The `cleveref` package*

Toby Cubitt
toby-cleveref@dr-qubit.org

30/10/2007

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.) and the context in which the reference is used. 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 about his achievements! 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 probably valid when it comes 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 facilities by allowing references to be formatted automatically according to the type of object they refer to (chapter, section, equation, theorem, etc.) and the context in which the reference is used. It can also automatically format references to multiple labels, automatically collapsing references to consecutive labels into a reference range, and all kinds of other clever wizardry.

In standard LaTeX, you have almost certainly found yourself writing things like `Eq.~(\ref{eq1})` and `Theorems~\ref{thm1} to~\ref{thm2}` over and over

*This document corresponds to cleveref 0.10, dated 30/10/2007.

again. Tedium isn't the only downside to this. 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 LaTeX source of your document, modifying all references to equations, or changing all references to `thm1`.

The `cleveref` package 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.

There are a number of other packages with similar goals, 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 missing out on automatic collapsing of consecutive references, `ntheorem` support, etc.

The `hyperref` package's `\autoref` command typesets a name before a reference, determined by 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 hyper-link 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 that specifies the label type. `cleveref`

works equally well when referencing any type of object, as well as fully supporting `ntheorem`. And `cleveref` provides a number additional features, 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 7). Basically, `cleveref` must be loaded *last*.

If you just want to get going quickly with `cleveref`, and come back later to read up on all the features it provides in more detail, then here's what you need to do. Wherever you would previously have used `\ref`, use `\cref` instead. (Except that at the beginning of a sentence, you should use `\Cref`.) You no longer need to put the name of the thing you're referencing in front of the `\cref` command, because `cleveref` will sort that out for you: i.e. use `\cref{eq1}` instead of `Eq.~\ref{eq1}`. If you want to refer to a range of labels, use `\crefrange`: `\crefrange{eq1}{eq5}`. Finally, if you want to refer to multiple things at once, you can now combine them all into one reference and leave `cleveref` to sort it out: e.g. `\cref{eq1,eq2,eq4,thm2,def1}`.

3 Typesetting References

`\cref` To automatically typeset a cross-reference according to the type of object referred to, simply refer to it using `\cref{`⟨*label*⟩`}`. `cleveref` imposes just one extra restriction on the names of labels: they are no longer allowed to contain commas ",". These are instead used to typeset multiple references (see below).

`\Cref` As it is very difficult[1] for LaTeX to determine whether a cross-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 fully and independently customised, see section 4.)

`\ref` `cleveref` does *not* modify the standard `\ref` command[2], so you can still use it to typeset the formatted label counter alone, without any additional text or formatting.

`\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:

[1] Actually, very likely impossible!

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

`\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,

```
\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)

(assuming you haven't customised the reference formats).

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 at which 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 reference between references that would never be collapsed anyway; it will simply be ignored.

`\cref*` `\Cref*` `\crefrange*` `\Crefrange*` When `cleveref` is used along with the `hyperref` package (see sections 4 and 7), 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.

4 Customising the Reference Format

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[3]. But if you don't like them, or are writing in a different language, or you need to cross-reference something for which no default format is defined, then you can take charge and define your own formats.

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. The error message indicates the name of the unknown reference type, which you will then probably want to define. (References to undefined labels still produce a "reference undefined" warning and are typeset as a double question mark, as usual.)

The reference formats are usually constructed out of components: the reference name (different for each type of reference), the label formats, and the conjunctions used in reference ranges and lists of multiple references. There are three levels of customisation: you can customise the components globally, individually for each reference type, or you can take full control and ignore the components entirely.

4.1 Global Customisation

`\crefdefaultlabelformat`

The label format can be customised globally using

`\crefdefaultlabelformat{`⟨*format*⟩`}`

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 7). The hyper-link arguments `#2` and `#3` *must* appear in that order. (Leaving them out completely will not cause an error, but in that case no hyper-link will be created when `hyperref` is used, and there are better ways to achieve this. See section 7.)

`\crefrangeconjunction`

The conjunction used in a reference range can be customised by redefining `\crefrangeconjunction`:

`\renewcommand{\crefrangeconjunction}{`⟨*conjunction*⟩`}`

[3] For any pedantic classics scholars out there: "lemmas" is recognised as a valid plural form of "lemma" in all current versions of the Oxford English Dictionary. "Lemmata" was last heard used in a mathematical debate that took place in a pub just around the corner from Hadrian's wall... a few years before the Romans pulled out of Britain. `cleveref` might have "clever" in its name, but even that doesn't make it pretentious enough to use "lemmata".

It does not have to be an actual conjunction in the linguistic sense, e.g. it is perfectly reasonable to define it to be an emdash "--". `\crefrangeconjunction` is used directly between the start and end references in a reference range, without any additional space surrounding it, e.g. `\crefrange{eq1}{eq2}` is typeset as

```
eqs.~\ref{eq1}\crefrangeconjunction\ref{eq2}
```

(assuming the default equation name hasn't been customised). So you may or may not want to include surrounding space, depending on the formatting you desire. For example,

```
\renewcommand{\crefrangeconjunction}{ and~}
```

does require surrounding space, whereas

```
\renewcommand{\crefrangeconjunction}{--}
```

does not.

`\crefpairconjunction`
`\crefmiddleconjunction`
`\creflastconjunction`

The conjunctions used in lists of references can be customised by redefining `\crefpairconjunction`, `\crefmiddleconjunction` and `\creflastconjunction`:

```
\renewcommand{\crefpairconjunction}{⟨conjunction⟩}
\renewcommand{\crefmiddleconjunction}{⟨conjunction⟩}
\renewcommand{\creflastconjunction}{⟨conjunction⟩}
```

`\crefpairconjunction` is used when there are only two references in the list, `\creflastconjunction` is used between the penultimate and final reference in a list of more than two, and `\crefmiddleconjunction` is used between all the others. Again, they do not have to be conjunctions in the linguistic sense, and the same comments about surrounding space apply as in the case of `\crefrangeconjunction`. For example, the default definition of `\crefmiddleconjunction` is:

```
\renewcommand{\crefmiddleconjunction}{, }
```

`\crefpairgroupconjunction`
`\crefmiddlegroupconjunction`
`\creflastgroupconjunction`

By default, the conjunctions used to separate sub-lists of different reference types in a multi-reference are identical to those used to separate references of the same type[4]. You can override this by redefining `\crefpairgroupconjunction`, `\crefmiddlegroupconjunction` and `\creflastgroupconjunction`.

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

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

is typeset as

```
eqs. (1)\crefrangeconjunction(3)\crefmiddlegroupconjunction
theorems 1\crefpairconjunction2\crefmiddlegroupconjunction
fig. 1\creflastgroupconjunctionTheorem 3
```

[4]More accurately, redefining \crefpairconjunction etc. automatically redefines \crefpairgroupconjunction etc. so that they match. If you don't redefine anything, the default definition of \creflastgroupconjunction has an additional comma lacking in \creflastconjunction.

4.2 Customising Individual Reference Types

\creflabelformat
\crefname
\Crefname

The reference name for a given reference type is customised using the `\crefname` and `\Crefname` commands:

`\crefname{`⟨*type*⟩`}{`⟨*singular*⟩`}{`⟨*plural*⟩`}`
`\Crefname{`⟨*type*⟩`}{`⟨*singular*⟩`}{`⟨*plural*⟩`}`

used by the `\cref` and `\Cref` commands, respectively. You must supply both ⟨*singular*⟩ and ⟨*plural*⟩ forms of the name. If the corresponding `\Crefname` is undefined when `\crefname` is called, it will define `\Crefname` to be a capitalised version of `\crefname`, using `\MakeUppercase`. Conversely, if the corresponding `\crefname` is undefined when `\Crefname` is called, it will define `\Crefname` to be a lower-case version of `\Crefname`, using `\MakeLowercase`. Obviously, this will only work properly if the names begin with a letter. If the first letter is a special character construct, such as an accented character, you will need to surround it by braces. If the first thing in the name is *not* a letter at all (e.g. if it is a LaTeX command), you *must* define both capitalisation variants explicitly. Otherwise 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.). `ntheorem` provides extra information about the environment when different theorem-like environments use a common counter, which `cleveref` makes use of to distinguish between them automatically.

You may want the label format for a particular reference type to differ from the global format set by `\crefdefaultlabelformat` (see 4.1. You can do this using

`\creflabelformat{`⟨*type*⟩`}{`⟨*format*⟩`}`

The ⟨*type*⟩ argument is the reference type to customise, and the ⟨*format*⟩ argument defines the label format for references of that type. As in the case of `\crefdefaultlabelformat`, the latter should contain the three arguments `#1`, `#2` and `#3`, the first being the formatted version of the label counter, the others determining the beginning and end of the portion that becomes a hyper-link when the `hyperref` package is loaded (see section 7). `#2` and `#3` must appear in that order.

\crefrangelabelformat

Normally, the start and end references in a reference range are typeset using using the usual label format (as defined by `\crefdefaultlabelformat` or `\creflabelformat`) separated by `\crefrangeconjunction` (section 4.1). You can override this for a given reference type using

`\crefrangelabelformat{`⟨*type*⟩`}{`⟨*format*⟩`}`

The ⟨*format*⟩ argument should contain six arguments: `#1`, `#2`, `#3`, `#4`, `#5`, `#6`. The first two (`#1` and `#2`) are the formatted versions of the start and end label counters.

The next two (`#3` and `#4`) denote the beginning and end of the hyper-link for the first reference, the final two (`#5` and `#6`) the hyper-link for the second reference. The hyper-link arguments must appear in order. For example, the default format for equations is defined as:

```
\crefrangelabelformat{equation}{(#3#1#4)--(#5#2#6)}
```

4.3 Taking Full Control

If you need more precise control over the reference format than is possible by customising the individual components, then you can take full control of the format for any given type, overriding the component-derived format entirely. The formats for single references, reference ranges and multi-references are customised separately. If you only customise some of these, the other formats will be constructed from components as usual.

4.3.1 Single References

`\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*⟩`}`

The ⟨*type*⟩ is usually the name of the counter, except for theorem-like environments when `ntheorem` is loaded, in which case it is the environment name.

As in the case of the `\crefname` and `\Crefname` commands, 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`. This will only work if the format starts with a letter, and the same comments apply as in the case of `\crefname` (see section 4.1).

The ⟨*format*⟩ argument can be any valid LaTeX code, though you will need to `\protect` fragile commands. It 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 forms the hyper-link when the `hyperref` package is used, and must appear in that order (see section 7).

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-link.

Note that the hyper-link arguments are *not* letters, so if `#2` appears at the beginning of ⟨*format*⟩, `cleveref` will not be able to automatically define the other capitalisation variant automatically using `\MakeUppercase` or `\MakeLowercase`. In this case, you will have to define both variants separately. For example, if you wanted to the "Eq." to be part of the hyper-link, you would have to define:

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

4.3.2 Reference Ranges

`\crefrangeformat` `\Crefrangeformat`

The format for reference ranges is defined by `\crefrangeformat` and `\Crefrangeformat`. 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 as in the case of single references: the ⟨*type*⟩ is usually the name of the counter, except for theorem-like environments when `ntheorem` is loaded, and if the 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.

4.3.3 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`. Multi-references also require *all* the other reference formats to be defined (see sections 4.3.1 and 4.3.2), including the single reference range formats, even if you never use the `\crefrange` and `\Crefrange` commands.

The commands all take five arguments: the reference type, the format for the first reference in a list, the format for the second reference in a list of two, the

format for the middle references in a list of more than two, and the format for the last reference in a list of more than two.

```
\crefmultiformat{⟨type⟩}{⟨first⟩}{⟨second⟩}{⟨middle⟩}{⟨last⟩}
\Crefmultiformat{⟨type⟩}{⟨first⟩}{⟨second⟩}{⟨middle⟩}{⟨last⟩}
\crefrangemultiformat{⟨type⟩}{⟨first⟩}{⟨second⟩}{⟨middle⟩}{⟨last⟩}
\Crefrangemultiformat{⟨type⟩}{⟨first⟩}{⟨second⟩}{⟨middle⟩}{⟨last⟩}
```

The ⟨*type*⟩ is, as ever, the counter name, or environment name in the case of `ntheorem` theorem-like environments. The same considerations apply to the formatting arguments ⟨*first*⟩, ⟨*second*⟩, ⟨*middle*⟩ and ⟨*last*⟩ as for the ⟨*format*⟩ argument of `\crefformat` or `\crefrangeformat`, including the meaning of the arguments that should appear in the formatting code (`#1`, `#2` and `#3` for `\crefmultiformat` and `\Crefmultiformat`, `#1`–`#6` for `\crefmultiformat` and `\Crefmultiformat`). However, when the corresponding other-capitalisation variant is automatically defined, only the first letter of the ⟨*first*⟩ argument is upper- or lower-cased; the other arguments are defined to be identical for both variants.

Be careful to get the spaces at the beginning and end of the formatting code correct: the ⟨*first*⟩ and `\meta{second}`, or ⟨*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, in the default equation format

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

the ⟨*middle*⟩ argument should *not* have a space at the beginning, whereas the ⟨*second*⟩ and ⟨*last*⟩ arguments *should* have a space.

5 Overriding the Reference Type

As described previously, a label's reference type is usually determined by its counter, or in the case of `ntheorem` theorem-like environments by the environment name. Occasionally, you may want to override the reference type for a particular label. You can do this by supplying the desired type as an optional argument to the `\label` command:

```
\label[⟨type⟩]{⟨label⟩}
```

One circumstance in which is useful is when you want to define a special reference format for certain labels of a given type. By specifying a type that doesn't already exist in the `\label`'s optional argument, you can then define the reference format for that new type in whatever way you like, without affecting other references of the same type. For example, if a particular equation contains multiple expressions and you want it to always be referred to in the plural, you could use:

```
\crefname{pluralequation}{eqs.}{eqs.}
...
\label[pluralequation]{eq1}
```

You can of course reuse this format for other plural equations, too.

If you need to do this frequently, it can become tedious specifying the label explicitly each time. An alternative is to use the `aliascnt` package. This lets you define one counter to be an alias for another, so that effectively the same counter has two names. Since `cleveref` determines the reference type from the counter name, the two counter aliases can have different reference formats whilst really being the same counter. You have to somehow arrange for the correct counter alias to be used depending on which reference format you want (probably by defining two variants of the environment in question). But the effort involved might be worth the convenience of not having to remember to pass an explicit optional argument to a large number of labels.

You can use this trick to get different reference formats for different theorem-like environments, *without* using the `ntheorem` package[5]. For example,

```
\usepackage{aliascnt}
\usepackage{cleveref}
\newaliascnt{lemma}{theorem}
\newtheorem{lemma}[lemma]{Lemma}
\aliascntresetthe{lemma}
\crefname{lemma}{lemma}{lemmas}
```

6 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 completely remove the `cleveref` dependency. The recipient won't even realise you used `cleveref`!

The `sed` 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

[5] Thanks to Anand Deopurkar for suggesting this neat trick.

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.)

7 Interaction with Other Packages

The `cleveref` package *must* be loaded *after* all other packages that don't specifically support it[6], i.e. the

```
\usepackage{cleveref}
```

line should always be the last `\usepackage` command in your document's preamble.

Since `cleveref` redefines many internal commands involved in LaTeX's referencing system, it can interact badly with other packages that do the same. `varioref`'s enhanced referencing features (the ones 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 packages' 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}
```

In fact, `cleveref` does this for you automatically if `ntheorem` was loaded with the `thref` option. Note that you can still use the other features of `varioref` and `ntheorem` whilst using `cleveref`, as long as `cleveref` is loaded *last*.

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

8 Known Bugs

In no particular order.

- `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

[6] At the time of writing, I'm not aware of any that do.

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/`).

- The `poorman` sed script always uses `\ref` rather than `\ref*`, even if the command it's replacing is a starred variant `\cref*`, `\crefrange*` etc. (This should be fixed in a future version.)
- `cleveref` provides no `babel` support. (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.
- The `poorman` sed script uses `\ref` when replacing `cleveref` cross-references, but this loses any custom `cleveref` hyper-link formatting you might have defined. The philosophy behind the `poorman` option is to replace `cleveref`'s enhanced cross-referencing with standard LaTeX cross-reference commands that are guaranteed to work with any LaTeX installation. Although it would be simple to fix this bug, it's not obvious how best to do it without compromising on the underlying philosophy.

9 Possible Future Improvements

In no particular order.

- Allow reference-range collapsing to be disabled entirely via a package option
- Add an option to automatically sort references in a multi-reference before typesetting them.
- Enhance the `poorman` option to allow a choice of script language rather than just sed. (E.g. awk, perl, ...?).

10 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 `\cref` commands which use 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 type 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` LaTeX package).

10.1 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}}
```

We redefine the `\label` command to allow it to take an optional argument that overrides the default reference type in `\@currentlabel`. We have to postpone this redefinition till the beginning of the document because other packages do, and we need to override their redefinitions.

```
\AtBeginDocument{%
  \let\old@label\label
  \def\label{\@ifnextchar[\label@optarg\old@label}%]
  \def\override@label@type[#1][#2][#3]#4\@nil#5{[#5][#2][#3]#4}
  \def\label@optarg[#1]{%
    \protected@edef\@currentlabel{%
      \expandafter\override@label@type\@currentlabel\@nil{#1}}%
    \old@label}%
```

The `amsmath` package redefines the `\label` command within equation environments, so if it is loaded we have to extend the behaviour to support the optional argument. With `amsmath`, the original `\label` command in stored in `\tx@label`,

and \label@in@display replaces \abel inside equations. \label@in@display just saves the label for later, and defining it is left until the end of the equation, when \ltx@label is finally called.

To allow \abel within equations to support an optional argument, we first store the original \label@in@display and the new \label macro we defined above (since \label will be clobbered inside equations). Then we redefine \abel@in@display so that it wraps all its arguments, including any optional argument, in an extra set of {}. These are stripped away again by \ltx@label before calling the \label macro we defined above (save in \cref@label.

```
\@ifpackageloaded{amsmath}{%
  \let\cref@label\label%
  \let\old@label@in@display\label@in@display%
  \def\label@in@display{%
    \@ifnextchar[\label@in@display@optarg\label@in@display@noarg}%]
  \def\label@in@display@noarg#1{\old@label@in@display{{#1}}}
  \def\label@in@display@optarg[#1]#2{\old@label@in@display{[#1]{#2}}}
  \def\ltx@label#1{\cref@label#1}%
}{}%
}
```

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@reflabel#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@reflabel}{#1}}%
 \def\ref{\expandafter\protect\csname ref \endcsname}%
}
```

10.2 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 other 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}}
```

A basic utility macro for appending tokens to a token register.

```
\def\append@toks#1#2{\toks0={#2}%
  \edef\act{\noexpand#1={\the#1\the\toks0}}%
  \act}%
```

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 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}
```

10.3 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%
```

```
\newif\if@secondgroup%
\newif\if@secondref%
\stack@init{\@refstack}%
\stack@push{#2}{\@refstack}%
\@firstgrouptrue%
\@secondgroupfalse%
\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%
    \@firstgroupfalse%
    \@secondgrouptrue%
  \else%
    \isstackfull{\@refstack}%
    \ifstackfull%
      \@setcref@middlegroupconjunction%
    \else%
      \if@secondgroup%
```

```
          \@setcref@pairgroupconjunction%
        \else%
          \@setcref@lastgroupconjunction%
        \fi%
      \fi%
      \@secondgroupfalse%
    \fi%
```

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.

```
    \@secondreftrue%
    \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%
        \if@secondref%
          \expandafter\@setcref\expandafter{\@beginref}{#1}{@second}%
        \else%
          \expandafter\@setcref\expandafter{\@beginref}{#1}{@last}%
        \fi%
```

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%
      \if@secondref%
        \def\@pos{@second}%
      \else%
        \def\@pos{@last}%
      \fi%
```

```
          \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%
        \@secondreffalse%
        \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 range 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@pairgroupconjunction{\crefpairgroupconjunction}
\def\@setcref@middlegroupconjunction{\crefmiddlegroupconjunction}
\def\@setcref@lastgroupconjunction{\creflastgroupconjunction}
```

10.4 Reference Format Customisation Commands

10.4.1 Format Component Commands

The reference formats are usually constructed out of components defined by the user-level `\crefname`, `\Crefname`, `\creflabel` and `\crefrangelabel` commands. They simply use the supplied arguments to define appropriately named macros containing the formatting components. If the corresponding `\Crefname` or `\crefname` variant is not already defined, `\crefname` and `\Crefname` define it to be a version with the first letter capitalised or lower-cased, respectively. `\@crefdefineallformats` is then called to define all the formats from the new components.

```
\newcommand{\crefdefaultlabelformat}[1]{%
  \def\cref@default@label##1##2##3{#1}}
\newcommand{\crefname}[3]{%
  \@crefname{cref}{#1}{#2}{#3}%
  \@crefdefineallformats{#1}}
\newcommand{\Crefname}[3]{%
  \@crefname{Cref}{#1}{#2}{#3}%
  \@crefdefineallformats{#1}}
\newcommand{\creflabelformat}[2]{%
  \@creflabelformat{#1}{#2}%
  \@crefdefineallformats{#1}}
\newcommand{\crefrangelabelformat}[2]{%
  \@crefrangelabelformat{#1}{#2}%
  \@crefdefineallformats{#1}}
\def\@creflabelformat#1#2{%
  \expandafter\def\csname cref@#1@label\endcsname##1##2##3{#2}}
\def\@crefrangelabelformat#1#2{%
  \expandafter\def\csname cref@#1@rangelabel\endcsname%
  ##1##2##3##4##5##6{#2}}
```

The `\@crefname` utility macro does the real work of defining format names, by defining an appropriately named command to contain the format component, and using the first argument ("cref" of "Cref") to determine how to define the corresponding command with the other capitalisation.

```
\def\@crefname#1#2#3#4{%
  \expandafter\def\csname #1@#2@name\endcsname{#3}%
  \expandafter\def\csname #1@#2@name@plural\endcsname{#4}%
```

The following `\@tmpa` macro makes use of the fact that the first character of #1 is "c" for lower-case and "C" for upper-case, in order to wrap the capitalisation-dependent parts in macros so that the rest of the code can be capitalisation-variant

agnostic.

```
  \def\@tmpa##1##2\@nil{%
    \if##1c%
      \def\@other{C##2}%
      \def\@changecase{\MakeUppercase}%
    \else%
      \def\@other{c##2}%
      \def\@changecase{\MakeLowercase}%
    \fi}%
  \@tmpa#1\@nil%
```

If the other capitalisation variant is not already defined...

```
  \@ifundefined{\@other @#2@name}{%
```

Define `\@tmpa` and `\tmpb` to be partial expansions (expanded just once) of the macros for the capitalisation variant we've just defined above. The `\@toska` token register just makes the code less verbose.

```
    \expandafter\expandafter\expandafter\def%
    \expandafter\expandafter\expandafter\@tmpa%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@name\endcsname}%
    \expandafter\expandafter\expandafter\def%
    \expandafter\expandafter\expandafter\@tmpb%
    \expandafter\expandafter\expandafter{%
      \csname#1@#2@name@plural\endcsname}%
```

Add the `\@changecase` command to the front of the definitions of `\@tmpa` and `\}tmpb`.

```
    \expandafter\expandafter\expandafter\def%
    \expandafter\expandafter\expandafter\@tmpa%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpa}%
    \expandafter\expandafter\expandafter\def%
    \expandafter\expandafter\expandafter\@tmpb%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpb}%
```

Define the other capitalisation variants to be the partial expansions (expanded just once) of `\@tmpa` and `\@tmpb`.

```
    \newtoks\@toksa%
    \@toksa={%
      \expandafter\def\csname\@other @#2@name\endcsname}%
    \expandafter\the\expandafter\@toksa\expandafter{\@tmpa}%
    \@toksa={%
      \expandafter\def\csname\@other @#2@name@plural\endcsname}%
    \expandafter\the\expandafter\@toksa\expandafter{\@tmpb}%
  }{}%
}
```

`\@crefconstructcomponents` utility macro puts the reference format components for the specified reference type into temporary macros, for use by later

macros. The ridiculous number of "#" characters ensure that enough are left when they come to be used later (pairs "##" are collapsed to a single "#" each time the code is expanded).

```
\def\@crefconstructcomponents#1{%
  \@ifundefined{cref@#1@label}{%
    \let\@tmplabel\cref@default@label%
  }{%
    \expandafter\let\expandafter\@tmplabel%
    \csname cref@#1@label\endcsname%
  }%
  \@ifundefined{cref@#1@rangelabel}{%
    \expandafter\def\expandafter\@tmpa\expandafter{%
      \@tmplabel{####1}{####3}{#####4}}%
    \expandafter\def\expandafter\@tmpb\expandafter{%
      \@tmplabel{####2}{####5}{#####6}}%
    \@toksa={\def\@tmprangelabel##1##2##3##4##5##6}%
    \expandafter\expandafter\expandafter\expandafter%
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\expandafter%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter\expandafter%
    \expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter\@tmpa%
      \expandafter\crefrangeconjunction\@tmpb}%
  }{%
    \expandafter\let\expandafter\@tmprangelabel%
    \csname cref@#1@rangelabel\endcsname%
  }%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpname%
  \expandafter\expandafter\expandafter{%
    \csname cref@#1@name\endcsname}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpName%
  \expandafter\expandafter\expandafter{%
    \csname Cref@#1@name\endcsname}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpnameplural%
  \expandafter\expandafter\expandafter{%
    \csname cref@#1@name@plural\endcsname}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpNameplural%
  \expandafter\expandafter\expandafter{%
    \csname Cref@#1@name@plural\endcsname}%
  \expandafter\def\expandafter\@tmplabel\expandafter{%
    \@tmplabel{########1}{########2}{########3}}%
  \expandafter\def\expandafter\@tmprangelabel\expandafter{%
    \@tmprangelabel{########1}{########2}{########3}%
    {########4}{########5}{########6}}%
```

```
}
```

The `\@crefdefineformat` et al. macros construct calls to `\crefformat` et al. for the supplied reference type that define the corresponding formats in terms of the format components. This is mostly just an arduous exercise in controlling macro expansion order.

```
\def\@crefdefineformat#1{%
  \newtoks\@toksa%
  \@crefconstructcomponents{#1}% puts format components into tmp macros
```

Assemble the arguments for `\crefformat` and `\Crefformat` from the components.

```
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpfirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpname\expandafter~\@tmplabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpFirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpName\expandafter~\@tmplabel}%
```

Define `\crefformat` and `\Crefformat`.

```
  \@toksa={\crefformat{#1}}%
  \expandafter\the\expandafter\@toksa\expandafter{\@tmpfirst}%
  \@toksa={\Crefformat{#1}}%
  \expandafter\the\expandafter\@toksa\expandafter{\@tmpFirst}%
}
%
\def\@crefrangedefineformat#1{%
  \newtoks\@toksa%
  \newtoks\@toksb%
  \@crefconstructcomponents{#1}% puts format components into tmp macros
```

Assemble the arguments for `\crefrangeformat` and `\Crefrangeformat` from the components.

```
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpfirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpnameplural\expandafter~\@tmprangelabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpFirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpNameplural\expandafter~\@tmprangelabel}%
```

Define `\crefrangeformat` and `\Crefrangeformat`.

```
  \@toksa={\crefrangeformat{#1}}%
  \expandafter\the\expandafter\@toksa\expandafter{\@tmpfirst}%
  \@toksa={\Crefrangeformat{#1}}%
  \expandafter\the\expandafter\@toksa\expandafter{\@tmpFirst}%
}
%
\def\@crefdefinemultiformat#1{%
```

```
\newtoks\@toksa%
\newtoks\@toksb%
\@crefconstructcomponents{#1}% puts format components into tmp macros
```

Assemble the arguments for `\crefmultiformat` and `\Crefmultiformat` from the components.

```
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\@tmpfirst%
\expandafter\expandafter\expandafter{%
  \expandafter\@tmpnameplural\expandafter~\@tmplabel}%
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\@tmpFirst%
\expandafter\expandafter\expandafter{%
  \expandafter\@tmpNameplural\expandafter~\@tmplabel}%
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\@tmpsecond%
\expandafter\expandafter\expandafter{%
  \expandafter\crefpairconjunction\@tmplabel}%
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\@tmpmiddle%
\expandafter\expandafter\expandafter{%
  \expandafter\crefmiddleconjunction\@tmplabel}%
\expandafter\expandafter\expandafter\def%
\expandafter\expandafter\expandafter\@tmplast%
\expandafter\expandafter\expandafter{%
  \expandafter\creflastconjunction\@tmplabel}%
```

Bundle all four arguments for `\crefmultiformat` in token register `\@toksb`, then call it.

```
\@toksb={}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpfirst}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpsecond}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpmiddle}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmplast}}%
\@toksa={\crefmultiformat{#1}}%
\expandafter\the\expandafter\@toksa\the\@toksb%
```

Bundle all four arguments for `\Crefmultiformat` in token register `\@toksb`, then call it.

```
\@toksb={}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpFirst}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpsecond}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
  \expandafter{\@tmpmiddle}}%
\expandafter\append@toks\expandafter\@toksb\expandafter{%
```

```
    \expandafter{\@tmplast}}%
  \@toksa={\Crefmultiformat{#1}}%
  \expandafter\the\expandafter\@toksa\the\@toksb%
}
%
\def\@crefrangedefinemultiformat#1{%
  \newtoks\@toksa%
  \newtoks\@toksb%
  \@crefconstructcomponents{#1}% puts format components into tmp macros
```

Assemble the arguments that need to be passed to `\crefrangemultiformat` and `\Crefrangemultiformat` from the reference components.

```
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpfirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpnameplural\expandafter~\@tmprangelabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpFirst%
  \expandafter\expandafter\expandafter{%
    \expandafter\@tmpNameplural\expandafter~\@tmprangelabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpsecond%
  \expandafter\expandafter\expandafter{%
    \expandafter\crefpairconjunction\@tmprangelabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmpmiddle%
  \expandafter\expandafter\expandafter{%
    \expandafter\crefmiddleconjunction\@tmprangelabel}%
  \expandafter\expandafter\expandafter\def%
  \expandafter\expandafter\expandafter\@tmplast%
  \expandafter\expandafter\expandafter{%
    \expandafter\creflastconjunction\@tmprangelabel}%
```

Bundle all four arguments for `\crefrangemultiformat` in token register `\@toksb`, then call it.

```
  \@toksb={}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmpfirst}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmpsecond}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmpmiddle}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmplast}}%
  \@toksa={\crefrangemultiformat{#1}}%
  \expandafter\the\expandafter\@toksa\the\@toksb%
```

Bundle all four arguments for `\Crefrangemultiformat` in token register `\@toksb`, then call it.

```
  \@toksb={}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
```

```
    \expandafter{\@tmpFirst}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmpsecond}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmpmiddle}}%
  \expandafter\append@toks\expandafter\@toksb\expandafter{%
    \expandafter{\@tmplast}}%
  \@toksa={\Crefrangemultiformat{#1}}%
  \expandafter\the\expandafter\@toksa\the\@toksb%
}
```

`\@crefdefineallformats` calls each of the above, to define all formats for the given type from the corresponding components.

```
\def\@crefdefineallformats#1{%
  \@crefdefineformat{#1}%
  \@crefrangedefineformat{#1}%
  \@crefdefinemultiformat{#1}%
  \@crefrangedefinemultiformat{#1}}
```

10.4.2 Format Definition Commands

`\crefformat` et al. are lower-level user commands that give complete control over the format of different reference types. They override the component-based formats, simply using the supplied arguments to define appropriately named formatting macros, 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}[5]{%
  \@crefmultiformat{cref}{#1}{#2}{#3}{#4}{#5}}
\newcommand{\Crefmultiformat}[5]{%
  \@crefmultiformat{Cref}{#1}{#2}{#3}{#4}{#5}}
\newcommand{\crefrangemultiformat}[5]{%
  \@crefrangemultiformat{crefrange}{#1}{#2}{#3}{#4}{#5}}
\newcommand{\Crefrangemultiformat}[5]{%
  \@crefrangemultiformat{Crefrange}{#1}{#2}{#3}{#4}{#5}}
```

The utility macros do the real work, by using the first argument ("cref" or "Cref", and "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}%
```

The following `\@tmpa` macro makes use of the fact that the first character of #1 is "c" for lower-case and "C" for upper-case, in order to wrap the capitalisation-

dependent parts in macros so that the rest of the code can be capitalisation-variant agnostic.

```
  \def\@tmpa##1##2\@nil{%
    \if##1c%
      \def\@other{C##2}%
      \def\@changecase{\MakeUppercase}%
    \else%
      \def\@other{c##2}%
      \def\@changecase{\MakeLowercase}%
    \fi}%
  \@tmpa#1\@nil%
```

If the other capitalisation variant is not already defined...

```
  \@ifundefined{\@other @#2@format}{%
```

Define `\@tmpa` to be a partial expansion (expanded just once) of the capitalisation variant we've just defined above. The `\@toska` token register just makes the code less verbose.

```
    \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}}%
```

Add the `\@changecase` command to the front of the definition of `\@tmpa`.

```
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpa{##1}{##2}{##3}}%
```

Define the other capitalisation variant to be the partial expansion (expanded just once) of `\@tmpa`.

```
    \@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}%
```

The following `\@tmpa` macro makes use of the fact that the first character of #1 is "c" for lower-case and "C" for upper-case, in order to wrap the capitalisation-dependent parts in macros so that the rest of the code can be capitalisation-variant agnostic.

```
  \def\@tmpa##1##2\@nil{%
    \if##1c%
      \def\@other{C##2}%
```

```
      \def\@changecase{\MakeUppercase}%
    \else%
      \def\@other{c##2}%
      \def\@changecase{\MakeLowercase}%
    \fi}%
  \@tmpa#1\@nil%
```

If the other capitalisation variant is not already defined...

```
  \@ifundefined{\@other @#2@format}{%
```

Define `\@tmpa` to be a partial expansion (expanded just once) of the capitalisation variant we've just defined above. The `\@toska` token register just makes the code less verbose.

```
    \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}}%
```

Add the `\@changecase` command to the front of the definition of `\@tmpa`.

```
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}%
```

Define the other capitalisation variant to be the partial expansion (expanded just once) of `\@tmpa`.

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

The following `\@tmpa` macro makes use of the fact that the first character of #1 is "c" for lower-case and "C" for upper-case, in order to wrap the capitalisation-dependent parts in macros so that the rest of the code can be capitalisation-variant agnostic.

```
  \def\@tmpa##1##2\@nil{%
    \if##1c%
      \def\@other{C##2}%
      \def\@changecase{\MakeUppercase}%
    \else%
      \def\@other{c##2}%
```

```
      \def\@changecase{\MakeLowercase}%
    \fi}%
  \@tmpa#1\@nil%
```

If the other capitalisation variant of the first part of the multi-format definition is not already defined...

```
  \@ifundefined{\@other @#2@format@first}{%
```

Define `\@tmpa` to be a partial expansion (expanded just once) of the capitalisation variant we've just defined above. The `\@toska` token register just makes the code less verbose.

```
    \@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}}%
```

Add the `\@changecase` command to the front of the definition of `\@tmpa`.

```
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpa{##1}{##2}{##3}}%
```

Define the other capitalisation variant to be the partial expansion (expanded just once) of `\@tmpa`.

```
    \@toksa={%
      \expandafter\def\csname\@other @#2@format@first\endcsname%
        ##1##2##3}%
    \expandafter\the\expandafter\@toksa\expandafter{%
      \@tmpa{##1}{##2}{##3}}%
  }{}%
```

The other parts of the multi-format definition are defined to be identical for both capitalisation variants.

```
  \@ifundefined{\@other @#2@format@second}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@second\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@second\endcsname%
  }{}%
  \@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#6{%
  \expandafter\def\csname #1@#2@format@first\endcsname%
    ##1##2##3##4##5##6{#3}%
  \expandafter\def\csname #1@#2@format@second\endcsname%
    ##1##2##3##4##5##6{#4}%
  \expandafter\def\csname #1@#2@format@middle\endcsname%
    ##1##2##3##4##5##6{#5}%
  \expandafter\def\csname #1@#2@format@last\endcsname%
    ##1##2##3##4##5##6{#6}%
```

The following `\@tmpa` macro makes use of the fact that the first character of #1 is "c" for lower-case and "C" for upper-case, in order to wrap the capitalisation-dependent parts in macros so that the rest of the code can be capitalisation-variant agnostic.

```
  \def\@tmpa##1##2\@nil{%
    \if##1c%
      \def\@other{C##2}%
      \def\@changecase{\MakeUppercase}%
    \else%
      \def\@other{c##2}%
      \def\@changecase{\MakeLowercase}%
    \fi}%
  \@tmpa#1\@nil%
```

If the other capitalisation variant of the first part of the multi-format definition is not already defined...

```
  \@ifundefined{\@other @#2@format@first}{%
```

Define `\@tmpa` to be a partial expansion (expanded just once) of the capitalisation variant we've just defined above. The `\@toska` token register just makes the code less verbose.

```
    \@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}}%
```

Add the `\@changecase` command to the front of the definition of `\@tmpa`.

```
    \expandafter\expandafter\expandafter\the%
    \expandafter\expandafter\expandafter\@toksa%
    \expandafter\expandafter\expandafter{%
      \expandafter\@changecase\@tmpa{##1}{##2}{##3}{##4}{##5}{##6}}%
```

Define the other capitalisation variant to be the partial expansion (expanded just once) of `\@tmpa`.

```
    \@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}}%
  }{}%
```

The other parts of the multi-format definition are defined to be identical for both capitalisation variants.

```
  \@ifundefined{\@other @#2@format@second}{%
    \@toksa={%
      \expandafter\let\csname\@other @#2@format@second\endcsname}%
    \expandafter\the\expandafter\@toksa%
      \csname #1@#2@format@second\endcsname%
  }{}%
  \@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%
  }{}%
}
```

10.5 Default Reference Formats

We define default reference format components appropriate for LaTeX documents written in English.

FIXME: add babel support

The following must be defined before the user has the chance to customise anything, otherwise the component commands will not work.

```
\newcommand{\crefrangeconjunction}{ to~}
\newcommand{\crefpairconjunction}{ and~}
\newcommand{\crefmiddleconjunction}{, }
\newcommand{\creflastconjunction}{ and~}
\crefdefaultlabelformat{#2#1#3}
```

Define default format components. We want don't want these to be defined during the document preamble in case the user defines them themselves (automatic definition of the other capitalisation variant would fail). Therefore, we postpone actual definition of the formats until the beginning of the document, and only define those that haven't already been defined.

FIXME: add babel support

```
\AtBeginDocument{%
```

If the group conjunctions haven't been defined, define them to be identical to the reference conjunctions.

```
\@ifundefined{crefpairgroupconjunction}{%
  \let\crefpairgroupconjunction\crefpairconjunction}{}%
\@ifundefined{crefmiddlegroupconjunction}{%
  \let\crefmiddlegroupconjunction\crefmiddleconjunction}{}%
```

If the last reference conjunction hasn't been modified from its default, define the last group conjunction to include an extra comma. However, if the user has modified the last reference conjunction but hasn't defined the last group conjunction, they will expect the last group conjunction to be identical to their last reference conjunction definition, so do that.

```
\@ifundefined{creflastgroupconjunction}{%
  \def\@tmpa{ and~}%
  \ifx\creflastconjunction\@tmpa%
    \def\creflastgroupconjunction{, and~}%
  \else%
    \let\creflastgroupconjunction\creflastconjunction%
  \fi}{}%
```

Define default names (and in the case of equations, also the label format). 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.

```
\@ifundefined{cref@equation@name}{%
  \@crefname{cref}{equation}{eq.}{eqs.}%
  \@crefname{Cref}{equation}{Equation}{Equations}}{}%
\@ifundefined{cref@equation@label}{%
  \@creflabelformat{equation}{\textup{(#2#1#3)}}%
  \@ifundefined{cref@equation@rangelabel}{%
    \@crefrangelabelformat{equation}{%
      \textup{(#3#1#4)}--\textup{(#5#2#6)}}}{}}{}%
 \@ifundefined{cref@chapter@name}{%
   \@crefname{cref}{chapter}{chapter}{chapters}%
   \@crefname{Cref}{chapter}{Chapter}{Chapters}}{}%
\@ifundefined{cref@section@name}{%
  \@crefname{cref}{section}{section}{sections}%
  \@crefname{Cref}{section}{Section}{Sections}}{}%
\@ifundefined{cref@subsection@name}{%
  \@crefname{cref}{subsection}{section}{sections}%
  \@crefname{Cref}{subsection}{Section}{Sections}}{}%
\@ifundefined{cref@subsubsection@name}{%
  \@crefname{cref}{subsubsection}{section}{sections}%
  \@crefname{Cref}{subsubsection}{Section}{Sections}}{}%
\@ifundefined{cref@subsubsubsection@name}{%
  \@crefname{cref}{subsubsubsection}{section}{sections}%
  \@crefname{Cref}{subsubsubsection}{Section}{Sections}}{}%
\@ifundefined{cref@figure@name}{%
  \@crefname{cref}{figure}{fig.}{figs.}%
  \@crefname{Cref}{figure}{Figure}{Figures}}{}%
\@ifundefined{cref@figure@rangelabel}{%
  \@crefrangelabelformat{figure}{#3#1#4--#5#2#6}}{}%
```

```
\@ifundefined{cref@table@name}{%
  \@crefname{cref}{table}{table}{tables}%
  \@crefname{Cref}{table}{Table}{Tables}}{}%
\@ifundefined{cref@theorem@name}{%
  \@crefname{cref}{theorem}{theorem}{theorems}%
  \@crefname{Cref}{theorem}{Theorem}{Theorems}}{}%
\@ifundefined{cref@enumi@name}{%
  \@crefname{cref}{enumi}{item}{items}%
  \@crefname{Cref}{enumi}{Item}{Items}}{}%
\@ifundefined{cref@enumii@name}{%
  \@crefname{cref}{enumii}{item}{items}%
  \@crefname{Cref}{enumii}{Item}{Items}}{}%
\@ifundefined{cref@enumiii@name}{%
  \@crefname{cref}{enumiii}{item}{items}%
  \@crefname{Cref}{enumiii}{Item}{Items}}{}%
\@ifundefined{cref@enumiv@name}{%
  \@crefname{cref}{enumiv}{item}{items}%
  \@crefname{Cref}{enumiv}{Item}{Items}}{}%
\@ifundefined{cref@enumv@name}{%
  \@crefname{cref}{enumv}{item}{items}%
  \@crefname{Cref}{enumv}{Item}{Items}}{}%
```

Define any undefined formats using the components.

```
\@ifundefined{cref@equation@format}{%
  \@crefdefineformat{equation}}{}%
\@ifundefined{crefrange@equation@format}{%
  \@crefrangedefineformat{equation}}{}%
\@ifundefined{cref@equation@format@first}{%
  \@crefdefinemultiformat{equation}}{}%
\@ifundefined{crefrange@equation@format@first}{%
  \@crefrangedefinemultiformat{equation}}{}%
\@ifundefined{cref@chapter@format}{%
  \@crefdefineformat{chapter}}{}%
\@ifundefined{crefrange@chapter@format}{%
  \@crefrangedefineformat{chapter}}{}%
\@ifundefined{cref@chapter@format@first}{%
  \@crefdefinemultiformat{chapter}}{}%
\@ifundefined{crefrange@chapter@format@first}{%
  \@crefrangedefinemultiformat{chapter}}{}%
\@ifundefined{cref@section@format}{%
  \@crefdefineformat{section}}{}%
\@ifundefined{crefrange@section@format}{%
  \@crefrangedefineformat{section}}{}%
\@ifundefined{cref@section@format@first}{%
  \@crefdefinemultiformat{section}}{}%
\@ifundefined{crefrange@section@format@first}{%
  \@crefrangedefinemultiformat{section}}{}%
\@ifundefined{cref@subsection@format}{%
  \@crefdefineformat{subsection}}{}%
\@ifundefined{crefrange@subsection@format}{%
```

```
    \@crefrangedefineformat{subsection}}{}%
  \@ifundefined{cref@subsection@format@first}{%
    \@crefdefinemultiformat{subsection}}{}%
  \@ifundefined{crefrange@subsection@format@first}{%
    \@crefrangedefinemultiformat{subsection}}{}%
  \@ifundefined{cref@subsubsection@format}{%
    \@crefdefineformat{subsubsection}}{}%
  \@ifundefined{crefrange@subsubsection@format}{%
    \@crefrangedefineformat{subsubsection}}{}%
  \@ifundefined{cref@subsubsection@format@first}{%
    \@crefdefinemultiformat{subsubsection}}{}%
  \@ifundefined{crefrange@subsubsection@format@first}{%
    \@crefrangedefinemultiformat{subsubsection}}{}%
  \@ifundefined{cref@subsubsubsection@format}{%
    \@crefdefineformat{subsubsubsection}}{}%
  \@ifundefined{crefrange@subsubsubsection@format}{%
    \@crefrangedefineformat{subsubsubsection}}{}%
  \@ifundefined{cref@subsubsubsection@format@first}{%
    \@crefdefinemultiformat{subsubsubsection}}{}%
  \@ifundefined{crefrange@subsubsubsection@format@first}{%
    \@crefrangedefinemultiformat{subsubsubsection}}{}%
  \@ifundefined{cref@figure@format}{%
    \@crefdefineformat{figure}}{}%
  \@ifundefined{crefrange@figure@format}{%
    \@crefrangedefineformat{figure}}{}%
  \@ifundefined{cref@figure@format@first}{%
    \@crefdefinemultiformat{figure}}{}%
  \@ifundefined{crefrange@figure@format@first}{%
    \@crefrangedefinemultiformat{figure}}{}%
  \@ifundefined{cref@theorem@format}{%
    \@crefdefineformat{theorem}}{}%
  \@ifundefined{crefrange@theorem@format}{%
    \@crefrangedefineformat{theorem}}{}%
  \@ifundefined{cref@theorem@format@first}{%
    \@crefdefinemultiformat{theorem}}{}%
  \@ifundefined{crefrange@theorem@format@first}{%
    \@crefrangedefinemultiformat{theorem}}{}%
  \@ifundefined{cref@enumi@format}{%
    \@crefdefineformat{enumi}}{}%
  \@ifundefined{crefrange@enumi@format}{%
    \@crefrangedefineformat{enumi}}{}%
  \@ifundefined{cref@enumi@format@first}{%
    \@crefdefinemultiformat{enumi}}{}%
  \@ifundefined{crefrange@enumi@format@first}{%
    \@crefrangedefinemultiformat{enumi}}{}%
  \@ifundefined{cref@enumii@format}{%
    \@crefdefineformat{enumii}}{}%
  \@ifundefined{crefrange@enumii@format}{%
    \@crefrangedefineformat{enumii}}{}%
  \@ifundefined{cref@enumii@format@first}{%
```

```
    \@crefdefinemultiformat{enumii}}{}%
  \@ifundefined{crefrange@enumii@format@first}{%
    \@crefrangedefinemultiformat{enumii}}{}%
  \@ifundefined{cref@enumiii@format}{%
    \@crefdefineformat{enumiii}}{}%
  \@ifundefined{crefrange@enumiii@format}{%
    \@crefrangedefineformat{enumiii}}{}%
  \@ifundefined{cref@enumiii@format@first}{%
    \@crefdefinemultiformat{enumiii}}{}%
  \@ifundefined{crefrange@enumiii@format@first}{%
    \@crefrangedefinemultiformat{enumiii}}{}%
  \@ifundefined{cref@enumiv@format}{%
    \@crefdefineformat{enumiv}}{}%
  \@ifundefined{crefrange@enumiv@format}{%
    \@crefrangedefineformat{enumiv}}{}%
  \@ifundefined{cref@enumiv@format@first}{%
    \@crefdefinemultiformat{enumiv}}{}%
  \@ifundefined{crefrange@enumiv@format@first}{%
    \@crefrangedefinemultiformat{enumiv}}{}%
  \@ifundefined{cref@enumiv@format}{%
    \@crefdefineformat{enumiv}}{}%
  \@ifundefined{crefrange@enumiv@format}{%
    \@crefrangedefineformat{enumiv}}{}%
  \@ifundefined{cref@enumiv@format@first}{%
    \@crefdefinemultiformat{enumiv}}{}%
  \@ifundefined{crefrange@enumiv@format@first}{%
    \@crefrangedefinemultiformat{enumiv}}{}%
}
```

10.6 `hyperref` Support

If the `hyperref` package is loaded, we add hyper-link support to `cleveref`. Since `hyperref` messes around with some of the same LaTeX internals as `cleveref`, we also have to override some of its redefinitions so that they work with `cleveref`.

```
\@ifpackageloaded{hyperref}{%
\PackageInfo{cleveref}{`hyperref' support loaded}
\@ifpackagewith{hyperref}{backref}{%
  \PackageError{cleveref}{`cleveref' is currently incompatible with
    `hyperref's `backref' option}{Remove the `backref' option from
    `hyperref' if you want to use `cleveref'}}{}
```

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

```
\def\cref@reflabel#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\@crefnostar{#1}{#2}\@crefstarredfalse}
\def\@crefrangenostar#1#2#3{\@setcrefrange{#2}{#3}{#1}{}}
\def\@crefrangestar#1#2#3{%
  \@crefstarredtrue\@crefrangenostar{#1}{#2}{#3}\@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}
}{}
```

10.7 `ntheorem` Support

If `ntheorem` is loaded, we need to modify its theorem referencing features so that they work with `cleveref`.

```
\@ifpackageloaded{ntheorem}{%
\PackageInfo{cleveref}{‘ntheorem’ support loaded}
\@ifpackagewith{ntheorem}{thref}{%
  \PackageWarning{cleveref}{‘cleveref’ supersedes ‘ntheorem’s ‘thref’
    option}%
  \renewcommand{\thref}{\cref}}{}
```

Newer versions of `ntheorem` require a call to `\theorem@prework` when typesetting theorems. If an older version of `ntheorem` if being used, we just `\let` it to `\relax` to make sure it's defined.

```
\@ifundefined{theorem@prework}{\let\theorem@prework\relax}{}
```

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}%  <<<<<
  \theorem@prework%
  \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.

```
\AtBeginDocument{%
 \@ifundefined{cref@lemma@name}{%
   \@crefname{cref}{lemma}{lemma}{lemmas}%
   \@crefname{Cref}{lemma}{Lemma}{Lemmas}}{}%
 \@ifundefined{cref@corollary@name}{%
   \@crefname{cref}{corollary}{corollary}{corollaries}%
   \@crefname{Cref}{corollary}{Corollary}{Corollaries}}{}%
 \@ifundefined{cref@proposition@name}{%
   \@crefname{cref}{proposition}{proposition}{propositions}%
   \@crefname{Cref}{proposition}{Proposition}{Proposition}}{}%
 \@ifundefined{cref@definition@name}{%
   \@crefname{cref}{definition}{definition}{definitions}%
   \@crefname{Cref}{definition}{Definition}{Definitions}}{}%
 \@ifundefined{cref@result@name}{%
   \@crefname{cref}{result}{result}{results}%
   \@crefname{cref}{result}{Result}{Results}}{}%
```

Define any undefined formats using the components.

```
 \@ifundefined{cref@lemma@format}{%
   \@crefdefineformat{lemma}}{}%
 \@ifundefined{crefrange@lemma@format}{%
   \@crefrangedefineformat{lemma}}{}%
 \@ifundefined{cref@lemma@format@first}{%
   \@crefdefinemultiformat{lemma}}{}%
 \@ifundefined{crefrange@lemma@format@first}{%
```

```
    \@crefrangedefinemultiformat{lemma}}{}%
  \@ifundefined{cref@corollary@format}{%
    \@crefdefineformat{corollary}}{}%
  \@ifundefined{crefrange@corollary@format}{%
    \@crefrangedefineformat{corollary}}{}%
  \@ifundefined{cref@corollary@format@first}{%
    \@crefdefinemultiformat{corollary}}{}%
  \@ifundefined{crefrange@corollary@format@first}{%
    \@crefrangedefinemultiformat{corollary}}{}%
  \@ifundefined{cref@definition@format}{%
    \@crefdefineformat{definition}}{}%
  \@ifundefined{crefrange@definition@format}{%
    \@crefrangedefineformat{definition}}{}%
  \@ifundefined{cref@definition@format@first}{%
    \@crefdefinemultiformat{definition}}{}%
  \@ifundefined{crefrange@definition@format@first}{%
    \@crefrangedefinemultiformat{definition}}{}%
  \@ifundefined{cref@result@format}{%
    \@crefdefineformat{result}}{}%
  \@ifundefined{crefrange@result@format}{%
    \@crefrangedefineformat{result}}{}%
  \@ifundefined{cref@result@format@first}{%
    \@crefdefinemultiformat{result}}{}%
  \@ifundefined{crefrange@result@format@first}{%
    \@crefrangedefinemultiformat{result}}{}%
}%
}{}
```

10.8 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

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

```
    \ifx\@tmpa\@eof%
      \@not@eoffalse%
    \else%
      \@not@eoftrue%
      \edef\@tmpa{\@tmpa}%
    \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/||[cC]refname<.*><.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
  \lowercase{\edef\@tmpa{s/||[cC]reflabelformat<.*><.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
  \lowercase{\edef\@tmpa{s/||[cC]refrangelabelformat<.*><.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
  \lowercase{\edef\@tmpa{s/||[cC]refdefaultlabelformat<.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
  \lowercase{\edef\@tmpa{%
      s/||renewcommand<||crefpairconjunction><.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
  \lowercase{\edef\@tmpa{%
      s/||renewcommand<||crefpairgroupconjunction><.*>//g}}%
  \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
    \@tmpa^^J}%
```

```
    \lowercase{\edef\@tmpa{%
        s/||renewcommand<||crefmiddleconjunction><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}%
    \lowercase{\edef\@tmpa{%
        s/||renewcommand<||crefmiddlegroupconjunction><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}%
    \lowercase{\edef\@tmpa{%
        s/||renewcommand<||creflastconjunction><.*>//g}}%
    \expandafter\g@addto@macro\expandafter\cref@text\expandafter{%
      \@tmpa^^J}%
    \lowercase{\edef\@tmpa{%
        s/||renewcommand<||creflastgroupconjunction><.*>//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^^J}%
  \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.

```
\@ifpackageloaded{hyperref}{%
  \def\@crefnostar#1#2{%
    \edef\cref@text{}%
    \@cref{#1}{#2}%
    \cref@writescript{%
      \expandafter\string\csname#1\endcsname\string{#2\string}}}
  \def\@crefrangenostar#1#2#3{%
    \edef\cref@text{}%
    \@setcrefrange{#2}{#3}{#1}{}%
    \cref@writescript{%
      \expandafter\string\csname#1range\endcsname%
      \string{#2\string}\string{#3\string}}}%
}{%
  \DeclareRobustCommand{\cref}[1]{%
    \edef\cref@text{}%
    \@cref{cref}{#1}%
    \cref@writescript{\string\cref\string{#1\string}}}
  \DeclareRobustCommand{\Cref}[1]{%
```

```
    \edef\cref@text{}%
    \@cref{Cref}{#1}%
    \cref@writescript{\string\Cref\string{#1\string}}}
  \DeclareRobustCommand{\crefrange}[2]{%
    \edef\cref@text{}%
    \@setcrefrange{#1}{#2}{cref}{}%
    \cref@writescript{%
      \string\crefrange\string{#1\string}\string{#2\string}}}
  \DeclareRobustCommand{\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` and `\@@setrangeref`, as well as the conjunction macros `\@setcref@middlegroupconjunction`, `\@setcref@lastgroupconjunction` and `\@setcref@pairgroupconjunction`, to append text they typeset to `\cref@text`, as well as actually doing the typesetting.

```
\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@pairgroupconjunction{%
  \crefpairgroupconjunction%
  \expandafter\g@addto@macro%
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{\crefpairgroupconjunction}}
\def\@setcref@middlegroupconjunction{%
  \crefmiddlegroupconjunction%
  \expandafter\g@addto@macro%
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{\crefmiddlegroupconjunction}}
\def\@setcref@lastgroupconjunction{%
  \creflastgroupconjunction%
```

```
  \expandafter\g@addto@macro%
    \expandafter{\expandafter\cref@text\expandafter}%
    \expandafter{\creflastgroupconjunction}}
} % end of poorman option
```

Process options.

```
\ProcessOptions\relax
```