 The Driver File

```
⟨*installer⟩
\def\batchfile{camel.ins}
\input docstrip.tex
\preamble

Copyright (C) 1992--95 Frank Bennett, Jr.
All rights reserved.

```

```
This file is part of the Camel package.
\endpreamble

\def\batchfile{camel.dst}      % ignored in distribution
\input docstrip.tex              % ignored in distribution

\Ask\answer{%
     ************************************************************
  ^^J*                       PLEASE NOTE
  ^^J*
  ^^J* Camel requires a large BibTeX, with a wizard-defined
  ^^J* function space of over 6,000.  The function space in
  ^^J* standard BibTeX is only 3,000, so if your BibTeX has never
  ^^J* been enlarged, it will probably need to be recompiled.
  ^^J*
  ^^J* If this seems terribly cryptic, just go ahead and give
  ^^J* it a go.  Feel free to write me on fb@soas.ac.uk if you
  ^^J* have problems.
  ^^J*
  ^^J* Also, to print the .dtx files in the Camel bundle, you
  ^^J* should use the usual gind.ist file for making the indexes;
  ^^J* camel.ist is used for special purposes outlined in the
  ^^J* manual.
  ^^J*
  ^^J* Shall I unpack my things?
  ^^J*
  ^^J* If you want me to go ahead, answer 'y' below, otherwise 'n'.
  ^^J*
  ^^J************************************************************}

\keepsilent

\preamble

This file is part of the Camel package.
---------------------------------------
This is a generated file.

IMPORTANT NOTICE:

You are not allowed to change this file.  You may however copy
this file to a file with a different name and then change the
copy if (a) you do not charge for the modified code, (b) you
acknowledge Camel and its author(s) in the new file, if it
is distributed to others, and (c) you attach these same
conditions to the new file.

You are not allowed to distribute this file alone.  You are not
allowed to take money for the distribution or use of this file
(or a changed version) except for a nominal charge for copying
```

```
etc.

You are allowed to distribute this file under the condition that
it is distributed with all of its contents, intact.

For error reports, or offers to help make Camel a more powerful,
friendlier, and better package, please contact me on
`fb' at soas.ac.uk

\endpreamble


{\ifx\answer\y


\Msg{*** Generating LaTeX style file (.sty) ***}

\generateFile{camel.sty}   {t}{\from{camel.dtx}{lexitex}}


\Msg{*** Generating makeindex style file (.ist) ***}

\generateFile{camel.ist}   {t}{\from{camel.dtx}{camelindex}}


\Msg{*** Generating BibTeX data file for the manual (.bib) ***}

\generateFile{camel.bib}   {t}{\from{camel.dtx}{bib}}


\Msg{*** Generating sample text file to mess around with (.tex) ***}

\generateFile{test.tex}{t}{\from{camel.dtx}{testtex}}


\Msg{*** Generating parser demonstration file (.tex) ***}

\generateFile{parsdemo.tex}{t}{\from{camel.dtx}{parseonly}}

\fi}

\keepsilent


\ifToplevel{
\Msg{************************************************************}
\Msg{*}
\Msg{* To finish the installation you have to move the following}
\Msg{* file into a directory searched by TeX:}
\Msg{*}
```

```
\Msg{* \space\space camel.sty}
\Msg{*}
\Msg{* You also need to move the following file into a directory}
\Msg{* searched by makeindex:}
\Msg{*}
\Msg{* \space\space camel.ist}
\Msg{*}
\Msg{* Note that Camel does not work by itself; style modules}
\Msg{* (such as law.dtx) are supplied separately.  They can}
\Msg{* be found on CTAN in the ‘styles’ subdirectory below}
\Msg{* Camel itself.}
\Msg{*}
\Msg{*************************************************************}
}
⟨/installer⟩
⟨*testtex⟩
\documentclass{article}
\usepackage{camel}

⟨+testtex⟩%\citationsubject[o=sta,i=stb]{statutes}{Statutory Materials}
⟨+testtex⟩%\citationsubject[o=sec,i=seb]{second}{Secondary Literature}
⟨+testtex⟩%\citationsubject[o=cas,i=cab]{case}{Cases}

\begin{document}
\citationstyle{law}
\citationdata{camel}

\section*{{\sc Camel} tests and examples\footnotemark{}{} }
\footnotetext{Note that, in addition to admiring the examples in
  this document, you can tinker with it to produce different
  kinds of bibliographies.  See the comments in the file for
  suggestions.}


We currently have support for articles (including newspaper
articles), items in collections of
essays, books, sections of books, ephemeral booklets, masters theses,
Commonwealth, US and Japanese cases, and statutes from a few
jurisdictions.  An example of each is given below, first without,
then with a pinpoint.  This does not give a complete picture of
the ‘formatting tree’, it’’s just a sample.

\begin{itemize}
\item Ordinary articles
  \begin{itemize}
  \item \source[s=second]{macauley}
  \item \source[fs=second]{macauley}[56]
  \end{itemize}
\item Newspaper articles
```

```
  \begin{itemize}
  \item \source[s=second]{appleyard-heed}
  \item \source[fs=second]{appleyard-heed}[21] (A silly example,
    since there's only one page to the piece!)
  \end{itemize}
\item Articles in collections of essays
  \begin{itemize}
  \item \source[s=second]{haley-land-lease}
  \item \source[fs=second]{haley-land-lease}[152]
  \end{itemize}
\item Books
  \begin{itemize}
  \item \source[fs=second]{latex-companion}
  \item \source[fs=second]{latex-companion}[371]
  \end{itemize}
\item Sections of books
  \begin{itemize}
  \item \source[fs=second]{companion-bibs}
  \item \source[fs=second]{companion-bibs}[374-75] (The BibTeX{}
    processing flow shown here is simplified for {\sc Camel} users)
  \end{itemize}
\item Ephemeral booklets
  \begin{itemize}
  \item \source[fs=second]{sansom-constitution}
  \item \source[fs=second]{sansom-constitution}[2]
  \end{itemize}
\item Masters theses
  \begin{itemize}
  \item \source[fs=second]{homma-derivative}
  \item \source[fs=second]{homma-derivative}[23]
  \end{itemize}
\item Commonwealth law cases
  \begin{itemize}
  \item \source[fs=case]{heap}
  \item \source[fs=case]{heap}[578]\footnote{Notice how the
      pinpointed citation gives only as many parallel as are
      specified in the pinpoint.  Compare this with the next example.}
  \end{itemize}
\item US law cases
  \begin{itemize}
  \item \source[fs=case]{bradshaw-v-us}
  \item \source[fs=case]{bradshaw-v-us}[145=366]
  \end{itemize}
\end{itemize}
Titles that end in a numeral are a special treat.  {\sc Camel}
is not tricked by them.
\begin{itemize}
  \item \source[s=statutes]{maki-constitution}[23]
  \item \source[fs=statutes]{maki-constitution}
\end{itemize}
```

```
2774
2775 ⟨+testtex⟩% If you uncomment the \citationsubject lines above,
2776 ⟨+testtex⟩% you can comment out the 'all' entry below, run
2777 ⟨+testtex⟩% makeindex in the appropriate way over the output
2778 ⟨+testtex⟩% files, uncomment the three special bibliography
2779 ⟨+testtex⟩% declarations below, and print a whole different sort
2780 ⟨+testtex⟩% of document.  There are a few bugs in the way
2781 ⟨+testtex⟩% bibliographies are set up.  Sorting them out will
2782 ⟨+testtex⟩% best be done once BibTeX 1.0 is available, since
2783 ⟨+testtex⟩% integrating with the new BibTeX will affect the same
2784 ⟨+testtex⟩% portions of the code.
2785 \printbibliography{all}
2786 ⟨+testtex⟩%\printbibliography{statutes}
2787 ⟨+testtex⟩%\printbibliography{case}
2788 ⟨+testtex⟩%\printbibliography{second}
2789 \end{document}
2790 ⟨/testtex⟩
```

The following code makes a demo file for the parsing routines contained in CAMEL. Just a sales gimmick.

```
2791 ⟨*parseonly⟩
2792 \documentclass{article}
2793 \begin{document}
2794 \makeatletter
2795 \def\@demo@plone#1#2{\def\@demo@listitem{#1}%
2796   \ifx\@demo@optionitem\@demo@listitem #2%
2797   \let\\=\@demo@finish\fi}
2798 \def\@demo@end{,}
2799 \def\@demo@finish#1\@demo@nomatch{}
2800 \def\@demo@plnomatch{\@latex@error{Invalid
2801     option `\@demo@optionitem'
2802     given to \string\parsedemo.^^J%
2803    Valid options are:^^J
2804     a (reports selection of `a' to terminal)^^J
2805     b=<string> (reports selection of `b=' and the option string^^J
2806     to terminal)^^J%
2807 To avoid errors in formatting, use commas in the option string^^J%
2808 only where they are logically necessary to the sense of the^^J%
2809 string.  Therefore \string\parsedemo{ab=birds} and
2810 \string\parsedemo{b=birds,a}^^J%
2811 are correct, while \string\parsedemo{a,b=birds} is NOT
2812 correct}\@ehc}
2813 \def\@demo@sourceoptionlist{%
2814  \\{a}{\message{(`a' was selected!)}}%
2815  \\{b}{\gdef\@demo@parsemacro##1{\message{(string option to `b='
2816        was `##1'!)}}%
2817         \let\@demo@parse\@demo@parsecomma}%
2818  \\{=}{\let\@demo@parse\@demo@parsebumequal}%
2819  \\{,}{}%
2820  \\{\end}{\let\@demo@nomatch\relax\let\@demo@parse\relax}%
```

```
    \@demo@nomatch}
\def\@demo@parsecomma=#1,{%
  \@demo@parsemacro{#1}\let\@demo@parse\@demo@parseplain%
  \@@parse}
\def\@demo@parsebumequal#1,{\let\@demo@parse\@demo@parseplain\@@parse}
\def\@demo@parseplain#1{%
  \let\@demo@nomatch=\@demo@plnomatch%
  \def\@demo@optionitem{#1}\let\\=\@demo@plone\@demo@sourceoptionlist\@@parse}
\def\@@parse{\@demo@parse}
\def\parsedemo#1{%
 {\let\@demo@parse=\@demo@parseplain\@demo@parse #1,\end}
}
\makeatother
\tracingmacros=2
\parsedemo{a,b=Whoopie!}
\end{document}
```

⟨/parseonly⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

Change History

LexiBib 1.0b

"General": Added swap$ as appropriate to expose string on stack for empty$ check. 75

1.0e

"General": Altered format.pages so that the short scan stops at a - character, rather than a non-integer. Simpler and more robust. 71

1.0m

"General": New function

`type.last.char` added, to cope with titles ending in a numeral, hence exceptionally requiring a comma in the Blue Book style. 63