The cleveref package*

Toby Cubitt
toby-cleveref@dr-qubit.org

10/07/2007

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. It's 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 0.8, dated 10/07/2007.

LaTeX 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`, `hyperref`'s `\autoref` command, and (for theorem-like environments) `ntheorem` (with the `thref` option). (There are many others, but these come closest to providing similar features to `cleveref`.) 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 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. (Not to mention automatic collapsing of consecutive references, `ntheorem` support, or any of the other additional `cleveref` features.)

The `hyperref` package's `\autoref` command typesets a (customisable) name before a reference, based on the reference type. This is less flexible than `cleveref`'s fully customisable reference formatting, but when combined with `varioref`, the two packages working together come close. However, even with `hyperref`, it is impossible to customise precisely which part of the reference is made into a hyperlink in PDF documents; this is very easy with `cleveref`. And it still remains impossible to typeset multiple references, have consecutive references collapsed, etc.

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` works equally well when referencing any type of object, as well as fully supporting `ntheorem`. `cleveref` also provides a number extra features too, such as multiple references, automatic collapsing of reference ranges, control over the placement of hyper-links, etc.

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 LaTeX'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 LaTeX 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` command[1], 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` `\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` `\Cref` To typeset multiple references, simply list the lables 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 to 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,

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

```
\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)

To prevent a sequence of consecutive references from being collapsed to a reference range, you can separate the references in the list by one or more empty references, at the point where you want to prevent collapsing. For example,

```
\cref{eq1,eq2,eq3,,eq4}
```

will be typeset as

Eqs. (1)–(3), and (4)

or

```
\cref{eq1,eq2,,eq3,eq4,eq5,,eq6,eq7,eq8}
```

will be typeset as

Eqs. (1), (2), (3)–(5), and (6)–(7)

You can safely put an empty references between references that would never be collapsed anyway; they will simply be ignored.

`\cref*` `\Cref*` `\crefrange*` `\Crefrange*`

When `cleveref` is used along with the `hyperref` package (see section 2.2, and section 3), additional starred variants of all the referencing commands are available. The standard referencing commands will make references into hyper-links; the starred variants prevent this, producing the same typeset text but without creating hyper-links.

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 \Crefformat

Reference formats for *single* references are defined or redefined using the `\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 whe `\Crefformat` is called, if 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. (⟨*counter*⟩)

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 \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 `\creformat` 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 ⟨*type*⟩ 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 ⟨*format*⟩ argument can again be any valid LaTeX code, with fragile commands `\protect`ed. 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. (⟨*counter1*⟩)–(⟨*counter2*⟩)

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

2.2.3 Multiple References

`\crefmultiformat` `\Crefmultiformat` `\crefrangemultiformat` `\Crefrangemultiformat`

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*⟩ LaTeX 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 `\crefformat`s) as

> Eqs. (1)–(3)`\crefmiddleconjunction` Theorems 1 and 2`\crefmiddleconjunction` Fig. 1`\creflastconjunction` Theorem 3

2.3 Poor Man's `cleveref`

Sometimes you may need to send your LaTeX source to someone who can't install the `cleveref` package themselves. For example, many academic journals accept papers in LaTeX format, but only support a subset of the packages available on CTAN. The `poorman` option was designed specifically to help in this situation.

When the `poorman` option is supplied, your document will be processed as normal. But in addition, a `sed` script will automatically be written, containing rules for replacing all the `\cref` commands with the LaTeX code that they would produce, and using the standard `\ref` command to produce the cross-reference numbers themselves. I.e. the script rewrites your document as you would have done if you had had to do it manually!

The advantage, of course, is that you *don't* have to do it manually. Instead, you can use all the features of `cleveref`, and once you've created a version of your document that you want to send elsewere, you can process it through the script to remove the `cleveref` dependency.

The script is written to the same directory as the LaTeX source file, and given the same name as the source file but with the extension `.sed`. To process your

document through the script, all you need to do is run the following from your shell:

sed -f ⟨*name*⟩`.sed` ⟨*name*⟩`.tex` ¿ ⟨*newname*⟩`.tex`

where ⟨*name*⟩ is the name of the file containing your LaTeX source file minus the `.tex` extension, and ⟨*newname*⟩ is whatever you want to call the new version. *Do not* make ⟨*newname*⟩ the same as ⟨*name*⟩. (It's in any case wise to keep the original LaTeX source file containing the `cleveref` commands, in case you need to produce an updated version of your document in the future. Think of the ⟨*newname*⟩`.tex` file as being rather like a `DVI` file: something you can always reproduce from the original source.)

3 Interaction with Other Packages

Since `cleveref` redefines many internal commands involved in LaTeX'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 LaTeX referencing system are unlikely to work properly with `cleveref`.

4 Known Bugs

- `cleveref` will not work properly with the standard LaTeX `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`, `multine` and `split`, which *do* work properly with `cleveref`. (See `http://www.tug.org/pracjourn/2006-4/madsen/`).

- `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 reference-range collapsing to be disabled entirely via a package option

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 cotains 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.

```
\let\old@refstepcounter\refstepcounter
\def\refstepcounter%
  {\@ifnextchar[{\refstepcounter@optarg}{\refstepcounter@noarg}}%]
\def\refstepcounter@noarg#1{%
  \old@refstepcounter{#1}%
  \reset@by{#1}{\@result}%
  \ifx\@result\relax\def\@result{}%
  \else\edef\@result{\csname the\@result\endcsname}\fi%
```

```
  \protected@edef\@currentlabel{%
    [#1][\arabic{#1}][\@result]\@currentlabel}}
\def\refstepcounter@optarg[#1]#2{%
  \old@refstepcounter{#2}%
  \reset@by{#2}{\@result}%
  \ifx\@result\relax\def\@result{}%
  \else\edef\@result{\csname the\@result\endcsname}\fi%
  \protected@edef\@currentlabel{%
    [#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. For some reason, `\DeclareRobustCommand` doesn't work here, so we make it robust manually.

```
\def\cref@label#1#2{\@result}
\AtBeginDocument{%
 \expandafter\def\csname ref \endcsname#1{%
   \expandafter\ifx\csname r@#1\endcsname\relax%
     \let\@result\relax%
   \else%
     \cref@getlabel{#1}{\@result}%
   \fi%
   \expandafter\@setref\csname r@#1\endcsname{\cref@label}{#1}}%
 \def\ref{\expandafter\protect\csname ref \endcsname}
}
```

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`.

```
\def\cref@getlabel#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
  \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
  \expandafter\@cref@getlabel\@tempa\@nil#2}
\def\@cref@getlabel{\@ifnextchar[%]
  \@@cref@getlabel{\@@cref@getlabel[][][]}}
\def\@@cref@getlabel[#1][#2][#3]#4\@nil#5{\def#5{#4}}
\def\cref@gettype#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
  \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
  \expandafter\@cref@gettype\@tempa\@nil#2}
```

```
\def\@cref@gettype{\@ifnextchar[%]
  \@@cref@gettype{\@@cref@gettype[][][]}}
\def\@@cref@gettype[#1][#2][#3]#4\@nil#5{\def#5{#1}}
\def\cref@getcounter#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
  \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
  \expandafter\@cref@getcounter\@tempa\@nil#2}
\def\@cref@getcounter{\@ifnextchar[%]
  \@@cref@getcounter{\@@cref@getcounter[][][]}}
\def\@@cref@getcounter[#1][#2][#3]#4\@nil#5{\def#5{#2}}
\def\cref@getprefix#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
  \edef\@tempa{\expandafter\@firstoftwo\@tempa}%
  \expandafter\@cref@getprefix\@tempa\@nil#2}
\def\@cref@getprefix{\@ifnextchar[%]
  \@@cref@getprefix{\@@cref@getprefix[][][]}}
\def\@@cref@getprefix[#1][#2][#3]#4\@nil#5{\def#5{#3}}
```

We treate 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 delimeter).

```
\def\stack@init#1{\def#1{\@nil}}
\def\stack@top#1{\expandafter\stack@top@aux#1}
\def\stack@top@aux#1,#2\@nil{#1}
\def\stack@pop#1{\expandafter\stack@pop@aux#1#1}
\def\stack@pop@aux#1,#2\@nil#3{\def#3{#2\@nil}}
\def\stack@push#1#2{\expandafter\stack@push@aux\expandafter{#2}{#1}{#2}}
\def\stack@push@aux#1#2#3{\def#3{#2,#1}}
\def\stack@pull#1#2{\expandafter\stack@pull@aux#2{#1}{#2}}
\def\stack@pull@aux#1\@nil#2#3{\def#3{#1#2,\@nil}}
\newif\ifstackempty
\newif\ifstackfull
\def\isstackempty#1{%
  \def\@tmpa{\@nil}%
  \ifx#1\@tmpa\stackemptytrue%
  \else\stackemptyfalse\fi
}
\def\isstackfull#1{%
  \def\@tmpa{\@nil}%
  \ifx#1\@tmpa\stackfullfalse%
  \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.

```
\newif\ifinresetlist
\def\isinresetlist#1#2{%
  \def\@counter{#1}%
  \begingroup%
```

```
    \def\@elt##1{##1,}%
    \expandafter\ifx\csname cl@#2\endcsname\relax%
      \gdef\@resetstack{,\@nil}%
    \else%
      \xdef\@resetstack{\csname cl@#2\endcsname\noexpand\@nil}%
    \fi%
  \endgroup%
  \isstackfull{\@resetstack}%
  \@whilesw\ifstackfull\fi{%
    \edef\@nextcounter{\stack@top{\@resetstack}}%
    \ifx\@nextcounter\@counter%
      \stackfullfalse%
    \else%
      \let\@nextcounter\relax%
      \stack@pop{\@resetstack}%
      \isstackfull{\@resetstack}%
    \fi}%
  \ifx\@nextcounter\relax%
    \inresetlistfalse%
  \else%
    \inresetlisttrue%
  \fi}
%
\def\reset@by#1#2{%
  \isinresetlist{#1}{subsubsubsection}%
  \ifinresetlist%
    \def#2{subsubsubsection}%
  \else%
    \isinresetlist{#1}{subsubsection}%
    \ifinresetlist%
      \def#2{subsubsection}%
    \else%
      \isinresetlist{#1}{subsection}%
      \ifinresetlist%
        \def#2{subsection}%
      \else%
        \isinresetlist{#1}{section}%
        \ifinresetlist%
          \def#2{section}%
        \else%
          \isinresetlist{#1}{chapter}%
          \ifinresetlist%
            \def#2{chapter}%
          \else%
            \isinresetlist{#1}{part}%
            \ifinresetlist%
              \def#2{part}%
            \else%
              \let#2\relax%
            \fi%
          \fi%
        \fi%
      \fi%
    \fi%
```

```
\fi}
```

6.3 Referencing Commands

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

```
\DeclareRobustCommand{\cref}[1]{\@cref{cref}{#1}}
\DeclareRobustCommand{\Cref}[1]{\@cref{Cref}{#1}}
\DeclareRobustCommand{\crefrange}[2]{\@setcrefrange{#1}{#2}{cref}{}}
\DeclareRobustCommand{\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.

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

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

```
  \countdef\count@consecutive=0%
  \def\@empty{}%
  \newif\if@firstgroup%
  \stack@init{\@refstack}%
  \stack@push{#2}{\@refstack}%
  \@firstgrouptrue%
  \isstackfull{\@refstack}%
```

Loop until the reference stack is empty.

```
  \@whilesw\ifstackfull\fi{%
    \stack@init{\@refsubstack}%
    \edef\@nextref{\stack@top{\@refstack}}%
    \expandafter\ifx\csname r@\@nextref\endcsname\relax%
      \def\@currenttype{\@undefined}%
    \else%
      \expandafter\cref@gettype\expandafter{\@nextref}{\@currenttype}%
    \fi%
    \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.

```
    \@whilesw\ifx\@nexttype\@currenttype\fi{%
      \expandafter\stack@pull\expandafter{\@nextref}{\@refsubstack}%
      \stack@pop{\@refstack}%
      \isstackempty{\@refstack}%
      \ifstackempty%
        \def\@nexttype{\relax}%
      \else%
        \edef\@nextref{\stack@top{\@refstack}}%
        \ifx\@nextref\@empty%
          \let\@currenttype\@nexttype%
        \else%
          \expandafter\ifx\csname r@\@nextref\endcsname\relax%
            \def\@currenttype{\@undefined}%
          \else%
```

```
            \expandafter\cref@gettype\expandafter%
              {\@nextref}{\@currenttype}%
          \fi%
        \fi%
      \fi}%
```

Typeset appropriate conjunction between groups of reference types.

```
    \if@firstgroup%
    \else%
      \isstackfull{\@refstack}%
      \ifstackfull%
        \@setcref@middleconjunction%
      \else%
        \@setcref@lastconjunction%
      \fi%
    \fi%
    \@firstgroupfalse%
```

Process first group of consecutive references from substack.

```
    \edef\@nextref{\stack@top{\@refsubstack}}%
    \stack@pop{\@refsubstack}%
```

If the substack only contains one reference, typeset it,

```
    \isstackempty{\@refsubstack}%
    \ifstackempty%
      \expandafter\@setcref\expandafter{\@nextref}{#1}{}%
```

otherwise, find end of consecutive references.

```
    \else%
      \edef\@beginref{\@nextref}%
      \let\@endref\relax%
      \edef\@nextref{\stack@top{\@refsubstack}}%
      \count@consecutive=1%
      \expandafter\ifx\csname r@\@beginref\endcsname\relax%
        \refconsecutivefalse%
      \else%
```

If next reference in substack is empty, it indicates that no further collapsing should take place. Having served its purpose, the empty reference and any consecutive empty references are removed from the substack.

```
        \ifx\@nextref\@empty%
          \refconsecutivefalse%
          \@whilesw\ifx\@nextref\@empty\fi{%
            \stack@pop{\@refsubstack}%
            \isstackempty{\@refsubstack}%
            \ifstackempty%
              \let\@nextref\relax%
            \else%
              \edef\@nextref{\stack@top{\@refsubstack}}%
            \fi%
          }%
          \ifnum\count@consecutive=2%
            \edef\@endref{\@endref,}%
          \fi%
```

Otherwise, test whether next reference is consecutive or not.

```
        \else
          \expandafter\ifx\csname r@\@nextref\endcsname\relax%
            \refconsecutivefalse%
          \else%
            \edef\@tmpa{{\@beginref}{\@nextref}}%
            \expandafter\isrefconsecutive\@tmpa%
          \fi%
        \fi%
      \fi%
      \@whilesw\ifrefconsecutive\fi{%
        \advance\count@consecutive 1%
        \let\@endref\@nextref%
        \stack@pop{\@refsubstack}%
        \isstackempty{\@refsubstack}%
        \ifstackempty%
          \refconsecutivefalse%
        \else%
          \edef\@nextref{\stack@top{\@refsubstack}}%
```

Test whether next reference is empty;

```
          \ifx\@nextref\@empty%
            \refconsecutivefalse%
            \@whilesw\ifx\@nextref\@empty\fi{%
              \stack@pop{\@refsubstack}%
              \isstackempty{\@refsubstack}%
              \ifstackempty%
                \let\@nextref\relax%
              \else%
                \edef\@nextref{\stack@top{\@refsubstack}}%
              \fi%
            }%
            \ifnum\count@consecutive=2%
              \edef\@endref{\@endref,}%
            \fi%
```

otherwise, test whether next reference is consecutive or not.

```
          \else
            \expandafter\ifx\csname r@\@nextref\endcsname\relax%
              \refconsecutivefalse%
            \else%
              \edef\@tmpa{{\@endref}{\@nextref}}%
              \expandafter\isrefconsecutive\@tmpa%
            \fi%
          \fi%
        \fi}%
```

If there were no consecutive references, typeset the first reference;

```
      \ifx\@endref\relax%
        \expandafter\@setcref\expandafter{\@beginref}{#1}{@first}%
```

if there were only two consecutive references, typeset the first one and return the second to the substack;

```
      \else%
        \ifnum\count@consecutive=2%
          \expandafter\@setcref\expandafter{\@beginref}{#1}{@first}%
          \expandafter\stack@push\expandafter{\@endref}{\@refsubstack}%
```

otherwise, typeset a reference range.

```
        \else
          \edef\@tmpa{{\@beginref}{\@endref}}%
          \ifstackempty%
            \expandafter\@setcrefrange\@tmpa{#1}{}%
          \else%
            \expandafter\@setcrefrange\@tmpa{#1}{@first}%
          \fi%
        \fi%
      \fi%
```

Process further groups of consecutive references, until substack is empty.

```
      \isstackfull{\@refsubstack}%
      \@whilesw\ifstackfull\fi{%
        \edef\@beginref{\stack@top{\@refsubstack}}%
        \stack@pop{\@refsubstack}%
        \let\@endref\relax%
```

If substack only contains only one reference, typeset it,

```
        \isstackempty{\@refsubstack}%
        \ifstackempty%
          \expandafter\@setcref\expandafter{\@beginref}{#1}{@last}%
```

otherwise, find end of consecutive references.

```
        \else%
          \edef\@nextref{\stack@top{\@refsubstack}}%
          \count@consecutive=1%
```

Test whether next reference is empty;

```
          \ifx\@nextref\@empty%
            \refconsecutivefalse%
            \@whilesw\ifx\@nextref\@empty\fi{%
              \stack@pop{\@refsubstack}%
              \isstackempty{\@refsubstack}%
              \ifstackempty%
                \let\@nextref\relax%
              \else%
                \edef\@nextref{\stack@top{\@refsubstack}}%
              \fi%
            }%
            \ifnum\count@consecutive=2%
              \edef\@endref{\@endref,}%
            \fi%
```

otherwise, test whether next reference is consecutive or not.

```
          \else
            \expandafter\ifx\csname r@\@nextref\endcsname\relax%
              \refconsecutivefalse%
            \else%
              \edef\@tmpa{{\@beginref}{\@nextref}}%
              \expandafter\isrefconsecutive\@tmpa%
            \fi%
          \fi%
          \@whilesw\ifrefconsecutive\fi{%
            \advance\count@consecutive 1%
            \let\@endref\@nextref%
```

```
            \stack@pop{\@refsubstack}%
            \isstackempty{\@refsubstack}%
            \ifstackempty%
              \refconsecutivefalse%
            \else%
              \edef\@nextref{\stack@top{\@refsubstack}}%
```

Test whether next reference is empty;

```
              \ifx\@nextref\@empty%
              \refconsecutivefalse%
              \@whilesw\ifx\@nextref\@empty\fi{%
                \stack@pop{\@refsubstack}%
                \isstackempty{\@refsubstack}%
                \ifstackempty%
                  \let\@nextref\relax%
                \else%
                  \edef\@nextref{\stack@top{\@refsubstack}}%
                \fi%
              }%
              \ifnum\count@consecutive=2%
                \edef\@endref{\@endref,}%
              \fi%
```

otherwise, test whether next reference is consecutive or not.

```
              \else
                \expandafter\ifx\csname r@\@nextref\endcsname\relax%
                  \refconsecutivefalse%
                \else%
                  \edef\@tmpa{{\@endref}{\@nextref}}%
                  \expandafter\isrefconsecutive\@tmpa%
                \fi%
              \fi%
            \fi}%
```

If the substack is now empty, we will need to typeset an "end" reference, otherwise we will need to typeset a "middle" reference.

```
          \isstackempty{\@refsubstack}%
          \ifstackempty%
            \def\@pos{@last}%
          \else%
            \def\@pos{@middle}%
          \fi%
```

If there were no consecutive references, just typeset the next reference;

```
          \ifx\@endref\relax%
            \edef\@tmpa{{\@beginref}{#1}{\@pos}}%
            \expandafter\@setcref\@tmpa%
          \else%
```

if there were only two consecutive references, typeset the first one, and return the second one to the substack,

```
            \ifnum\count@consecutive=2%
              \expandafter\@setcref\expandafter%
                {\@beginref}{#1}{@middle}%
              \expandafter\stack@push\expandafter%
                {\@endref}{\@refsubstack}%
```

otherwise, typeset a reference range.

```
            \else
              \edef\@tmpa{{\@beginref}{\@endref}{#1}{\@pos}}%
              \expandafter\@setcrefrange\@tmpa%
            \fi%
          \fi%
        \fi%
        \isstackfull{\@refsubstack}%
      }% end loop over reference substack
    \fi%
    \isstackfull{\@refstack}%
  }% end loop over main reference stack
  \endgroup}
```

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

```
\def\@setcref#1#2#3{%
  \expandafter\ifx\csname r@#1\endcsname\relax%
    \protect\G@refundefinedtrue%
    \nfss@text{\reset@font\bfseries ??}%
    \@latex@warning{Reference `#1' on page \thepage \space undefined}%
  \else%
    \cref@gettype{#1}{\@temptype}%  puts label type in \@temptype
    \cref@getlabel{#1}{\@templabel}%  puts label in \@templabel
    \expandafter\ifx\csname #2@\@temptype @format#3\endcsname\relax%
      \protect\G@refundefinedtrue%
      \nfss@text{\reset@font\bfseries ??}~\@templabel%
      \@latex@warning{\string\Cref \space reference format for label
        type `\@temptype' undefined}%
    \else%
      \expandafter\@@setcref\expandafter%
        {\csname #2@\@temptype @format#3\endcsname}{#1}%
    \fi%
  \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.

```
\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 prefices are identical (i.e. the references come from the same chapter, section or whatever), and that the label counters differ by 1.

```
\newif\ifrefconsecutive%
\def\isrefconsecutive#1#2{%
  \begingroup%
    \countdef\refa@counter=1%
    \countdef\refb@counter=2%
    \cref@getcounter{#1}{\@result}%
    \refa@counter=\@result%
```

```
    \advance\refa@counter 1%
    \cref@getcounter{#2}{\@result}%
    \refb@counter=\@result%
    \cref@getprefix{#1}{\refa@prefix}%
    \cref@getprefix{#2}{\refb@prefix}%
    \def\@after{\refconsecutivefalse}%
    \ifx\refa@prefix\refb@prefix%
      \ifnum\refa@counter=\refb@counter%
        \def\@after{\refconsecutivetrue}%
      \fi%
    \fi%
  \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.

```
\def\@setcrefrange#1#2#3#4{%
```

Check if both references are defined.

```
  \expandafter\ifx\csname r@#1\endcsname\relax%
    \protect\G@refundefinedtrue%
    \@latex@warning{Reference `#1' on page \thepage \space undefined}%
    \expandafter\ifx\csname r@#2\endcsname\relax%
      \nfss@text{\reset@font\bfseries ??}--%
      \nfss@text{\reset@font\bfseries ??}%
      \@latex@warning{Reference `#2' on page \thepage \space undefined}%
    \else%
      \cref@getlabel{#2}{\@labelb}%
      \nfss@text{\reset@font\bfseries ??}--\@labelb%
    \fi%
  \else%
    \expandafter\ifx\csname r@#2\endcsname\relax%
      \protect\G@refundefinedtrue%
      \cref@getlabel{#1}{\@labela}%
      \@labela--\nfss@text{\reset@font\bfseries ??}%
      \@latex@warning{Reference `#2' on page \thepage \space undefined}%
```

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

```
    \else%
      \cref@gettype{#1}{\@typea}%
      \cref@gettype{#2}{\@typeb}%
      \cref@getlabel{#1}{\@labela}%
      \cref@getlabel{#2}{\@labelb}%
      \edef\@formata{\expandafter\noexpand%
        \csname #3range@\@typea @format#4\endcsname}%
      \edef\@formatb{\expandafter\noexpand%
        \csname #3range@\@typeb @format#4\endcsname}%
      \expandafter\ifx\@formata\relax%
        \protect\G@refundefinedtrue%
        \nfss@text{\reset@font\bfseries ??}~\@labela--\@labelb%
        \@latex@warning{#3 \space reference format for label
          type `\@typea' undefined}%
```

```
      \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.)

```
        \ifx\formata\formatb%
          \expandafter\@@setcrefrange\expandafter{\@formata}{#1}{#2}%
        \else%
          \protect\G@refundefinedtrue%
          \nfss@text{\reset@font\bfseries ??}~\@labela--\@labelb%
          \@latex@warning{Types inconsistent in reference range for
            references `#1' and `#2' on page \thepage}%
        \fi%
      \fi%
    \fi%
  \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.

```
\def\@@setcrefrange#1#2#3{%
  \cref@getlabel{#2}{\@labela}%
  \cref@getlabel{#3}{\@labelb}%
  #1{\@labela}{\@labelb}{}{}{}{}}
```

The typesetting of conjunctions is also separated out into separate macros, for the same reason.

```
\def\@setcref@middleconjunction{\crefmiddleconjunction}
\def\@setcref@lastconjunction{\creflastconjunction}
```

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.

```
\newcommand{\crefformat}[2]{\@crefformat{cref}{#1}{#2}}
\newcommand{\Crefformat}[2]{\@crefformat{Cref}{#1}{#2}}
\newcommand{\crefrangeformat}[2]{\@crefrangeformat{crefrange}{#1}{#2}}
\newcommand{\Crefrangeformat}[2]{\@crefrangeformat{Crefrange}{#1}{#2}}
\newcommand{\crefmultiformat}[4]{%
  \@crefmultiformat{cref}{#1}{#2}{#3}{#4}}
\newcommand{\Crefmultiformat}[4]{%
  \@crefmultiformat{Cref}{#1}{#2}{#3}{#4}}
\newcommand{\crefrangemultiformat}[4]{%
  \@crefrangemultiformat{crefrange}{#1}{#2}{#3}{#4}}
\newcommand{\Crefrangemultiformat}[4]{%
  \@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.

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

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

```
  \def\@tmpa##1##2\@nil{%
    \if##1c \def\@other{C##2}%
    \else\def\@other{c##2}\fi}%
  \@tmpa#1\@nil%
  \@ifundefined{\@other @#2@format}{%
    \def\@tmpa##1##2\@nil{%
      \if##1c\def\@changecase{\MakeUppercase}%
      \else\def\@changecase{\Makelowercase}\fi}%
    \@tmpa#1\@nil%
    \newtoks\@toksa%
    \@toksa={\def\@tmpa##1##2##3}%
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@format\endcsname{##1}{##2}{##3}}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \expandafter\MakeUppercase\@tmpa{##1}{##2}{##3}}%
    \@toksa={%
      \expandafter\def\csname\@other @#2@format\endcsname##1##2##3}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \@tmpa{##1}{##2}{##3}}%
  }{}%
}
%
\def\@crefrangeformat#1#2#3{%
  \expandafter\def\csname #1@#2@format\endcsname%
    ##1##2##3##4##5##6{#3}%
```

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

```
  \def\@tmpa##1##2\@nil{%
    \if##1c \def\@other{C##2}%
    \else\def\@other{c##2}\fi}%
  \@tmpa#1\@nil%
  \@ifundefined{\@other @#2@format}{%
    \def\@tmpa##1##2\@nil{%
      \if##1c\def\@changecase{\MakeUppercase}%
      \else\def\@changecase{\Makelowercase}\fi}%
    \@tmpa#1\@nil%
    \newtoks\@toksa%
    \@toksa={\def\@tmpa##1##2##3##4##5##6}%
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@format\endcsname{##1}{##2}{##3}{##4}{##5}{##6}}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \expandafter\MakeUppercase\@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}
    \@toksa={\expandafter\def%
      \csname\@other @#2@format\endcsname##1##2##3##4##5##6}%
    \expandafter\the\expandafter\@toksa\expandafter{%
```

```
      \@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}%
  }{}%
}
%
\def\@crefmultiformat#1#2#3#4#5{%
  \expandafter\def\csname #1@#2@format@first\endcsname##1##2##3{#3}%
  \expandafter\def\csname #1@#2@format@middle\endcsname##1##2##3{#4}%
  \expandafter\def\csname #1@#2@format@last\endcsname##1##2##3{#5}%
```

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

```
  \newtoks\@toksa%
  \def\@tmpa##1##2\@nil{%
    \if##1c \def\@other{C##2}%
    \else\def\@other{c##2}\fi}%
  \@tmpa#1\@nil%
  \@ifundefined{\@other @#2@format@first}{%
    \def\@tmpa##1##2\@nil{%
      \if##1c\def\@changecase{\MakeUppercase}%
      \else\def\@changecase{\Makelowercase}\fi}%
    \@tmpa#1\@nil%
    \@toksa={\def\@tmpa##1##2##3}%
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@format@first\endcsname{##1}{##2}{##3}}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \expandafter\MakeUppercase\@tmpa{##1}{##2}{##3}}%
    \@toksa={%
      \expandafter\def\csname\@other @#2@format@first\endcsname%
        ##1##2##3}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \@tmpa{##1}{##2}{##3}}%
  }{}
  \@ifundefined{\@other @#2@format@middle}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@middle\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@middle\endcsname%
  }{}%
  \@ifundefined{\@other @#2@format@last}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@last\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@last\endcsname%
  }{}%
}
%
\def\@crefrangemultiformat#1#2#3#4#5{%
  \expandafter\def\csname #1@#2@format@first\endcsname%
    ##1##2##3##4##5##6{#3}%
  \expandafter\def\csname #1@#2@format@middle\endcsname%
    ##1##2##3##4##5##6{#4}%
  \expandafter\def\csname #1@#2@format@last\endcsname%
```

```
    ##1##2##3##4##5##6{#5}%
```

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

```
  \def\@tmpa##1##2\@nil{%
    \if##1c \def\@other{C##2}%
    \else\def\@other{c##2}\fi}%
  \@tmpa#1\@nil%
  \@ifundefined{\@other @#2@format@first}{%
    \def\@tmpa##1##2\@nil{%
      \if##1c\def\@changecase{\MakeUppercase}%
      \else\def\@changecase{\MakeLowercase}\fi}%
    \@tmpa#1\@nil%
    \@toksa={\def\@tmpa##1##2##3##4##5##6}%
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@format@first\endcsname%
        {##1}{##2}{##3}{##4}{##5}{##6}}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \expandafter\MakeUppercase\@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}%
    \@toksa={%
      \expandafter\def\csname\@other @#2@format@first\endcsname%
        ##1##2##3##4##5##6}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}%
  }{}%
  \@ifundefined{\@other @#2@format@middle}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@middle\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@middle\endcsname%
  }{}%
  \@ifundefined{\@other @#2@format@last}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@last\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@last\endcsname%
  }{}%
}
```

6.5 Default Reference Formats

Define the default reference formats, appropriate for LaTeX documents written in English. We define the lowercase and capitalised versions separately, rather than relying on the automatic definitions, because the code produced by the poor man's sed script is then slightly tidier.

To-Do: add babel support

```
\crefformat{equation}{eq.~\textup{(#2#1#3)}}
\Crefformat{equation}{Equation~\textup{(#2#1#3)}}
\crefformat{chapter}{chapter~#2#1#3}
\Crefformat{chapter}{Chapter~#2#1#3}
\crefformat{section}{section~#2#1#3}
\Crefformat{section}{Section~#2#1#3}
```

```
\crefformat{subsection}{section~#2#1#3}
\Crefformat{subsection}{Section~#2#1#3}
\crefformat{subsubsection}{section~#2#1#3}
\Crefformat{subsubsection}{Section~#2#1#3}
\crefformat{subsubsubsection}{section~#2#1#3}
\Crefformat{subsubsubsection}{Section~#2#1#3}
\crefformat{figure}{fig.~#2#1#3}
\Crefformat{figure}{Figure~#2#1#3}
\crefformat{theorem}{theorem~#2#1#3}
\Crefformat{theorem}{Theorem~#2#1#3}
\crefformat{enumi}{item~#2#1#3}
\Crefformat{enumi}{Item~#2#1#3}
\crefformat{enumii}{item~#2#1#3}
\Crefformat{enumii}{Item~#2#1#3}
\crefformat{enumiii}{item~#2#1#3}
\Crefformat{enumiii}{Item~#2#1#3}
\crefformat{enumiv}{item~#2#1#3}
\Crefformat{enumiv}{Item~#2#1#3}
%
\crefmultiformat{equation}%
  {eqs.~\textup{(#2#1#3)}}%
  {, \textup{(#2#1#3)}}%
  { and~\textup{(#2#1#3)}}
\Crefmultiformat{equation}%
  {Equations~\textup{(#2#1#3)}}%
  {, \textup{(#2#1#3)}}%
  { and~\textup{(#2#1#3)}}
\crefmultiformat{chapter}%
  {chapters~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{chapter}%
  {Chapters~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{section}%
  {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{section}%
  {Sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{subsection}%
  {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{subsection}%
  {Sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{subsubsection}%
  {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{subsubsection}%
  {Sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{subsubsubsection}%
  {sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{subsubsubsection}%
  {Sections~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{figure}%
  {figs.~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{figure}%
  {Figures~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{theorem}%
  {theorems~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{theorem}%
```

```
  {Theorems~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{enumi}%
  {items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{enumi}%
  {Items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{enumii}%
  {items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{enumii}%
  {Items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{enumiii}%
  {items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{enumiii}%
  {Items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{enumiv}%
  {items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{enumiv}%
  {Items~#2#1#3}{, #2#1#3}{ and~#2#1#3}
%
\crefrangeformat{equation}{%
  eqs.~\textup{(#3#1#4)}--\textup{(#5#2#6)}}
\Crefrangeformat{equation}{%
  Equations~\textup{(#3#1#4)}--\textup{(#5#2#6)}}
\crefrangeformat{chapter}{chapters~#3#1#4--#5#2#6}
\Crefrangeformat{chapter}{Chapters~#3#1#4--#5#2#6}
\crefrangeformat{section}{sections~#3#1#4--#5#2#6}
\Crefrangeformat{section}{Sections~#3#1#4--#5#2#6}
\crefrangeformat{subsection}{sections~#3#1#4--#5#2#6}
\Crefrangeformat{subsection}{Sections~#3#1#4--#5#2#6}
\crefrangeformat{subsubsection}{sections~#3#1#4--#5#2#6}
\Crefrangeformat{subsubsection}{Sections~#3#1#4--#5#2#6}
\crefrangeformat{subsubsubsection}{sections~#3#1#4--#5#2#6}
\Crefrangeformat{subsubsubsection}{Sections~#3#1#4--#5#2#6}
\crefrangeformat{figure}{figs.~#3#1#4--#5#2#6}
\Crefrangeformat{figure}{Figures~#3#1#4--#5#2#6}
\crefrangeformat{theorem}{theorems~#3#1#4--#5#2#6}
\Crefrangeformat{theorem}{Theorems~#3#1#4--#5#2#6}
\crefrangeformat{enumi}{items~#3#1#4--#5#2#6}
\Crefrangeformat{enumi}{Items~#3#1#4--#5#2#6}
\crefrangeformat{enumii}{items~#3#1#4--#5#2#6}
\Crefrangeformat{enumii}{Items~#3#1#4--#5#2#6}
\crefrangeformat{enumiii}{items~#3#1#4--#5#2#6}
\Crefrangeformat{enumiii}{Items~#3#1#4--#5#2#6}
\crefrangeformat{enumiv}{items~#3#1#4--#5#2#6}
\Crefrangeformat{enumiv}{Items~#3#1#4--#5#2#6}
%
\crefrangemultiformat{equation}%
  {eqs.~\textup{(#3#1#4)}--\textup{(#5#2#6)}}%
  {, \textup{(#3#1#4)}--\textup{(#5#2#6)}}%
  { and~\textup{(#3#1#4)}--\textup{(#5#2#6)}}
\Crefrangemultiformat{equation}%
  {Equations~\textup{(#3#1#4)}--\textup{(#5#2#6)}}%
  {, \textup{(#3#1#4)}--\textup{(#5#2#6)}}%
  { and~\textup{(#3#1#4)}--\textup{(#5#2#6)}}
\crefrangemultiformat{chapter}%
```

```
  {chapters~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{chapter}%
  {Chapters~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{section}%
  {sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{section}%
  {Sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{subsection}%
  {sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{subsection}%
  {Sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{subsubsection}%
  {sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{subsubsection}%
  {Sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{subsubsubsection}%
  {sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{subsubsubsection}%
  {Sections~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{figure}%
  {figs.~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{figure}%
  {Figures~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{theorem}%
  {theorems~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{theorem}%
  {Theorems~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
```

```
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{enumi}%
  {items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{enumi}%
  {Items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{enumii}%
  {items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{enumii}%
  {Items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{enumiii}%
  {items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{enumiii}%
  {Items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{enumiv}%
  {items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{enumiv}%
  {Items~#3#1#4--#5#2#6}%
  {, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
%
\def\crefmiddleconjunction{, }
\def\creflastconjunction{, and }
```

6.6 hyperref Support

```
\DeclareOption{hyperref}{%
\PackageInfo{cleveref}{option `hyperref' loaded}
```

We redefine the utility macros to cope with the extra arguments supplied by hyperref (via the aux file).

```
\def\cref@label#1#2#3#4#5{\@result}
\def\cref@hyperref#1{\expandafter\expandafter\expandafter%
  \@fourthoffive\csname r@#1\endcsname}
\def\cref@getlabel#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname%
  \edef\@tempa{\expandafter\@firstoffive\@tempa}%
  \expandafter\@cref@getlabel\@tempa\@nil#2}
\def\cref@gettype#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
  \edef\@tempa{\expandafter\@firstoffive\@tempa}%
```

```
  \expandafter\@cref@gettype\@tempa\@nil#2}
\def\cref@getcounter#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
  \edef\@tempa{\expandafter\@firstoffive\@tempa}%
  \expandafter\@cref@getcounter\@tempa\@nil#2}
\def\cref@getprefix#1#2{%
  \expandafter\let\expandafter\@tempa\csname r@#1\endcsname
  \edef\@tempa{\expandafter\@firstoffive\@tempa}%
  \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`.

```
\def\H@refstepcounter#1{%
  \stepcounter{#1}%
  \reset@by{#1}{\@result}%
  \ifx\@result\relax\def\@result{}%
  \else\edef\@result{\csname the\@result\endcsname}\fi%
  \protected@edef\@currentlabel%
    {[#1][\arabic{#1}][\@result]%
      \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.

```
\def\refstepcounter@noarg#1{\old@refstepcounter{#1}}
\def\refstepcounter@optarg[#1]#2{%
  \old@refstepcounter{#2}%
  \expandafter\@cref@getlabel\@currentlabel\@nil{\@templabel}%
  \reset@by{#2}{\@tempreset}%
  \ifx\@tempreset\relax\def\@tempreset{}%
  \else\edef\@tempreset{\csname the\@tempreset\endcsname}\fi%
  \protected@edef\@currentlabel{%
    [#1][\arabic{#2}][\@tempreset]\@templabel}}
```

Redefine `\cref` and all the others to allow starred variants, which don't create hyper-links. The starred variants simply set a flag, which is tested in `\@@setcref` and `\@@setrangeref` (below).

```
\newif\if@crefstarred
\DeclareRobustCommand{\cref}{%
  \@ifstar{\@crefstar{cref}}{\@crefnostar{cref}}}
\DeclareRobustCommand{\Cref}{%
  \@ifstar{\@crefstar{Cref}}{\@crefnostar{Cref}}}
\DeclareRobustCommand{\crefrange}{%
  \@ifstar{\@crefrangestar{cref}}{\@crefrangenostar{cref}}}
\DeclareRobustCommand{\Crefrange}{%
  \@ifstar{\@crefrangestar{Cref}}{\@crefrangenostar{Cref}}}
\def\@crefnostar#1#2{\@cref{#1}{#2}}
\def\@crefstar#1#2{\@crefstarredtrue\@cref{#1}{#2}\@crefstarredfalse}
\def\@crefrangenostar#1#2#3{\@setcrefrange{#2}{#3}{#1}{}}
\def\@crefrangestar#1#2#3{%
  \@crefstarredtrue\@setcrefrange{#2}{#3}{#1}{}\@crefstarredfalse}
```

Redefine @setcref and @setrangeref to create hyper-links (unless the starred flag is set), using the extra arguments supplied in `\r@`⟨*label*⟩ (via the aux file)

by hyperref.

```
\def\@@setcref#1#2{%
  \cref@getlabel{#2}{\@templabel}%
  \if@crefstarred%
    #1{\@templabel}{}{}%
  \else%
    \edef\@templink{\cref@hyperref{#2}}%
    #1{\@templabel}{\hyper@linkstart{link}{\@templink}}{\hyper@linkend}%
  \fi}
\def\@@setcrefrange#1#2#3{%
  \cref@getlabel{#2}{\@labela}%
  \cref@getlabel{#3}{\@labelb}%
  \if@crefstarred%
    #1{\@labela}{\@labelb}{}{}{}{}%
  \else%
    \edef\@linka{\cref@hyperref{#2}}%
    \edef\@linkb{\cref@hyperref{#3}}%
    #1{\@labela}{\@labelb}%
      {\hyper@linkstart{link}{\@linka}}{\hyper@linkend}%
      {\hyper@linkstart{link}{\@linkb}}{\hyper@linkend}%
  \fi}
} % end of hyperref option
```

6.7 ntheorem Support

```
\DeclareOption{ntheorem}{%
\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.

```
\gdef\@thm#1#2#3{%
   \if@thmmarks
     \stepcounter{end\InTheoType ctr}%
   \fi
   \renewcommand{\InTheoType}{#1}%
   \if@thmmarks
     \stepcounter{curr#1ctr}%
     \setcounter{end#1ctr}{0}%
   \fi
   \refstepcounter[#1]{#2}%  <<<<<
   \thm@topsepadd \theorempostskipamount
   \ifvmode \advance\thm@topsepadd\partopsep\fi
   \trivlist
   \@topsep \theorempreskipamount
   \@topsepadd \thm@topsepadd
   \advance\linewidth -\theorem@indent
   \advance\@totalleftmargin \theorem@indent
   \parshape \@ne \@totalleftmargin \linewidth
   \@ifnextchar[{\@ythm{#1}{#2}{#3}}{\@xthm{#1}{#2}{#3}}%]
}
```

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

```
\crefformat{lemma}{lemma~#2#1#3}
\Crefformat{lemma}{Lemma~#2#1#3}
\crefformat{corrollary}{corrollary~#2#1#3}
\Crefformat{corrollary}{Corrollary~#2#1#3}
```

```
\crefformat{proposition}{proposition~#2#1#3}
\Crefformat{proposition}{Proposition~#2#1#3}
\crefformat{definition}{definition~#2#1#3}
\Crefformat{definition}{Definition~#2#1#3}
\crefformat{result}{result~#2#1#3}
\Crefformat{result}{Result~#2#1#3}
\crefrangeformat{lemma}{lemma~#3#1#4--#5#2#6}
\Crefrangeformat{lemma}{Lemma~#3#1#4--#5#2#6}
\crefrangeformat{corrollary}{corrollary~#3#1#4--#5#2#6}
\Crefrangeformat{corrollary}{Corrollary~#3#1#4--#5#2#6}
\crefrangeformat{proposition}{proposition~#3#1#4--#5#2#6}
\Crefrangeformat{proposition}{Proposition~#3#1#4--#5#2#6}
\crefrangeformat{definition}{definition~#3#1#4--#5#2#6}
\Crefrangeformat{definition}{Definition~#3#1#4--#5#2#6}
\crefrangeformat{result}{result~#3#1#4--#5#2#6}
\Crefrangeformat{result}{Result~#3#1#4--#5#2#6}
\crefmultiformat{lemma}%
  {lemmas~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{lemma}%
  {Lemmas~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{corrollary}%
  {corrollaries~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{corrollary}%
  {Corrollaries~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{proposition}%
  {propositions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{proposition}%
  {Propositions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{definition}%
  {definitions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{definition}%
  {Definitions~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefmultiformat{result}%
  {results~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\Crefmultiformat{result}%
  {Results~#2#1#3}{, #2#1#3}{ and~#2#1#3}
\crefrangemultiformat{lemma}%
  {lemmas~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}%
\Crefrangemultiformat{lemma}%
  {Lemmas~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{corrollary}%
  {corrollaries~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{corrollary}%
  {Corrollaries~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{proposition}%
  {propositions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{proposition}%
  {Propositions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
```

```
\crefrangemultiformat{definition}%
  {definitions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{definition}%
  {Definitions~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\crefrangemultiformat{result}%
  {results~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
\Crefrangemultiformat{result}%
  {Results~#3#1#4--#5#2#6}{, #3#1#4--#5#2#6}%
  { and~#3#1#4--#5#2#6}
} % end of ntheorem option
```

6.8 Poor Man's cleveref

```
\DeclareOption{poorman}{%
\PackageInfo{cleveref}{option 'poorman' loaded}
```

Define global macro `\cref@text` to store the text produced by the `\cref` commands, and open an output stream for writing the script before starting to process to document body.

```
\edef\cref@text{}
\AtBeginDocument{%
  \newwrite\@crefscript%
  \immediate\openout\@crefscript=\jobname.sed%
}
```

After processing the document body, we re-read in the temporary script file, and write it out again to the final sed script file, escaping regexp special characters in the process. The escaping is carried out by turning the regexp special characters into active characters, and defining them to expand to their escaped form. This involves a lot of juggling of catcodes and lccodes!

Both `\DeclareOption` and `\AtEndDocument` store their arguments in token lists, so all the following TEXcode is already tokenised long before it is expanded and evaluated. Thus there is no (easy) way to change the catcodes of the characters appearning here before they are tokenised. In one way this is convenient: the catcode changes we make don't "take" until evaluated, so we can continue to use the standard TEXcharacters (\, {, } etc.) even after the lines containing the catcode commands. But in another, more significant, way, it is very inconvenient: it makes it difficult to define the regexp special characters as active characters, since it's impossible to directly create tokens with the correct char- and catcodes.

We get around this by creating the unusual charcode/catcode combinations using the `\lowercase` trick (`\lowercase` changes the charcodes of all characters in its argument to their lccodes, but *leaves* their catcodes alone). That way, the argument of `\AtEndDocument` is tokenised correctly, and when it comes to be expanded and evaluated, the `\lowercase` commands create tokens with the correct char- and catcodes.

```
\AtEndDocument{%
  \immediate\closeout\@crefscript%
  \newread\@crefscript%
  \immediate\openin\@crefscript=\jobname.sed%
  \begingroup%
    \newif\if@not@eof%
```

```
    \def\@eof{\par }%
```

Change catcodes of regexp special characters to make them active characters and define them to expand to their escaped forms. Change those of TEXspecial characters to make them normal letters.

```
    \catcode`.=13 \catcode`[=13 \catcode`]=13
    \catcode`^=13 \catcode`$=13 %$
    \catcode`\=0 \catcode`<=1 \catcode`>=2
    \catcode`\\=13 \catcode`\{=12 \catcode`\}=12 \catcode`_=12
    \lccode`/=92
    \lccode`~=92\lowercase{\def~{\string/\string/}}%
    \lccode`~=46\lowercase{\def~{\string/\string.}}%
    \lccode`~=91\lowercase{\def~{\string/\string[}}%
    \lccode`~=93\lowercase{\def~{\string/\string]}}%
    \lccode`~=94\lowercase{\def~{\string/\string^}}%
    \lccode`~=36\lowercase{\def~{\string/\string$}}% $
    \lccode`~=0 \lccode`/=0 \catcode`~=12
```

Read lines from the temporary script file, expand them to escape regexp special characters, and store them in `\cref@text`.

```
    \def\cref@text{}%
    \immediate\read\@crefscript to \@tmpa%
    \edef\@tmpa{\@tmpa}%
    \ifx\@tmpa\@eof%
      \@not@eoffalse%
    \else%
      \@not@eoftrue%
    \fi%
    \@whilesw\if@not@eof\fi{%
      \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
        \@tmpa^^J}%
      \immediate\read\@crefscript to \@tmpa%
      \edef\@tmpa{\@tmpa}%
      \ifx\@tmpa\@eof%
        \@not@eoffalse%
      \else%
        \@not@eoftrue%
      \fi}%
  \endgroup%
  \immediate\closein\@crefscript%
```

Add some rules to remove likely `cleveref` preamble commands. We use the `\lowercase` trick again for writing the \, { and } characters. (This could be done in other ways, but since we're in `\lowercase` mood, why not go with it!)

```
  \begingroup%
    \lccode`|=92 \lccode`<=123 \lccode`>=125 \lccode`C=67
    \lowercase{\edef\@tmpa{s/||usepackage|(|[.*|]|)|?<cleveref>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{s/||[cC]refformat<.*><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{s/||[cC]refrangeformat<.*><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{s/||[cC]refmultiformat<.*><.*><.*><.*>//g}}%
```

```
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{%
        s/||[cC]refrangemultiformat<.*><.*><.*><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{s/||renewcommand<||[cC]ref><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}
    \lowercase{\edef\@tmpa{s/||renewcommand<||[cC]refrange><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\%
    expandafter{\@tmpa}
  \endgroup
```

Overwrite the script file with the new, escaped regexp rules.

```
  \newwrite\@crefscript%
  \immediate\openout\@crefscript=\jobname.sed%
  \immediate\write\@crefscript{\cref@text}%
  \immediate\closeout\@crefscript%
}%
```

Redefine the user-level referencing commands so that they write a substitution rule for the reference to the script, as well as typesetting the reference itself.

```
\renewcommand{\cref}[1]{%
  \edef\cref@text{}%
  \@cref{cref}{#1}%
  \cref@writescript{\string\cref\string{#1\string}}}
\renewcommand{\Cref}[1]{%
  \edef\cref@text{}%
  \@cref{Cref}{#1}%
  \cref@writescript{\string\Cref\string{#1\string}}}
\renewcommand{\crefrange}[2]{%
  \edef\cref@text{}%
  \@setcrefrange{#1}{#2}{cref}{}%
  \cref@writescript{%
    \string\crefrange\string{#1\string}\string{#2\string}}}
\renewcommand{\Crefrange}[2]{%
  \edef\cref@text{}%
  \@setcrefrange{#1}{#2}{Cref}{}%
  \cref@writescript{%
    \string\Crefrange\string{#1\string}\string{#2\string}}}
```

The `\cref@writescript` utility macro does the actual writing of the substitution rule to the script.

```
\def\cref@writescript#1{%
  \edef\@tmpa{\cref@getmeaning{\cref@text}}%
  \immediate\write\@crefscript{s/#1/\@tmpa/g}%
}
```

Redefine `\@@setcref`, `\@@setrangeref`, `\@setcref@middleconjunction` and `\@setcref@lastconjunction` to append the text they typeset to `\cref@text`, as well as actually typesetting it.

```
\let\old@@setcref\@@setcref
\let\old@@setcrefrange\@@setcrefrange
\def\cref@getmeaning#1{\expandafter\@cref@getmeaning\meaning#1\@nil}
\def\@cref@getmeaning#1->#2\@nil{#2}
```

```
\def\@@setcref#1#2{%
  \old@@setcref{#1}{#2}%
  \expandafter\g@addto@macro\expandafter{%
    \expandafter\cref@text\expandafter}\expandafter{%
    #1{\ref{#2}}{}{}}}
\def\@@setcrefrange#1#2#3{%
  \old@@setcrefrange{#1}{#2}{#3}%
  \expandafter\g@addto@macro
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{#1{\ref{#2}}{\ref{#3}}{}{}{}{}}}
\def\@setcref@middleconjunction{%
  \crefmiddleconjunction%
  \expandafter\g@addto@macro
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{\crefmiddleconjunction}}
\def\@setcref@lastconjunction{%
  \creflastconjunction%
  \expandafter\g@addto@macro
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{\creflastconjunction}}
} % end of poorman option
```

Process options.

```
\ProcessOptions\relax
```