

The `cleveref` package*

Toby Cubitt
`toby-cleveref@dr-qubit.org`

26/05/2007

Abstract

The `cleveref` package enhances \LaTeX 's cross-referencing features, allowing the format of references to be determined automatically according to the type of reference (equation, section, etc.). The formatting for each reference type can be fully customised in the preamble of your document. In addition, `cleveref` can typeset references to lists of multiple labels, automatically formatting them according to their type, and collapsing sequences of numerically consecutive labels to a reference range. Again, the multiple-reference formatting is fully customisable.

1 Introduction

When “clever” is used in the name of a computer program, it usually signifies that the programmer is overly smug! Or, to take a more charitable attitude, that the programmer is pleased with the sophisticated processing that he has managed to encapsulate in code, relieving the user of some complicated by tedious task. On the other hand, at the heart of the \LaTeX philosophy is the idea that it is clever to delegate as much of the typesetting as possible to the computer, in order to achieve a beautiful, but above all consistent, visual appearance.

Both these points of view are valid with respect to the `cleveref` package. Its goals are two-fold: to use the information that \LaTeX inherently has about labels as intelligently as possible in order to type-set references to them (clever processing); and to enable you to produce an attractive, consistent formatting of references throughout your document, with the minimum of effort (you’d be clever to use it!).

The `cleveref` package enhances \LaTeX 's cross-referencing facility by allowing references to be formatted automatically according to the type of object they refer to (chapter, section, equation, theorem, etc.). It can also automatically format references to multiple labels.

In standard \LaTeX , you have almost certainly found yourself writing things like `Eq.~(\ref{eq1})` and `Theorem~\ref{thm1}` over and over again. This isn’t just tedious. What happens if you later decide you want equation references to be typeset as `Equation~\ref{eq1}` instead? What happens if you decide to change the theorem labelled `thm1` into a lemma? You have to search through the entire

*This document corresponds to `cleveref` 0.4, dated 26/05/2007.

L^AT_EX source of your document, modifying all references to equations, or changing all references to `thm1`.

`cleveref` allows you to define the format for references once-and-for-all in the preamble of your document. If you later decide to change the typesetting of equation references, you only have to change one preamble definition. If you later decide to change a theorem into a lemma, you don't need to change any references at all, because `cleveref` will automatically typeset references to it using the appropriate formatting. This makes it far easier to typeset references uniformly across your whole document, as well as saving repetitively typing the same text for each and every reference.

A number of other packages provide similar features, most notably `varioref`, `fancyref` and (for theorem-like environments) `ntheorem` (with the `thref` option). However, all have certain deficiencies which `cleveref` attempts to overcome.

The `fancyref` package doesn't automatically determine the type of object being referred to. Instead, it relies on you adhering to a naming convention for labels. This is usually a good idea in any case, but it can be inconvenient. For example, if you change a theorem into a lemma, you have to change the label name, and therefore also all references to it. So you are back to searching and replacing through the entire document.

The enhanced referencing feature provided by the `varioref` package decides how to format references when the label is *defined*, rather than when it is *referenced*. Most of the time, this isn't a problem. But it makes it impossible to format references differently according to the context in which they are referenced, which can sometimes be very useful. For example, if you want references at the beginning of a sentence formatted any other way than by capitalising the first letter of the reference text, it is impossible using `varioref`. Perhaps even more significantly, it makes it impossible to typeset multiple references automatically; you are back typesetting `Eqs.~(\ref{eq1}) and~(\ref{eq2})` or `Eqs.~(\ref{eq1})--(\ref{eq3})` by hand.

The `ntheorem` package (with the `thref` option) does things right... except that it only works for theorem-like environments. It is possible to use it for other environments, but only in a bastardized form, by manually supplying an optional argument to `\label` commands telling it the label type. `cleveref` essentially extends the `ntheorem` theorem referencing feature to work for all objects. (Although the implementation details differ, and `cleveref` can be used independently of `ntheorem`.) `cleveref` provides a number extra features too, such as multiple references, automatic collapsing of reference ranges, and control over the placement of hyper-links.

2 Usage

The `cleveref` package is loaded in the usual way, by putting the line

```
\usepackage{cleveref}
```

in your document's preamble. However, care must be taken when using `cleveref` in conjunction with other packages that modify L^AT_EX's referencing system (see section 3).

2.1 Typesetting References

`\cref` To automatically typeset a reference according to the type of object referred to, simply refer to it using `\cref{<label>}`. `cleveref` imposes just one restriction on the names of labels: they are no longer allowed to contain commas “,”. These can instead be used to typeset multiple references (see below).

`\Cref` As it is very difficult for L^AT_EX to determine whether a reference appears at the beginning of a sentence or not, a capitalised version exists: `\Cref{<label>}`. By default, this typesets the reference with the first letter capitalised, though the formatting of the `\cref` and `\Cref` forms can be customised independently (see section 2.2).

`\ref` `cleveref` does *not* modify the standard `\ref` command¹, so you can still use it to typeset the formatted label counter alone, without any additional text or formatting. If you would like to use `cleveref`’s enhanced features, but want to continue to use `\ref` for referencing, simply redefine the `\ref` command in your document’s preamble, thus:

```
\renewcommand{\ref}{\cref}
```

Similarly, if you decide you would like all references capitalised, you can redefine the `\cref` command:

```
\renewcommand{\cref}{\Cref}
```

`\crefrange` To typeset a reference range, e.g. Eqs.~(1.1)–(1.5), use `\crefrange` or `\Crefrange` (depending on the capitalisation you require), which take the beginning and end of the range as arguments:

```
\crefrange{<label1>}{<label2>}
```

`\cref` To typeset multiple references, simply list the labels inside the `\cref` or `\Cref` command, separated by commas (you are not allowed to use commas in label names when using `cleveref`):

```
\cref{<label1>,<label2>,<label3>,...}
```

The references will be typeset in the order in which they appear in the list, and sequences of consecutive references will be collapsed into a reference range. It is up to you to put the labels in the order you require. This is especially significant if the labels refer to different types of object, or to consecutive sequences of labels. For example, if `eq*` are equation labels, and `thm*` are theorem labels,

```
\cref{eq1,eq2,eq3,thm1,thm2}
```

will be typeset as

Eqs. (1)–(3), and Theorems 1 and 2

whereas

```
\cref{eq1,eq2,thm1,thm2,eq3}
```

will be typeset as

Eqs. (1) and (2), Theorems 1 and 2, and Eq. (3)

¹This is not quite true. The original `\ref` command no longer works when `cleveref` is loaded, so `cleveref` redefines it to recover the correct behaviour.

2.2 Customising the Reference Formats

2.2.1 Single References

The `cleveref` package allows you to take full control of the typesetting of references. Defaults appropriate for English documents are provided for the standard reference types. But if you don't like them, or are writing in a different language, or you are using an environment for which no default format is defined, you will need to define your own.

If `cleveref` encounters a reference to a type it does not know, it will produce a “reference type undefined” warning, and typeset the reference as

`?? \ref{<label>}`

i.e. it will typeset the reference as the label counter preceded by a double question mark. References to undefined labels still produce a “reference undefined” warning and are typeset as a double question mark, as usual.

`\crefformat` Reference formats for *single* references are defined or redefined using the
`\Crefformat` `\crefformat` and `\Crefformat` commands, which are used by the `\cref` and
 `\Cref` commands respectively. These take two arguments: the reference type, and
 the formatting code:

`\crefformat{<type>}{<format>}`
`\Crefformat{<type>}{<format>}`

If the corresponding `\Crefformat` is undefined when `\crefformat` is called, it will define the `\Crefformat` to produce a capitalised version of `\crefformat`, using `\MakeUppercase`. Conversely, if the corresponding `\crefformat` is undefined when `\Crefformat` is called, it will define the `\crefformat` to produce a lower-case version of `\Crefformat`, using `\MakeLowercase`. Obviously, this will only work properly if the formats have a letter at the start. If the first letter is a special construct, such as an accented character, you will need to surround everything whose case should be changed by braces. If the first thing in the format is not a letter (e.g. if it is a `LATEX` command, or a hyper-link argument, see below), you will get strange and fatal errors when processing the document.

The reference `<type>` is usually the name of the counter for the environment (equation, chapter, section, etc.). Currently, the only exception is theorem-like environments when the `ntheorem` package is loaded, for which `<type>` should instead be the environment name (lemma, corollary, definition, etc.). This is because `ntheorem` allows different theorem-like environments to use a common counter.

The `<format>` argument can be any valid `LATEX` code, though you will need to `\protect` fragile commands. It can (and almost certainly should!) contain three arguments, `#1`, `#2` and `#3`. The first argument is the formatted version of the label counter (e.g. `\theequation`). The other two are used to mark the beginning and end of the part of the reference that should form the hyper-link when the `hyperref` package is used (see section 3). The hyper-link arguments `#2` and `#3` *must* appear in that order. Leaving them out completely will not cause an error, but will mean that no hyper-link will be created when `hyperref` is used.

As an example,

`\crefformat{equation}{Eq.~(#2#1#3)}`

will typeset equation references as

Eq. ($\langle counter \rangle$)

with the counter (excluding the brackets) forming the hyper-links. Note that the hyper-link arguments are *not* letters, so if you want to make the “Eq.” part of the hyper-link by putting #2 at the start of the formatting code, you will have to define both capitalisation variants independently:

```
\crefformat{equation}{#2eq.~(#1)#3}
\Crefformat{equation}{#2Eq.~(#1)#3}
```

2.2.2 Reference Ranges

`\crefrangeformat` The format for reference ranges is defined by `\crefrangeformat` and `\Crefrangeformat`. You do not have to define it if you don’t plan to use any reference ranges, i.e. you never use the `\crefrange` and `\Crefrange` commands, and you never use multiple references (see section 2.2.3). However, you will get “reference format undefined” errors if you later change your mind and forget to define the format. Like `\crefformat` and `\Crefformat`, the commands take two arguments: the reference type, and the formatting code.

```
\crefrangeformat{<type>}{<format>}
\Crefrangeformat{<type>}{<format>}
```

The same comments apply: the $\langle type \rangle$ is usually the name of the counter, except for theorem-like environments, and if the corresponding other-capitalisation variant is not already defined, it will be defined automatically.

The $\langle format \rangle$ argument can again be any valid L^AT_EXcode, with fragile commands `\protected`. However, this time it should contain *six* arguments, #1–#6. The first two (#1 and #2) are the formatted versions of the label counters, the next two (#3 and #4) are used to mark the beginning and end of the hyper-link for the first reference (#1), and the final two (#5 and #6) mark the beginning and end of the second reference’s hyper-link.

As an example,

```
\crefrangeformat{equation}{Eqs.~(#3#1#4)--(#5#2#6)}
```

will typeset equation reference ranges as

Eqs. ($\langle counter1 \rangle$)–($\langle counter2 \rangle$)

with the counters (excluding the brackets) forming the hyper-links.

2.2.3 Multiple References

`\crefmultiformat` The format for multiple references is defined by `\crefmultiformat` and `\Crefmultiformat`, and that of reference ranges within multiple references by `\crefrangemultiformat` and `\Crefrangemultiformat`. If you don’t use any multiple references for a particular reference type, you are not obliged to define the multi-formats. However, you will get “reference format undefined” errors if you later decide to use a multiple reference without ensuring the multi-formats are defined. You will also need to define *all* the other reference formats (see sections 2.2.1 and 2.2.2), including the reference range formats, even if you never use the `\crefrange` and `\Crefrange` commands.

The commands all take four arguments: the reference type, the format for the first reference in a list, the format for the middle references in a list, and the format for the last reference in a list.

```
\crefmultiformat{<type>}{<first>}{<middle>}{<last>}
\Crefmultiformat{<type>}{<first>}{<middle>}{<last>}
\crefrangemultiformat{<type>}{<first>}{<middle>}{<last>}
\Crefrangemultiformat{<type>}{<first>}{<middle>}{<last>}
```

The same considerations apply to the `<type>` and formatting arguments `<first>`, `<middle>` and `<last>` as for the `<format>` argument of `\crefformat` and `\Crefformat` or `\crefrangeformat` and `\Crefrangeformat`, including the meaning of the arguments that should appear in the formatting code. However, when the corresponding other-capitalisation variant is automatically defined, only the first letter of the `<first>` argument is upper- or lower-cased.

Be careful to get the spaces at the beginning and end of the formatting code correct: the `<first>`, `<middle>` and `<last>` L^AT_EX code is typeset one after another in a multi-reference, with no space separating them. You may or may not want spaces at the beginning and end of the formatting code, depending on the formatting you desire. For example, with the default format

```
\crefmultiformat{equation}%
{Eqs.~(#2#1#3)}{, (#2#1#3)}{ and~(#2#1#3)}
```

the `<middle>` arguments should *not* have a space at the beginning, whereas the `<last>` arguments *should* have a space.

```
\crefmiddleconjunction
\creflastconjunction
```

The `\crefmiddleconjunction` and `\creflastconjunction` commands are used in between sub-lists of different types of references that occur in a multiple reference. If there are two or more reference types in the list, `\creflastconjunction` is used between the penultimate and final group of references, whereas `\crefmiddleconjunction` is used between all others. You can redefine them with `\renewcommand` in the usual way:

```
\renewcommand{\crefmiddleconjunction}{<conjunction>}
\renewcommand{\creflastconjunction}{<conjunction>}
```

For example, if `eq*`, `thm*` and `fig*` are respectively equation, theorem and figure labels,

```
\cref{eq1,eq2,eq3,thm1,thm2,fig1,thm3}
```

is typeset (assuming the default `\crefformats`) as

```
Eqs. (1)–(3)\crefmiddleconjunction Theorems 1
and 2\crefmiddleconjunction Fig. 1\creflastconjunction Theo-
rem 3
```

3 Interaction with Other Packages

Since `cleveref` redefines many internal commands involved in L^AT_EX's referencing system, it can interact badly with other packages that do the same. In general, `cleveref` should be loaded *last*, and the appropriate options must be supplied if the `hyperref` and/or `ntheorem` packages are also used. E.g.

```
\usepackage{hyperref}
\usepackage[hyperref]{cleveref}
```

or

```
\usepackage{ntheorem}
\usepackage[ntheorem]{cleveref}
```

or

```
\usepackage{hyperref}
\usepackage[hyperref]{ntheorem}
\usepackage[hyperref,ntheorem]{cleveref}
```

`varioref`'s enhanced referencing features (which you make use of by via the `\labelformat` command), the `fancyref` package, and `ntheorem`'s `thref` option are incompatible with `cleveref`. However, since `cleveref` implements an enhanced version of these features, this is not really a problem. For example, if you have a pre-existing document that uses `ntheorem`'s `\thref` command, you can simply redefine it to call `\cref` instead:

```
\renewcommand{\thref}{\cref}
```

Note that you can still use the other features of `varioref` and `ntheorem` whilst using `cleveref`.

Other packages which alter the L^AT_EX referencing system are unlikely to work properly with `cleveref`.

4 Known Bugs

- `cleveref` will not work properly with the standard L^AT_EX `eqnarray` environment. The `eqnarray` environment is poorly implemented, making it somewhat difficult to get it to work properly with `cleveref`. You're better off using the `amsmath` replacements in any case, such as `gather`, `align`, `multline` and `split`, which *do* work properly with `cleveref`. (See <http://www.tug.org/pracjourn/2006-4/madsen/>).
- `cleveref`'s referencing features currently may or may not work properly with references to list items. (This should be fixed in a future version.)
- `cleveref` breaks `hyperref`'s `backref` option, and probably also the `backref` package when used by itself. (This should be fixed in a future version.)
- `cleveref` assumes that counters are only ever reset by the standard sectioning commands (chapter, section, etc.). If this is not the case, the automatic collapsing of consecutive references into a reference range may be incorrect.
- `cleveref` provides no `babel` support.

5 Future Improvements

- Allow selective disabling of reference-range collapsing in multiple references, e.g. by the addition of an extra comma between the labels in the label list.
- Allow reference-range collapsing to be disabled entirely via a package option
- Implement a “poor-man’s” `cleveref`, for situations in which you are able to install and use `cleveref` locally, but another party can not (e.g. when submitting the \LaTeX source of a paper to a journal). When this option is set, `cleveref` would generate a sed or awk script through which the original source could be processed, generating a version of the source with all the `cleveref` references replaced by the text they would generate plus standard `\ref` commands.

6 Implementation

Essentially, the core of the implementation consists of causing an extra piece of information – the label “type” – to be written to the aux file, and defining an `\cref` command which uses this extra information to typeset the reference.

The least invasive implementation seems to be that used by the `varioref` package. Namely, to redefine the `\refstepcounter` command so that the `\@currentlabel` macro, which usually just contains the typeset version of the counter, now contains the additional information (in fact, we write three extra pieces of information: the type, the counter value itself, and the formatted version of the counter that causes the label’s counter to be reset, which we call the “prefix” from now on). `\@currentlabel` eventually gets written to the aux file as an argument to `\newlabel` by the usual \LaTeX mechanisms. This involves less hacking to get everything else working again, since very few macros other than `\ref` use this particular `\newlabel` argument (nor are other packages likely to, given that `varioref` is a required package).

6.1 Redefinitions of \LaTeX kernel macros

We store the original `\refstepcounter` in `\old@refstepcounter`, then redefine `\refstepcounter` so that it first calls the old version and then adds the extra information to `\@currentlabel`. The new `\@refstepcounter` can take an optional argument, which overrides using the counter name as the “type” and instead uses whatever is supplied.

```
1 \let\old@refstepcounter\refstepcounter
2 \def\refstepcounter%
3   {\@ifnextchar[{\refstepcounter@optarg}{\refstepcounter@noarg}}%
4 \def\refstepcounter@noarg#1{%
5   \old@refstepcounter{#1}%
6   \reset@by{#1}{\@result}%
7   \ifx\@result\relax\def\@result{}%
8   \else\edef\@result{\csname the\@result\endcsname}\fi%
9   \protected@edef\@currentlabel{%
10     [#1][\arabic{#1}][\@result]\@currentlabel}%
11 \def\refstepcounter@optarg[#1]#2{%
12   \old@refstepcounter{#2}%
```



```

13 \resetby{#2}{\@result}%
14 \ifx\@result\relax\def\@result{%
15 \else\edef\@result{\csname the\@result\endcsname}\fi%
16 \protected@edef\@currentlabel{%
17 [#1][\arabic{#2}][\@result]\@currentlabel}}

```

The standard `\ref` macro spits out whatever was in `\@currentlabel` when the label was written to the aux file, but this now contains the additional type information which we don't want. Therefore, we redefine `\cref` to recover the original behaviour. We have to defer redefinition of `\ref` till the beginning of the document, in case other packages (such as `ntheorem`) modify it after `cleveref` is loaded.

```

18 \def\cref@label#1#2{\@result}
19 \AtBeginDocument{%
20 \def\ref#1{%
21 \expandafter\ifx\csname r@#1\endcsname\relax%
22 \let\@result\relax%
23 \else%
24 \cref@getlabel{#1}{\@result}%
25 \fi%
26 \expandafter\setref\csname r@#1\endcsname{\cref@label}{#1}}%
27 }

```

6.2 Utility Macros

Define some utility macros for extracting label, type, and counter information from the contents of `\@currentlabel`, as written to the aux file and stored in `\r@{label}` when this is re-read on the next pass. Some other packages commandeer the referencing system to write label information to the aux file for other purposes, and probably use `\ref` to recover it later. We still want them to work, so our utility macros must cope with the type information being absent. However, since we need them to be fully expandable in various places, and `\ifnextchar` is definitely *not* fully expandable, we use the work-around of having the macros store their result in another macro, whose name is passed as the second argument. This macro *will* then be fully expandable, and can be used e.g. inside an `\edef` or `\csname... \endcsname`.

```

28 \def\cref@getlabel#1#2{%
29 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
30 \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
31 \expandafter\cref@getlabel\@tempa\@nil#2}
32 \def\@cref@getlabel{\ifnextchar[%]
33 \@@cref@getlabel{\@cref@getlabel [] [] []}}
34 \def\@@cref@getlabel[#1][#2][#3]#4\@nil#5{\def#5{#4}}
35 \def\cref@gettype#1#2{%
36 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
37 \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
38 \expandafter\cref@gettype\@tempa\@nil#2}
39 \def\@cref@gettype{\ifnextchar[%]
40 \@@cref@gettype{\@cref@gettype [] [] []}}
41 \def\@@cref@gettype[#1][#2][#3]#4\@nil#5{\def#5{#1}}
42 \def\cref@getcounter#1#2{%
43 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
44 \edef\@tempa{\expandafter\@firstoftwo\@tempa}%

```

```

45 \expandafter\@cref@getcounter\@tempa\@nil#2}
46 \def\@cref@getcounter{\@ifnextchar[%]
47 \@@cref@getcounter{\@cref@getcounter[] [] []}}
48 \def\@@cref@getcounter[#1][#2][#3]#4\@nil#5{\def#5{#2}}
49 \def\@cref@getprefix#1#2{%
50 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
51 \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
52 \expandafter\@cref@getprefix\@tempa\@nil#2}
53 \def\@cref@getprefix{\@ifnextchar[%]
54 \@@cref@getprefix{\@cref@getprefix[] [] []}}
55 \def\@@cref@getprefix[#1][#2][#3]#4\@nil#5{\def#5{#3}}

```

We treat multiple references, supplied as a comma-separated list to `\cref` or `\Cref`, as a stack structure. So we define some utility macros for manipulating stacks (`\@nil` is used as an end-of-stack delimiter).

```

56 \def\stack@init#1{\def#1{\@nil}}
57 \def\stack@top#1{\expandafter\stack@top@aux#1}
58 \def\stack@top@aux#1,#2\@nil{#1}
59 \def\stack@pop#1{\expandafter\stack@pop@aux#1#1}
60 \def\stack@pop@aux#1,#2\@nil#3{\def#3{#2}\@nil}}
61 \def\stack@push#1#2{\expandafter\stack@push@aux\expandafter{#2}{#1}{#2}}
62 \def\stack@push@aux#1#2#3{\def#3{#2,#1}}
63 \def\stack@pull#1#2{\expandafter\stack@pull@aux#2{#1}{#2}}
64 \def\stack@pull@aux#1\@nil#2#3{\def#3{#1#2,\@nil}}
65 \newif\ifstackempty
66 \newif\ifstackfull
67 \def\isstackempty#1{%
68 \expandafter\ifx#1\@nil\stackemptytrue%
69 \else\stackemptyfalse\fi}
70 \def\isstackfull#1{%
71 \expandafter\ifx#1\@nil\stackfullfalse%
72 \else\stackfulltrue\fi}

```

We need to be able to determine which counter is used to reset a given counter. Usually, resets are done by sectioning counters, and we assume that to be the case here. `\isinresetlist` searches through one counter's reset list, stored in `\cl@<counter>`, to determine whether another counter appears there, and sets the new conditional appropriately. `\reset@by` searches through all the sectioning counters' reset lists, from lowest-level (subsubsection) to highest (part), checking whether the given counter is in the list, and returns the first sectioning counter whose list it appears in.

```

73 \newif\ifinresetlist
74 \def\isinresetlist#1#2{%
75 \def\@counter{#1}%
76 \begingroup%
77 \def\@elt##1{##1,}%
78 \expandafter\ifx\csname cl@#2\endcsname\relax%
79 \gdef\@resetstack{\@nil}%
80 \else%
81 \xdef\@resetstack{\csname cl@#2\endcsname\noexpand\@nil}%
82 \fi%
83 \endgroup%
84 \isstackfull{\@resetstack}%
85 \@whilesw\ifstackfull\fi{%

```

```

86 \edef\@nextcounter{\stack@top{\@resetstack}}%
87 \ifx\@nextcounter\@counter%
88 \stackfullfalse%
89 \else%
90 \let\@nextcounter\relax%
91 \stack@pop{\@resetstack}%
92 \isstackfull{\@resetstack}%
93 \fi}%
94 \ifx\@nextcounter\relax%
95 \inresetlistfalse%
96 \else%
97 \inresetlisttrue%
98 \fi}
99 %
100 \def\reset@by#1#2{%
101 \isinresetlist{#1}{subsubsubsection}%
102 \ifinresetlist%
103 \def#2{subsubsubsection}%
104 \else%
105 \isinresetlist{#1}{subsection}%
106 \ifinresetlist%
107 \def#2{subsection}%
108 \else%
109 \isinresetlist{#1}{section}%
110 \ifinresetlist%
111 \def#2{section}%
112 \else%
113 \isinresetlist{#1}{chapter}%
114 \ifinresetlist%
115 \def#2{chapter}%
116 \else%
117 \isinresetlist{#1}{part}%
118 \ifinresetlist%
119 \def#2{part}%
120 \else%
121 \let#2\relax%
122 \fi%
123 \fi%
124 \fi%
125 \fi%
126 \fi%
127 \fi%
128 \fi%
129 \fi%
130 \fi%
131 \fi}

```

6.3 Referencing Commands

Define the main referencing macros `\cref` and the start-of-sentence variant `\Cref`.

```

132 \newcommand{\cref}[1]{\@cref{cref}{#1}}
133 \newcommand{\Cref}[1]{\@cref{Cref}{#1}}
134 \newcommand{\crefrange}[2]{\@setcrefrange{#1}{#2}{cref}{}}
135 \newcommand{\Crefrange}[2]{\@setcrefrange{#1}{#2}{Cref}{}}

```

To save duplicating code, the referencing macros pass an argument determining the variant to an auxilliary macro `\@cref`, which does the real work. The `\@cref` macro is the behemoth at the heart of all the smart referencing features. It deals with grouping references by type, typesetting the conjunctions between groups, choosing the right formatting macro to use for each reference, and collapsing consecutive references into ranges.

```
136 \def\@cref#1#2{%
137   \begingroup%
```

Initialise some things, and put all the references into a stack called `\@refstack`.

```
138   \countdef\count@consecutive=0%
139   \newif\iffirstgroup%
140   \stack@init{\@refstack}%
141   \stack@push{#2}{\@refstack}%
142   \firstgrouptrue%
143   \isstackfull{\@refstack}%
```

Loop until the reference stack is empty.

```
144   \@whilesw\ifstackfull\fi{%
145     \stack@init{\@refsubstack}%
146     \edef\@nextref{\stack@top{\@refstack}}%
147     \expandafter\ifx\csname r@\@nextref\endcsname\relax%
148       \def\@currenttype{\@undefined}%
149     \else%
150       \expandafter\cref@gettype\expandafter{\@nextref}{\@currenttype}%
151     \fi%
152     \let\@nexttype\@currenttype%
```

Move references from `\@refstack` into a different stack called `\@refsubstack`, until we encounter a reference that has a different type to those that came before.

```
153     \@whilesw\ifx\@nexttype\@currenttype\fi{%
154       \expandafter\stack@pull\expandafter{\@nextref}{\@refsubstack}%
155       \stack@pop{\@refstack}%
156       \isstackempty{\@refstack}%
157       \ifstackempty%
158         \def\@nexttype{\relax}%
159       \else%
160         \edef\@nextref{\stack@top{\@refstack}}%
161         \expandafter\ifx\csname r@\@nextref\endcsname\relax%
162           \def\@currenttype{\@undefined}%
163         \else%
164           \expandafter\cref@gettype\expandafter{\@nextref}{\@currenttype}%
165         \fi%
166       \fi}%
```

Typeset appropriate conjunction between groups of reference types.

```
167     \iffirstgroup%
168     \else%
169       \isstackfull{\@refstack}%
170       \ifstackfull%
171         \crefmiddleconjunction%
172       \else%
173         \creflastconjunction%
174       \fi%
175     \fi%
```

```

176 \firstgroupfalse%
    Process first group of consecutive references from substack.
177 \edef\@nextref{\stack@top{\@refsubstack}}%
178 \stack@pop{\@refsubstack}%
    If the substack only contains one reference, typeset it,
179 \isstackempty{\@refsubstack}%
180 \ifstackempty%
181 \expandafter\@setcref\expandafter{\@nextref}{#1}{}%
    otherwise, find end of consecutive references.
182 \else%
183 \edef\@beginref{\@nextref}%
184 \let\@endref\relax%
185 \edef\@nextref{\stack@top{\@refsubstack}}%
186 \count@consecutive=1%
187 \expandafter\ifx\csname r@\@beginref\endcsname\relax%
188 \refconsecutivefalse%
189 \else%
190 \expandafter\ifx\csname r@\@nextref\endcsname\relax%
191 \refconsecutivefalse%
192 \else%
193 \edef\@tmpa{\@beginref}{\@nextref}%
194 \expandafter\isrefconsecutive\@tmpa%
195 \fi%
196 \fi%
197 \@whilesw\ifrefconsecutive\fi%
198 \advance\count@consecutive 1%
199 \let\@endref\@nextref%
200 \stack@pop{\@refsubstack}%
201 \isstackempty{\@refsubstack}%
202 \ifstackempty%
203 \refconsecutivefalse%
204 \else%
205 \edef\@nextref{\stack@top{\@refsubstack}}%
206 \expandafter\ifx\csname r@\@nextref\endcsname\relax%
207 \refconsecutivefalse%
208 \else%
209 \edef\@tmpa{\@endref}{\@nextref}%
210 \expandafter\isrefconsecutive\@tmpa%
211 \fi%
212 \fi}%
    If there were no consecutive references, typeset the first reference;
213 \ifx\@endref\relax%
214 \expandafter\@setcref\expandafter{\@beginref}{#1}{@first}%
215 \else%
    if there were only two consecutive references, typeset the first one and return the
    second to the substack,
216 \ifnum\count@consecutive=2%
217 \expandafter\@setcref\expandafter{\@beginref}{#1}{@first}%
218 \expandafter\stack@push\expandafter{\@endref}{\@refsubstack}%
    otherwise, typeset a reference range.

```

```

219         \else
220         \edef\tmpa{\@beginref}{\@endref}}%
221         \ifstackempty%
222         \expandafter\@setcrefrange\tmpa{#1}{}%
223         \else%
224         \expandafter\@setcrefrange\tmpa{#1}{@first}%
225         \fi%
226     \fi%
227 \fi%

    Loop until substack is empty.
228     \isstackfull{\@refsubstack}%
229     \@whilesw\ifstackfull\fi{%
230         \edef\@beginref{\stack@top{\@refsubstack}}%
231         \stack@pop{\@refsubstack}%
232         \let\@endref\relax%

        If substack only contains only one reference, typeset it,
233         \isstackempty{\@refsubstack}%
234         \ifstackempty%
235         \expandafter\@setcref\expandafter{\@beginref}{#1}{@last}%
        otherwise, find end of consecutive references.
236     \else%
237         \edef\@nextref{\stack@top{\@refsubstack}}%
238         \count@consecutive=1%
239         \expandafter\ifx\csname r@\@nextref\endcsname\relax%
240         \refconsecutivefalse%
241     \else%
242         \edef\tmpa{\@beginref}{\@nextref}}%
243         \expandafter\isrefconsecutive\tmpa%
244     \fi%
245     \@whilesw\ifrefconsecutive\fi{%
246         \advance\count@consecutive 1%
247         \let\@endref\@nextref%
248         \stack@pop{\@refsubstack}%
249         \isstackempty{\@refsubstack}%
250         \ifstackempty%
251         \refconsecutivefalse%
252     \else%
253         \edef\@nextref{\stack@top{\@refsubstack}}%
254         \expandafter\ifx\csname r@\@nextref\endcsname\relax%
255         \refconsecutivefalse%
256     \else%
257         \edef\tmpa{\@endref}{\@nextref}}%
258         \expandafter\isrefconsecutive\tmpa%
259     \fi%
260 \fi%

    If the substack is now empty, we will need to typeset an “end” reference, otherwise
    we will need to typeset a “middle” reference.
261     \isstackempty{\@refsubstack}%
262     \ifstackempty%
263     \def\@pos{@last}%
264 \else%
265     \def\@pos{@middle}%

```

```

266         \fi%
    If there were no consecutive references, just typeset the next reference;
267         \ifx\@endref\relax%
268         \edef\@tmpa{\@beginref}{#1}{\@pos}}%
269         \expandafter\@setcref\@tmpa%
270         \else%
    if there were only two consecutive references, typeset the first one, and return the
    second one to the substack,
271         \ifnum\count@consecutive=2%
272         \edef\@tmpa{\@beginref}{#1}{\@pos}}%
273         \expandafter\@setcref\@tmpa%
    otherwise, typeset a reference range.
274         \else
275         \edef\@tmpa{\@beginref}{\@endref}{#1}{\@pos}}%
276         \expandafter\@setcrefrange\@tmpa%
277         \fi%
278     \fi%
279     \fi%
280     \isstackfull{\@refsubstack}%
281 }%
282 \fi%
283 \isstackfull{\@refstack}%
284 }%
285 \endgroup}

```

The internal `\@setcref` macro deals with actually typesetting the reference, by calling the appropriate type-dependent formatting macro defined by `\crefformat` etc.

```

286 \def\@setcref#1#2#3{%
287   \expandafter\ifx\csname r@#1\endcsname\relax%
288     \protect\G@refundefinedtrue%
289     \nfss@text{\reset@font\bfseries ??}%
290     \@latex@warning{Reference ‘#1’ on page \thepage \space undefined}%
291   \else%
292     \cref@gettype{#1}{\@temptype}% puts label type in \@temptype
293     \cref@getlabel{#1}{\@templabel}% puts label in \@templabel
294     \expandafter\ifx\csname #2@\@temptype @format#3\endcsname\relax%
295       \protect\G@refundefinedtrue%
296       \nfss@text{\reset@font\bfseries ??}~\@templabel%
297       \@latex@warning{\string\Cref \space reference format for label
298         type ‘\@temptype’ undefined}%
299     \else%
300       \expandafter\@setcref\expandafter%
301         {\csname #2@\@temptype @format#3\endcsname}{#1}%
302     \fi%
303   \fi}

```

We separate out the very final typesetting step into a separate macro, in order to make it easier to redefine things later to make them work with the `hyperref` package.

```

304 \def\@@setcref#1#2{\cref@getlabel{#2}{\@templabel}#1{\@templabel}{}}

```

Define a new conditional to test whether two references are consecutive (needed when typesetting reference ranges). This uses the counter and prefix (i.e. formatted version of the counter that resets the label's counter) information provided by `\r@{label}` (via the aux file) to check if the prefixes are identical (i.e. the references come from the same chapter, section or whatever), and that the label counters differ by 1.

```

305 \newif\ifrefconsecutive%
306 \def\isrefconsecutive#1#2{%
307   \begingroup%
308     \countdef\refa@counter=1%
309     \countdef\refb@counter=2%
310     \cref@getcounter{#1}{\@result}%
311     \refa@counter=\@result%
312     \advance\refa@counter 1%
313     \cref@getcounter{#2}{\@result}%
314     \refb@counter=\@result%
315     \cref@getprefix{#1}{\refa@prefix}%
316     \cref@getprefix{#2}{\refb@prefix}%
317     \def\@after{\refconsecutivefalse}%
318     \ifx\refa@prefix\refb@prefix%
319       \ifnum\refa@counter=\refb@counter%
320         \def\@after{\refconsecutivetrue}%
321       \fi%
322     \fi%
323   \expandafter\endgroup\@after}

```

The internal `\@setcrefrange` macro deals with typesetting reference ranges, just as `\setcref` does for normal references. The actual typesetting is no more complicated in the range case; it's the error checking that makes the code so much longer. We now have to check whether *two* references are undefined, whether *two* reference formats are undefined, whether the reference types are consistent, and also combinations of these various errors.

```

324 \def\@setcrefrange#1#2#3#4{%
325   \expandafter\ifx\csname r@#1\endcsname\relax%
326     \protect\G@refundefinedtrue%
327     \@latex@warning{Reference ‘#1’ on page \thepage \space undefined}%
328     \expandafter\ifx\csname r@#2\endcsname\relax%
329       \nfss@text{\reset@font\bfseries ??}--%
330       \nfss@text{\reset@font\bfseries ??}%
331       \@latex@warning{Reference ‘#2’ on page \thepage \space undefined}%
332     \else%
333       \cref@getlabel{#2}{\@labelb}%
334       \nfss@text{\reset@font\bfseries ??}--\@labelb%
335     \fi%
336   \else%
337     \expandafter\ifx\csname r@#2\endcsname\relax%
338       \protect\G@refundefinedtrue%
339       \cref@getlabel{#1}{\@labela}%
340       \@labela--\nfss@text{\reset@font\bfseries ??}%
341       \@latex@warning{Reference ‘#2’ on page \thepage \space undefined}%

```

If both references are defined, check that the reference format is defined.


```

342 \else%
343 \cref@gettype{#1}{\@typea}%
344 \cref@gettype{#2}{\@typeb}%
345 \cref@getlabel{#1}{\@labela}%
346 \cref@getlabel{#2}{\@labelb}%
347 \edef\@formata{\expandafter\noexpand%
348 \csname #3range@\@typea @format#4\endcsname}%
349 \edef\@formatb{\expandafter\noexpand%
350 \csname #3range@\@typeb @format#4\endcsname}%
351 \expandafter\ifx\@formata\relax%
352 \protect\G@refundefinedtrue%
353 \nfss@text{\reset@font\bfseries ??}~\@labela--\@labelb%
354 \@latex@warning{#3 \space reference format for label
355 type '\@typea' undefined}%
356 \else%

```

If reference types are identical, typeset reference range, otherwise display warning. (Note: there's no need to check if reference format for second type is defined, since if it isn't it will be caught here as a non-identical type.)

```

357 \ifx\formata\formatb%
358 \@@setcrefrange{\@formata}{#1}{#2}%
359 \else%
360 \protect\G@refundefinedtrue%
361 \nfss@text{\reset@font\bfseries ??}~\@labela--\@labelb%
362 \@latex@warning{Types inconsistent in reference range for
363 references '#1' and '#2' on page \thepage}%
364 \fi%
365 \fi%
366 \fi%
367 \fi}

```

We again separate out the very final typesetting step into a separate macro, in order to make it easier to redefine things later to make them work with the `hyperref` package.

```

368 \def\@@setcrefrange#1#2#3{%
369 \cref@getlabel{#2}{\@labela}%
370 \cref@getlabel{#3}{\@labelb}%
371 #1{\@labela}{\@labelb}{-}{-}{-}}

```

6.4 Reference Format Customisation Commands

`\crefformat` et al. are user-level commands used to define the format of different reference types. They simply use the supplied argument(s) to define appropriately named formatting macro(s), which are called by `\setcref`. The only moderately interesting part is that if the corresponding `\Crefformat` or `\crefformat` variant is not already defined, they define it to be a version with the first letter capitalised or lower-cased.

```

372 \newcommand{\crefformat}[2]{\@crefformat{cref}{#1}{#2}}
373 \newcommand{\Crefformat}[2]{\@crefformat{Cref}{#1}{#2}}
374 \newcommand{\crefrangeformat}[2]{\@crefrangeformat{crefrange}{#1}{#2}}
375 \newcommand{\Crefrangeformat}[2]{\@crefrangeformat{Crefrange}{#1}{#2}}
376 \newcommand{\crefmultiformat}[4]{%
377 \@crefmultiformat{cref}{#1}{#2}{#3}{#4}}
378 \newcommand{\Crefmultiformat}[4]{%

```

```

379 \@crefmultiformat{Cref}{#1}{#2}{#3}{#4}}
380 \newcommand{\crefrangemultiformat}[4]{%
381 \@crefrangemultiformat{crefrange}{#1}{#2}{#3}{#4}}
382 \newcommand{\Crefrangemultiformat}[4]{%
383 \@crefrangemultiformat{Crefrange}{#1}{#2}{#3}{#4}}

```

The utility macros do the real work, by using the first argument (“cref”, “Cref”, “crefrange” or “Crefrange”) to determine how to define the corresponding command with the other capitalisation.

```

384 \def\@crefformat#1#2#3{%
385 \expandafter\def\csname #1@#2@format\endcsname##1##2##3{#3}%

```

Note that the \tmpa macro makes use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

386 \def\@tmpa##1##2\@nil{%
387 \if##1c \def\@other{C##2}%
388 \else\def\@other{c##2}\fi}%
389 \@tmpa#1\@nil%

```

Note that the \tmpa macro makes use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

390 \def\@tmpa##1##2\@nil{%
391 \if##1c\def\@changepcase{\MakeUppercase}%
392 \else\def\@changepcase{\MakeLowercase}\fi}%
393 \@tmpa#1\@nil%
394 \ifundefined{\@other @#2@format}{%
395 \expandafter\edef\csname\@other @#2@format\endcsname##1##2##3{%
396 \expandafter\noexpand\@changepcase\csname #1@#2@format\endcsname%
397 {##1}{##2}{##3}}}%
398 %

```

```

399 \def\@crefrangeformat#1#2#3{%
400 \expandafter\def\csname #1@#2@format\endcsname##1##2##3##4##5##6{#3}%

```

Note that the \tmpa macro makes use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

401 \def\@tmpa##1##2\@nil{%
402 \if##1c \def\@other{C##2}%
403 \else\def\@other{c##2}\fi}%
404 \@tmpa#1\@nil%

```

Note that the \tmpa macro makes use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

405 \def\@tmpa##1##2\@nil{%
406 \if##1c\def\@changepcase{\MakeUppercase}%
407 \else\def\@changepcase{\MakeLowercase}\fi}%
408 \@tmpa#1\@nil%
409 \ifundefined{\@other @#2@format}{%
410 \expandafter\edef%
411 \csname\@other @#2@format\endcsname##1##2##3##4##5##6{%
412 \expandafter\noexpand\@changepcase\csname #1@#2@format\endcsname%
413 {##1}{##2}{##3}{##4}{##5}{##6}}}%
414 %
415 \def\@crefmultiformat#1#2#3#4#5{%
416 \expandafter\def\csname #1@#2@format@first\endcsname##1##2##3{#3}%
417 \expandafter\def\csname #1@#2@format@middle\endcsname##1##2##3{#4}%
418 \expandafter\def\csname #1@#2@format@last\endcsname##1##2##3{#5}%

```

Note that the \tmpa macros make use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

419 \def\tmpa##1##2\@nil{%
420   \if##1c \def\@other{C##2}%
421   \else\def\@other{c##2}\fi}%
422 \@tmpa#1\@nil%
423 \def\tmpa##1##2\@nil{%
424   \if##1c\def\@changepcase{\MakeUppercase}%
425   \else\def\@changepcase{\MakeLowercase}\fi}%
426 \@tmpa#1\@nil%
427 \@ifundefined{\@other @#2@format@first}{%
428   \expandafter\edef%
429     \csname\@other @#2@format@first\endcsname##1##2##3{%
430     \expandafter\noexpand\@changepcase%
431     \csname #1@#2@format@first\endcsname%
432     {##1}{##2}{##3}}}%
433 \@ifundefined{\@other @#2@format@middle}{%
434   \expandafter\edef%
435     \csname\@other @#2@format@middle\endcsname##1##2##3{%
436     \csname #1@#2@format@middle\endcsname{##1}{##2}{##3}}}%
437 \@ifundefined{\@other @#2@format@last}{%
438   \expandafter\edef%
439     \csname\@other @#2@format@last\endcsname##1##2##3{%
440     \csname #1@#2@format@last\endcsname{##1}{##2}{##3}}}%
441 %
442 \def\@crefrangemultiformat#1#2#3#4#5{%
443   \expandafter\def\csname #1@#2@format@first\endcsname%
444     ##1##2##3##4##5##6{#3}%
445   \expandafter\def\csname #1@#2@format@middle\endcsname%
446     ##1##2##3##4##5##6{#4}%
447   \expandafter\def\csname #1@#2@format@last\endcsname%
448     ##1##2##3##4##5##6{#5}%

```

Note that the \tmpa macros make use of the fact that the first character of #1 is “c” for lower-case, “C” for upper-case.

```

449 \def\tmpa##1##2\@nil{%
450   \if##1c \def\@other{C##2}%
451   \else\def\@other{c##2}\fi}%
452 \@tmpa#1\@nil%
453 \def\tmpa##1##2\@nil{%
454   \if##1c\def\@changepcase{\MakeUppercase}%
455   \else\def\@changepcase{\MakeLowercase}\fi}%
456 \@tmpa#1\@nil%
457 \@ifundefined{\@other @#2@format@first}{%
458   \expandafter\edef%
459     \csname\@other @#2@format@first\endcsname##1##2##3##4##5##6{%
460     \expandafter\noexpand\@changepcase%
461     \csname #1@#2@format@first\endcsname%
462     {##1}{##2}{##3}{##4}{##5}{##6}}}%
463 \@ifundefined{\@other @#2@format@middle}{%
464   \expandafter\edef%
465     \csname\@other @#2@format@middle\endcsname##1##2##3##4##5##6{%
466     \csname #1@#2@format@middle\endcsname%
467     {##1}{##2}{##3}{##4}{##5}{##6}}}%

```

```

468 \@ifundefined{\@other @#2@format@last}{%
469 \expandafter\edef%
470 \csname\@other @#2@format@last\endcsname##1##2##3##4##5##6{%
471 \csname #1@#2@format@last\endcsname%
472 {##1}{##2}{##3}{##4}{##5}{##6}}{}}

```

6.5 Default Reference Formats

Define the default reference formats, appropriate for L^AT_EX documents written in English.

To-Do: add babel support

```

473 \crefformat{equation}{eq.~#2(#1)#3}
474 \crefformat{chapter}{chapter~#2#1#3}
475 \crefformat{section}{section~#2#1#3}
476 \crefformat{subsection}{section~#2#1#3}
477 \crefformat{subsubsection}{section~#2#1#3}
478 \crefformat{subsubsubsection}{section~#2#1#3}
479 \crefformat{figure}{fig.~#2#1#3}
480 \crefformat{theorem}{theorem~#2#1#3}
481 \crefmultiformat{equation}%
482 {eqs.~#2(#1)#3}{, #2(#1)#3}{ and~#2(#1)#3}
483 \crefmultiformat{chapter}%
484 {chapters~#2#1#3}{, #2#1#3}{ and~#2#1#3}
485 \crefmultiformat{section}%
486 {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
487 \crefmultiformat{subsection}%
488 {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
489 \crefmultiformat{subsubsection}%
490 {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
491 \crefmultiformat{subsubsubsection}%
492 {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
493 \crefmultiformat{figure}%
494 {figs.~#2#1#3}{, #2#1#3}{ and~#2#1#3}
495 \crefmultiformat{theorem}%
496 {theorems~#2#1#3}{, #2#1#3}{ and~#2#1#3}
497 %
498 \crefrangeformat{equation}{eqs.~#3(#1)#4--#5(#2)#6}
499 \crefrangeformat{chapter}{chapters~#3#1#4--#5#2#6}
500 \crefrangeformat{section}{sections~#3#1#4--#5#2#6}
501 \crefrangeformat{subsection}{sections~#3#1#4--#5#2#6}
502 \crefrangeformat{subsubsection}{sections~#3#1#4--#5#2#6}
503 \crefrangeformat{subsubsubsection}{sections~#3#1#4--#5#2#6}
504 \crefrangeformat{figure}{figs.~#3#1#4--#5#2#6}
505 \crefrangeformat{theorem}{theorems~#3#1#4--#5#2#6}
506 \crefrangemultiformat{equation}%
507 {eqs.~#3(#1)#4--#5(#2)#6}%
508 {, #3(#1)#4--#5(#2)#6}%
509 { and~#3(#1)#4--#5(#2)#6}
510 \crefrangemultiformat{chapter}%
511 {chapters~#3#1#4--#5#2#6}%
512 {, #3#1#4--#5#2#6}%
513 { and~#3#1#4--#5#2#6}
514 \crefrangemultiformat{section}%

```

```

515 {sections~#3#1#4--#5#2#6}%
516 {, #3#1#4--#5#2#6}%
517 { and~#3#1#4--#5#2#6}
518 \crefrangemultiformat{subsection}%
519 {sections~#3#1#4--#5#2#6}%
520 {, #3#1#4--#5#2#6}%
521 { and~#3#1#4--#5#2#6}
522 \crefrangemultiformat{subsubsection}%
523 {sections~#3#1#4--#5#2#6}%
524 {, #3#1#4--#5#2#6}%
525 { and~#3#1#4--#5#2#6}
526 \crefrangemultiformat{subsubsubsection}%
527 {sections~#3#1#4--#5#2#6}%
528 {, #3#1#4--#5#2#6}%
529 { and~#3#1#4--#5#2#6}
530 \crefrangemultiformat{figure}%
531 {figs~#3#1#4--#5#2#6}%
532 {, #3#1#4--#5#2#6}%
533 { and~#3#1#4--#5#2#6}
534 \crefrangemultiformat{theorem}%
535 {theorems~#3#1#4--#5#2#6}%
536 {, #3#1#4--#5#2#6}%
537 { and~#3#1#4--#5#2#6}
538 %
539 \def\crefmiddleconjunction{, }
540 \def\creflastconjunction{, and }

```

6.6 hyperref Support

```

541 \DeclareOption{hyperref}{%
542 \PackageInfo{cleveref}{option ‘hyperref’ loaded}

We redefine the utility macros to cope with the extra arguments supplied by
hyperref (via the aux file).
543 \def\cref@label#1#2#3#4#5{\@result}
544 \def\cref@hyperref#1{\expandafter\expandafter\expandafter%
545 \@fourthoffive\csname r@#1\endcsname}
546 \def\cref@getlabel#1#2{%
547 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
548 \edef\@tempa{\expandafter\@firstoffive\@tempa}%
549 \expandafter\@cref@getlabel\@tempa\@nil#2}
550 \def\cref@gettype#1#2{%
551 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
552 \edef\@tempa{\expandafter\@firstoffive\@tempa}%
553 \expandafter\@cref@gettype\@tempa\@nil#2}
554 \def\cref@getcounter#1#2{%
555 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
556 \edef\@tempa{\expandafter\@firstoffive\@tempa}%
557 \expandafter\@cref@getcounter\@tempa\@nil#2}
558 \def\cref@getprefix#1#2{%
559 \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
560 \edef\@tempa{\expandafter\@firstoffive\@tempa}%
561 \expandafter\@cref@getprefix\@tempa\@nil#2}

```

The `hyperref` package stores the original `\refstepcounter` definition as `\H@refstepcounter`, which we therefore need to modify so that it adds the extra information to `\@currentlabel`.

```

562 \def\H@refstepcounter#1{%
563   \stepcounter{#1}%
564   \reset@by{#1}{\@result}%
565   \ifx\@result\relax\def\@result{}%
566   \else\edef\@result{\csname the\@result\endcsname}\fi%
567   \protected@edef\@currentlabel%
568     {[#1][\arabic{#1}][\@result]%
569     \csname p@#1\endcsname\csname the#1\endcsname}}

```

The original `\refstepcounter`, as stored earlier in `\old@refstepcounter` already calls `\refstepcounter` if `hyperref` is loaded, and we just redefined it to store the type information. so we only need to change `\@currentlabel` in our `\refstepcounter` if an optional argument was supplied.

```

570 \def\refstepcounter@noarg#1{\old@refstepcounter{#1}}
571 \def\refstepcounter@optarg[#1]#2{%
572   \old@refstepcounter{#2}%
573   \expandafter\@cref@getlabel\@currentlabel\@nil{\@templabel}%
574   \reset@by{#2}{\@tempreset}%
575   \ifx\@tempreset\relax\def\@tempreset{}%
576   \else\edef\@tempreset{\csname the\@tempreset\endcsname}\fi%
577   \protected@edef\@currentlabel{%
578     [#1][\arabic{#2}][\@tempreset]\@templabel}}

```

Redefine `@setcref` and `@setrangeref` to create hyperlinks, using the extra arguments supplied in `\@<label>` (via the aux file) by `hyperref`.

```

579 \def\@setcref#1#2{%
580   \cref@getlabel{#2}{\@templabel}%
581   \edef\@templink{\cref@hyperref{#2}}%
582   #1{\@templabel}{\hyper@linkstart{link}{\@templink}}{\hyper@linkend}}
583 \def\@setcrefrange#1#2#3{%
584   \cref@getlabel{#2}{\@labela}%
585   \cref@getlabel{#3}{\@labelb}%
586   \edef\@linka{\cref@hyperref{#2}}%
587   \edef\@linkb{\cref@hyperref{#3}}%
588   #1{\@labela}{\@labelb}{\hyper@linkstart{link}{\@linka}}{\hyper@linkend}}%
589   {\hyper@linkstart{link}{\@linkb}}{\hyper@linkend}}
590 } % end of hyperref option

```

6.7 ntheorem Support

```

591 \DeclareOption{ntheorem}{%
592 \PackageInfo{cleveref}{option 'ntheorem' loaded}

```

We modify `ntheorem`'s version of the `\thm` macro very slightly, to have it call `\refstepcounter` with an optional argument containing the theorem type.

```

593 \gdef\@thm#1#2#3{%
594   \if@thmmarks
595     \stepcounter{end\InTheoType ctr}%
596   \fi
597   \renewcommand{\InTheoType}{#1}%
598   \if@thmmarks
599     \stepcounter{curr#1ctr}%
600     \setcounter{end#1ctr}{0}%

```

```

601 \fi
602 \refstepcounter[#1]{#2}% <<<<
603 \thm@topsepadd \theorempostskipamount
604 \ifvmode \advance\thm@topsepadd\partopsep\fi
605 \trivlist
606 \@topsep \theorempreskipamount
607 \@topsepadd \thm@topsepadd
608 \advance\linewidth -\theorem@indent
609 \advance\@totalleftmargin \theorem@indent
610 \parshape \@ne \@totalleftmargin \linewidth
611 \@ifnextchar[{\@ythm{#1}{#2}{#3}}{\@xthm{#1}{#2}{#3}}%]
612 }

```

Default formats for new theorem-like environments defined by ntheorem.

```

613 \crefformat{lemma}{lemma~#2#1#3}
614 \crefformat{corollary}{corollary~#2#1#3}
615 \crefformat{proposition}{proposition~#2#1#3}
616 \crefformat{definition}{definition~#2#1#3}
617 \crefformat{result}{result~#2#1#3}
618 \crefrangeformat{lemma}{lemma~#3#1#4--#5#2#6}
619 \crefrangeformat{corollary}{corollary~#3#1#4--#5#2#6}
620 \crefrangeformat{proposition}{proposition~#3#1#4--#5#2#6}
621 \crefrangeformat{definition}{definition~#3#1#4--#5#2#6}
622 \crefrangeformat{result}{result~#3#1#4--#5#2#6}
623 \crefmultiformat{lemma}%
624 {lemmas~#2#1#3}{, #2#1#3}{ and~#2#1#3}
625 \crefmultiformat{corollary}%
626 {corollaries~#2#1#3}{, #2#1#3}{ and~#2#1#3}
627 \crefmultiformat{proposition}%
628 {propositions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
629 \crefmultiformat{definition}%
630 {definitions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
631 \crefmultiformat{result}%
632 {results~#2#1#3}{, #2#1#3}{ and~#2#1#3}
633 \crefrangemultiformat{lemma}%
634 {lemmas~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
635 { and~#3#1#4--#5#2#6}
636 \crefrangemultiformat{corollary}%
637 {corollaries~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
638 { and~#3#1#4--#5#2#6}
639 \crefrangemultiformat{proposition}%
640 {propositions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
641 { and~#3#1#4--#5#2#6}
642 \crefrangemultiformat{definition}%
643 {definitions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
644 { and~#3#1#4--#5#2#6}
645 \crefrangemultiformat{result}%
646 {results~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
647 { and~#3#1#4--#5#2#6}
648 } % end of ntheorem option

```

Process options.

```

649 \ProcessOptions\relax

```