The newclude LaTeX package
A new system for including files (Frankenstein's backbone)

Matt Swift <swift@alum.mit.edu>

Version: 2 Date: 1999/11/02
Documentation revision: 1999/11/02

Abstract

Newclude is a backwards-compatible reimplementation of the LaTeX system for including files. The princpal new features are: (1) the restriction that `\clearpage`s must surround an included file is removed, (2) the restriction that `\include`s cannot be nested is removed, and (3) the provision of hooks executed before and after the contents of an included file. Newclude accomplishes the first two by using a single `aux` file instead of many.

Still in development, but already useful in many situations, are new commands that include partial contents of independent LaTeX files, which can also be processed on their own (that is, files that contain `\documentclass`, `\begin{document}`, etc.). *Newclude* absorbs and supersedes the former package *includex*.

Contents

Part I
Discussion

1 Introduction

Let us call a file that might be included into another document with a command in the `\include` family a *part*. When a part is actually included during a particular processing run, let us call it an *included part*, and when it is not included, let us call it an *unincluded part*. Notice that an unincluded part is *not* the same as a file that was never a candidate for inclusion with a command in the `\include` family.

The *newclude* package adds these features to the standard LaTeX inclusion system:

1. Hooks `\AtBeginInclude` and `\AtEndInclude` are provided.
2. Optional arguments to `\include` and friends override current values of `\AtBeginInclude` and `\AtEndInclude`.
3. `\include*` is like `\include` but with arbitrary commands rather than `\clearpage`s surrounding the part.
4. `\include` and friends can be nested.
5. `\includeall` cancels the effect of `\includeonly`.
6. `\IfAllowed` ⟨*filename*⟩ is a new conditional that branches, depending on what has been declared in an `\includeonly`.
7. Commands `\includedoc` etc. include a part that can be processed independently. These features are in development.

Newclude accepts three mutually-exclusive package options, with `tag` the default when no package option is given.

Loading *newclude* with the `simple` option provides only features 1 and 2. If you don't use either of these new features, the standard LaTeX and *newclude* inclusion systems will behave identically except in some unusual and benign odd cases relating the the parsing of the new optional arguments to `\include`, which are discussed below in that command's documentation.

The options `tag` and `allocate` each implement all the above features with a different method. Each method introduces different discrepancies from standard LaTeX which are discussed below in sections 4.2 and 4.3. If I discover how to make one method never inferior to the other, I will remove the other option from the package.

2 Usage

`\include` `\AtBeginInclude` `\AtEndInclude`

`\include [`⟨*prehook*⟩`]{`⟨*filename*⟩`}[`⟨*posthook*⟩`]` behaves like standard LaTeX's `\include` except that it can be nested and the contents of the two hook arguments, when they are given, are inserted at the beginning and end of the part whenever it is included, overriding the current values of `\AtBeginInclude` and `\AtEndInclude`.

Warning: *Right square braces (`]`s) in the optional arguments must be surrounded by curly braces to avoid confusing the argument parser.*

Warning: *A left square brace (`[`) that immediately follows an* `\include` *command's mandatory ⟨filename⟩ argument (after optional whitespace) will be considered to delimit the beginning of the ⟨posthook⟩ argument. If you want an actual left brace character in this position, you must precede it with something that will terminate TEX's search for an optional argument, such as* `\relax`*, , or a paragraph division (explicit or implicit).*

The commands `\AtBeginInclude` ⟨*tokens*⟩ and `\AtEndInclude` ⟨*tokens*⟩ are analagous to standard LaTeX's commands `\AtBeginDocument` ⟨*tokens*⟩ and `\AtEndDocument` ⟨*tokens*⟩.

FIX: multiple instances concatenate?

FIX give name to what's held by atbegininclude so that an override can mention it

When the optional argument ⟨*prehook*⟩ is given to `\include`, its contents will be used instead of whatever has been specified with `\AtBeginInclude`, for that one inclusion. Likewise, ⟨*posthook*⟩ will be used in place of whatever has been specified with `\AtEndInclude` for that one inclusion.

For example, putting the `\chapter` declaration in the ⟨*prehook*⟩ argument allows the chapter name, and, optionally, a corresponding LaTeX label, to be kept in the including file, rather than the included file:

```
\include [\chapter{Whales}
          \label{ch:whales}]
         {big-cetecea}
```

The ⟨*posthook*⟩ argument can be used, for example, to delimit or undo declarations made in the ⟨*prehook*⟩ or the included file: FIX: better example, since these coudl simple appear before/after the `\include` without ill effect.

```
\include [\begingroup\larger] % this part in larger type
         {manifesto}
         [\endgroup]
```

`\include*` `\include*[`⟨*prehook*⟩`]{`⟨*filename*⟩`}[`⟨*posthook*⟩`]` is like `\include` but omits the usual `\clearpage`s that surround an included part, replacing them with `\IncludeSurround`, which defaults to `\DefaultIncludeSurround`. The contents of `\IncludeSurround` are inserted before the ⟨*prehook*⟩ or whatever has been specified with `\AtBeginInclude`, and after the ⟨*posthook*⟩ or whatever has been specified with `\AtEndInclude`.

`\IncludeSurround`

`\DefaultIncludeSurround`

Warning: *A space gets inserted after an* `\include*` *unless it is suppressed by a immediately following. Combined with trailing spaces in the included file, this may lead to unwanted spaces. For this reason,* `\DefaultIncludeSurround` *is initialized to* `\par`*. When the user must explicitly change* `\IncludeSurround` *to achieve totally smooth flow from main file to included file, they are more likely to consult this documentation if they spot a problem. Package and class writers should take this difficulty into account when changing* `\DefaultIncludeSurround`*.*

`\includeonly` The `\includeonly` command is reimplemented, but its usage and behavior is the same as the standard LaTeX version.

`\includeall` The `\includeall` command cancels the effect of any `\includeonly` command presently in effect.

If you write an `\includeonly` so that each file appears on its own line, it is particularly easy to add and remove files to include by commenting out their lines, but it becomes laborious to comment out the entire `\includeonly` command. It's easy, however, to uncomment a single `\includeall` command when you want to process the entire document. (Or `\includeall` could be inserted from the command line that invokes LaTeX, and so on.)

3 Experimental features

`\includeenv` `\includeenv [`⟨*prehook*⟩`]{`⟨*filename*⟩`}{`⟨*environment name*⟩`} {`⟨*instance*⟩`}[`⟨*posthook*⟩`]`
`\includeenv *[`⟨*prehook*⟩`]{`⟨*filename*⟩`}{`⟨*environment name*⟩`} {`⟨*instance*⟩`}[`⟨*posthook*⟩`]`

`\includeenv` includes the contents of a single LaTeX environment that appears in ⟨*filename*⟩. The environment is specified by giving its name (⟨*environment*⟩) and an instance of that environment in the file (⟨*instance*⟩). Presently, ⟨*instance*⟩ is ignored, so that it will always be the contents of the first occurrence in ⟨*filename*⟩ of a LaTeX environment with the name ⟨*environment*⟩ that will be included. In the future, the ⟨*instance*⟩ argument may be used to specify the *n*th instance of the environment within the file, or further specify the environment to be extracted in some other way.

FIX: right now they're required; skip text up to documentclass OR the target, then branch?

Good preamble syntactic sugar: `\let\TheMarkupDeclarations\begin`

To do: *You can insert a* `\usepackage` *into the main aux file and have it loaded properly. If we discover a* `\usepackage` *that is not a formatting package, one strategy is to insert a corresponding* `\usepackage` *into the (main) aux file and then bail after the preamble.*

To do: *You can't skip verbatim text via macro argument processing and sugar. this means that a major reimplementation of skipping using verbatim methods will have to be done.*

The included file is permitted (but not required) to have its own `\documentclass` command and `\begin{document}` . . . `\end{document}` pair. `\includeenv` extracts the specified environment by processing the preamble if one exists, skipping text up until the beginning of the specified environment, processing the contents of the environment, and skipping the rest of the included part.

Notice that while a `\begin{document}` . . . `\end{document}` pair may not technically delimit a LaTeX environment, you may nevertheless (because it looks exactly like an environment) set ⟨*environment*⟩ to `document` to extract the contents of the `document` "environment" of ⟨*filename*⟩.

Consider the following issues when you are tempted to use this command. Maybe the `\usepackage` you are about to disregard is necessary to processing the part's contents. Maybe it conflicts with a package already loaded at top level. Maybe both! The same holds of course for the defining commands like `\newcommand` that one expects to find in a package.

A deep problem with the design of a LaTeX source file exists with respect to the function of the preamble. The preamble contains declarations that determine how the document below will be formatted. Unfortunately, there is no way to make the distinction between:

1. declarations that signal that certain markup will appear in the document

that are either not defined in the LaTeX kernel or are used with a different syntax

2. declarations that describe how a certain instance of the document should be formatted

Examples in the first category are `\usepackage{url}` and `FIX%example`, and examples in the second are `\usepackage{times}` and `FIX`. When you want to include the document or a part of it in another document, it is absolutely necessary to make this distinction so that declarations in category (1) can be processed and declarations in category (2) can be ignored.

Adopting a convention on the use of the preamble can overcome this design problem, but it will not fix the problem for legacy files whose preambles do not obey the convention. Legacy files that contain category (1) declarations in their preambles must either be altered or specifically accommodated with additional commands.

The convention I suggest is to `\usepackage{preamble}`. `\beginmarkup` `\endmarkup`. FIX. Can we arrange to load

When `\includeenv` encounters a `\usepackage` command in the included part, it looks at the packages in the argument of `\usepackage` and issues a warning if the package is not already loaded and does not appear on a list of packages known whose use falls entirely within category (2). (See the `\DeclareFormattingPackage` command below.)

The `\documentclass` command is of course also a category (1) declaration. Presently, if `\includeenv` detects that the arguments to an included `\documentclass` command differ from the arguments of the `\documentclass` command of the including document, it will issue a warning, and continue. In the future, I hope to make this behavior smarter by having `\includeenv` take specific actions for specific combinations of arguments. For example, if the included document's class implies the use of markup not defined in the parent's class, an appropriate action would be to define the missing markup commands. A document of class *report* and a document of class *article*, on the other hand, do not (I don't think) declare different markup, so that there should be no warning in this case.

`\includeenv*` `\includeenv*` is analogous to `\include*`, that is, it surrounds the included part with `\IncludeSurround` rather than `\clearpage`.

`\includedoc` `\includedoc[`⟨*prehook*⟩`]{`⟨*file name*⟩`}[`⟨*posthook*⟩`]` is shorthand for `\includeenv` `\includedoc*` `[`⟨*prehook*⟩`]{`⟨*filename*⟩`}{`⟨***document***⟩`}{`⟨⟩`}[`⟨*posthook*⟩`]`.

`\includedoc*` is analogous to `\includeenv*`.

4 Options

4.1 Simple

If the `simple` option is given, the only new feature provided is the hooks (features 1 and 2 above). As with standard LaTeX, `\clearpage`s surround an `\include` and nesting `\include`s gives an error. *Newclude* will only behave differently than standard LaTeX command scans for possible optional arguments will make a different.

4.2 Tag

The `tag` option causes LaTeX to use just one `aux` file. This option, which is the default, works well. I am aware of the following two differences from the kernel's including system:

1. If the LaTeX process is stopped during the processing of a part, all information normally stored in an `aux` file from that point in the document forward is lost. In the kernel's system, processing the document twice more would recover any `aux` information previously generated for parts.

   If LaTeX is always invoked in `\nonstopmode` (e.g., by `AUC-TeX`), then this difference is only going to occur when there are catastrophic errors that cause even `\nonstopmode` to terminate processing.

2. Other packages and classes that redefine kernel commands that write to `\@auxout` will cause problems.

The first difference must be accepted. The second difference can be removed on a case by case basis, by specifically coding compatibility with such packages and classes. I intend to do this. Here is a list of such packages and classes known to me:

⟨*none so far*⟩ If you discover any more for this list, please write me!

It's also very easy to revise the other package to be compatible with *newclude* as it is now. See section 6 below, which includes a list of relevant kernel commands.

4.3 Allocate

The second way (the `allocate` option) represents my first attempt at a solution, and until I am sure it has no advantages over `tag` under any circumstances, it will continue to be an option.

The `allocate` option causes LaTeX to dynamically allocate TeX output streams to each part as they are needed. Streams are allocated when processing of the part begins, and are reclaimed after the ejection of the last page to which the part has contributed. Like the old system, a separate `aux` file is created for each part. The limitation of this implementation is that TeX only possesses 16 output streams. Each of the commands `\tableofcontents`, `\listoffigures`, `\listoftables`, `\makeglossary`, and `\makeindex` causes LaTeX to use one output stream. The remainder (minus any streams required by packages and classes) are available for the including system. If n streams are available, the level of nesting possible is $n - 1$ minus the maximum number of parts that occur on the same page. For example, if 10 streams are available and the parts never appear on the same page (the old behavior required by the `\clearpage`s), then 8 levels of nesting are possible (which is 8, not 7 more than with the old system). The maximum number of parts that may contribute the to same page is calculated with the same equation. Note: TeX's page-breaking algorithm looks ahead until it has more than enough material to fill one page. You must count all the new `aux` files that are opened during a look-ahead as contributing to the page in question, even if some of the later ones do not actually contribute to the page after the break is chosen.

The `allocate` solution is itself implemented in two ways. The system either reserves a fixed number of output streams from the start, or will dynamically claim and free them as needed. The dynamic solution is the default. I do not see much

use for the static solution at present. If the dynamic system claims streams that are later required, then it is simply a question of whether *newclude* or the other feature is going to signal an error about having no more streams to allocate.

5 Programmers' interface

\IfAllowed `\IfAllowed {`⟨*part name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` executes ⟨*true*⟩ if ⟨*part-name*⟩ is on the list of files to be included and ⟨*false*⟩ otherwise. If there is no list, executes ⟨*true*⟩.

\IncludeName `\IncludeName` expands to the name of the part currently being processed. In the toplevel source file, it will expand to `\jobname`.

\ParentName `\ParentName` expands to the name of the part that includes the part currently being processed. In the toplevel source file, expanding `\ParentName` will generate a warning and expand to `\jobname` (which is also what `\IncludeName` expands to).

FIX: root source file? toplevel? master? principle source? glossary!

\DeclareFormattingPackage `\DeclareFormattingPackage {`⟨*package name*⟩`}` declares ⟨*package name*⟩ to be a package that only makes formatting declarations, that is, the effect of using it falls entirely within category (2). If a formatting package occurs in a `\usepackage` declaration in the preamble of a part included by `\includeenv`, no warning will be given. An example of a formatting package is the *times* package. No facility is provided to distinguish the case when a package is used with or without certain package options, so do not declare a package as a formatting package unless it is so regardless of the options it is passed.

If you send me the names of formatting packages, I will include them in the next release of *newclude*. Meanwhile, you may declare them in `newclude.cfg`. Do the same for your local formatting packages if you wish. It does no harm to declare a package as a formatting package more than once.

\ifSkipPreamble
\SkipPreambletrue
\SkipPreamblefalse
\Disable
\DisableAll

`\Disable {`⟨*tokens*⟩`}` provides a way to ignore additional commands when using `\includeenv` and friends. If you want to cause the macro `\foo` which takes no arguments to be entirely ignored in parts, issue the command `\Disable{\let\foo\relax}` any time before including the parts you want to affect. If `\foo` takes one mandatory argument, write `\let\foo\Gobble` instead. If `\foo` takes one optional and one mandatory, write `\let\foo\GobbleOM`. And so on. For other examples, see the gobbling commands in the *moredefs* package (which *newclude* requires), or write your own.

The arguments to `\Disable` are accumulated and executed by the command `\DisableAll`, which is executed inside a group that contains a part when it is included.

There is no way to undo the effect of issuing a `\Disable` command.

6 How to play nicely with *newclude*

To adapt a package or class for use with the `tag` option of *newclude*:

1. replace `\immediate\write\@auxout` with `\@writeaux`
2. replace `\protected@write\@auxout` with `\protected@writeaux`

3. add

```
\providecommand\@writeaux {%
  \immediate\write\@auxout
}
\providecommand\protected@writeaux {%
  \protected@write\@auxout
}
```

Part II
Implementation

7 Version control

\fileinfo
\DoXUsepackagE
\HaveECitationS
\fileversion
\filedate
\docdate
\PPOptArg

These definitions must be the first ones in the file.

```
\def\fileinfo{A new system for including files (Frankenstein's backbone)}
\def\DoXPackageS {}
\def\fileversion{v2}
\def\filedate{1999/11/02}
\def\docdate{1999/11/02}
\edef\PPOptArg {%
  \filedate\space \fileversion\space \fileinfo
}
```

If we're loading this file from a `\ProcessDTXFile` command (see the *compsci* package), then `\JusTLoaDInformatioN` will be defined; othewise we assume it is not (that's why the FunkY NamE).

If we're loading from `\ProcessDTXFile`, we want to load the packages listed in `\DoXPackageS` (needed to typeset the documentation for this file) and then bail out. Otherwise, we're using this file in a normal way as a package, so do nothing. `\DoXPackageS`, if there are any, are declared in the `dtx` file, and, if you're reading the typeset documentation of this package, would appear just above. (It's OK to call `\usepackage` with an empty argument or `\relax`, by the way.)

```
\makeatletter% A special comment to help create bst files.  Don't change!
\@ifundefined{JusTLoaDInformatioN} {%
  }{% ELSE (we know the compsci package is already loaded, too)
  \UndefineCS\JusTLoaDInformatioN
  \SaveDoXVarS
  \eExpand\csname DoXPackageS\endcsname\In {%use \csname in case it's undefined
    \usepackage{#1}%
  }%
  \RestoreDoXVarS
  \makeatother
  \endinput
}% A special comment to help create bst files.  Don't change!
```

Now we check for LaTeX2e and declare the LaTeX package.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{newclude}[\PPOptArg]
```

8 Review of the kernel's inclusion system

One `aux` file is written to disk for the *principle source* and one for each of the included *parts*. The reason to have a separate ones for the parts is so that information from the last time the part was included is retained in subsequent runs even when the part is excluded by `\includeonly`. Suppose a part is processed once, and on a subsequent run its name is removed from the `\includeonly` list. This run will still read in the part's `aux` file, since the `aux` file of any part that was `\included` during the last run is always read. But the information therein is

not going to be regenerated in this run, since the part will not be processed. The main `aux` file is created anew with each run, so this information would be lost if it resided there.

To handle writing these multiple `aux` files, the kernel uses two of TEX's output streams. When a routine writes to an auxiliary file, it writes to `\@auxout`, which is `\let` to either `\@mainaux`, the `aux` file for the principle source, or `\@partaux` the `aux` file for all the parts each in turn.

When encountering an `\include` command, but before deciding whether or not to actually load the part, the kernel writes a command to `\@mainaux` that will load the part's `aux` file. The main `aux` file is loaded by `\document`, so that *all* `aux` files are read in every time the principle source is processed.

If a part is actually loaded, a *checkpoint* is written to the part's `aux` file consisting of a snapshot of the counters (a record of the values of all LATEX counters). On the next run, if the part is not actually loaded, the information in its `aux` file has nevertheless already been processed by `\document`. Processing the checkpoint causes a macro to be defined that when invoked restores the counter state. When `\include` does not actually load a part it calls this checkpoint macro instead to alter the present counter state.

This system has pitfalls as well as benefits. It is useful to keep the bibliography, citations, cross references, and page numbers up to date in certain situations, but the results can be confusing sometimes, because checkpoints are not documented. (Perhaps this is remedied in the 2d edition of the LATEX manual.) How, besides reading the code, or finding out the hard way, is anyone supposed to guess that rearranging two "deactivated" `\include` statements in a principle source will bring havoc on the page numbers?

9 Discussion of *newclude*'s inclusion system

The simple removal of the `\clearpage`s that surround an included part would cause a problem involving the delayed action of `\write` commands. Suppose a part ending with a `\write` command ends halfway down a page, and another `\write` occurs in the principle source immediately (or soon) after the inclusion. The first must be written to `\@partaux` and the second to `\@mainaux`. If we close `\@partaux` while the first `\write` is still pending, that is, before the current page has been shipped out, then the `\write` will be destined for a closed stream and therefore go to the log file and terminal. The `\clearpage`s solve this by flushing all pending `\write`s. Then we can close `\@partaux` immediately and reopen `\@mainaux`.

Successful removal of the `\clearpage`s can be accomplished either by having the entire document use just one auxiliary file, or by allocating additional output streams so that it becomes possible to avoid closing `\@partaux` until after the current page is shipped out when all the `\write`'s to it have been completed.

10 Package initialization

```
23 \RequirePackage{moredefs}
24 \InitCS\sc@t@a
25 \DeclareOption{simple} {%
```

```
  \input{simple.sto}
  \let\sc@t@a\endinput
}
%^^A\DeclareOption{group} {%
%^^A  \AtEndOfPackage {\input{group.sto}}
%^^A}
\DeclareOption{tag} {%
  \AtEndOfPackage {\input{tag.sto}}
}
\DeclareOption{allocate} {%
  \AtEndOfPackage {\input{allocate.sto}}
}
\DeclareBooleanOptions{dynamic}{static}
\ExecuteOptions{tag}
\ProcessOptions
```

If the `simple` option has been given, end right here.

```
\sc@t@a
```

11 Simple

The above option processing causes the file `simple.sto` to be loaded when the `simple` is given. After it is loaded, processing stops. When the `simple` option is not given, *newclude* package code continues in section 12.

The `simple` option adds the optional argument to `\include`, and does nothing else.

`\include` I'm not really sure why the `\relax` is there; I'm imitating the kernel's command.

```
\defcommand\include {%
  \relax
  \ifnum\@auxout=\@partaux
    \@latex@error{\string\include\space cannot be nested}\@eha
  \else
    \expandafter\@include
  \fi
}
```

`\@include`

```
\defcommand\@include [2][] {%
  \clearpage
  \if@filesw
    \immediate\write\@mainaux{\string\@input{#2.aux}}%
  \fi
  \@tempswatrue
  \if@partsw
    \@tempswafalse
    \edef\reserved@b{#2}%
    \@for\reserved@a:=\@partlist\do
      {\ifx\reserved@a\reserved@b\@tempswatrue\fi}%
  \fi
  \if@tempswa
    \let\@auxout\@partaux
    \if@filesw
      \immediate\openout\@partaux #2.aux
```

```
      \immediate\write\@partaux{\relax}%
    \fi
```

All we did was change `#1` to `#2` and add the next line.

```
    #1%
    \@input@{#2.tex}%
    \clearpage
    \@writeckpt{#2}%
    \if@filesw
      \immediate\closeout\@partaux
    \fi
  \else
    \@nameuse{cp@#2}%
  \fi
  \let\@auxout\@mainaux
}
```

12 Common

The code in this section is common to the `tag` and `allocate` options.

`\nc@t@a` `\nc@t@b` `\nc@t@c` `\nc@toks@a` Scratch variables.

```
\ReserveCS\nc@t@a
\ReserveCS\nc@t@b
\ReserveCS\nc@t@c
\newtokens\nc@toks@a
```

`\IncludeSurround` `\DefaultIncludeSurround`

```
\newcommand\DefaultIncludeSurround {%
  \par
}
\newlet\IncludeSurround\DefaultIncludeSurround
```

`\c@IncludeDepth` With nested `\include`s, we need some way for the various ones to distinguish themselves, so we keep track of the nested depth with the `IncludeDepth` counter.

```
\newcounter{IncludeDepth} % starts at 0
```

`\IfAllowed` `\includeonly` `\includeall` I think it's more efficient to define a macro for each included part on the list than it is to search through the list possibly twice for each one. Other opinions on making this whole thing more efficient?

We are using the usual LaTeX trick of undefined control sequences comparing equally with `\relax`. Empty control sequences are *not* the same. Should be followed by ⟨*true clause*⟩ then ⟨*false clause*⟩.

```
\newcommand\IfAllowed [1] {%
  \@firstoftwo
}
\newcommand\includeall {%
  \let\includeonly\Gobble
}
\defcommand\includeonly [1] {%
  \@partswtrue
%  \DTypeout{INCLUDEONLY}%
```

```
  \edef\@partlist {\zap@space#1 \@empty}%
  \@for\nc@t@a:=\@partlist \do {%
    \InitName*{nc@part@\nc@t@a}%
  }%
  \defcommand\IfAllowed [1] {% args: part-name
    \@ifundefined{nc@part@##1} {%
%         \DTypeout{##1 NOTALLOWED}%
        \let\nc@t@c\@secondoftwo
      }{% ELSE
%       \DTypeout{##1 ALLOWED}%
      \let\nc@t@c\@firstoftwo
    }%
    \nc@t@c
  }%
%  \DTypeout{ENDINCLUDEONLY}%
}
```

`\include` `\include*` This is the principle user command. The scratch variable `\nc@t@b` contains what really surrounds the included file.

```
\def\include {%
  \@ifstar {%
      \let\nc@t@b\IncludeSurround
      \nc@include
    }{% ELSE
    \let\nc@t@b\clearpage
    \nc@include
  }%
}
```

13 Experimental common

`\Disable` `\DisableAll` This allows the disabling hacks.

```
\ReserveCS\DisableAll
\newcommand\Disable [1] {%
  \g@addto@macro\DisableAll{#1}%
}
```

We start with considering how to quit inputting a file. The idea is to make the `\end{document}` command of the part call `\endinput`. But there is a hitch that characters on the line after the `\end{document}` get inserted when you don't want them to. To beat that limitation, we have to do some work.

`\nc@radical@shutdown` We will add a bunch of commands to this macro, with the idea of `\catcode`ing everything and its brother to a comment. This would be a brute force method!

```
\ReserveCS\nc@radical@shutdown
```

First log a message that we're about to do some crazy things. In case something goes wrong, this might help.

```
\addto@macro\nc@radical@shutdown {%
  \MonsterInfo{newclude}
    {\protect\nc@radical@shutdown\space beginning}}
```

Now we start adding `\catcode` commands. These first two should be redundant; but just in case someone changed things. . . .

```
\addto@macro\nc@radical@shutdown{\catcode`\%=14}     % 14 = comment
\addto@macro\nc@radical@shutdown{\catcode`\^=7}      % 7 = superscript
```

`\nc@disable@char` Next, we define a command we weill use in a loop in a moment.

```
\newcommand\nc@disable@char[1] {%
  \addto@macro\nc@radical@shutdown
    {\catcode`#1=14}}  % 14 = comment
```

The following list contains every keyboard char except these three, which are treated specially: `%#`. The first is already a comment, and we handle the second in a moment. Each character in the following list is `\catcode`d to a comment:

```
\@tfor\sc@t@a:=abcdefghijklmnopqrstuvwxyz%
                ABCDEFGHIJKLMNOPQRSTUVWXYZ%
                ~!@$&*()_+-=[]|/?.,<>%
                1234567890%
                `'";:%
                \^\\\{\}\ % this is really the chars "^\{}" and space
    \do {\expandafter\nc@disable@char\sc@t@a}
```

We add `#` separately, because it's tricky or impossible to put it into the list we just used.

```
\nc@disable@char\#
```

We end the macro with `\endinput`. This has to come after all the previous, otherwise, TEX goes ahead and reads to the end of the line immediately, with regular catcodes. This is a good theory, I'm not sure it's necessary.

```
\addto@macro\nc@radical@shutdown{\endinput}
```

`c@radical@shutdown@aftergroup` We need to use `\nc@radical@shutdown` this way.

```
\newcommand\nc@radical@shutdown@aftergroup {%
  \aftergroup\nc@radical@shutdown
}
```

`\includedoc`
`\includedoc*`

```
\newcommand\includedoc {%
  \md@check@star
  \Expand \sc@star@nothing\In {%
    \IncludeEnv##1{document}{}%
  }%
}
```

`\includedocskip`
`\includedocskip*`

```
\newcommand\includedocskip {%
  \md@check@star
  \Expand \sc@star@nothing\In {%
    \IncludeEnvSkip##1{document}{}%
  }%
}
```

`\IncludeEnv`
`\nc@includeenv`
`\nc@@includeenv`

```
\newcommand\IncludeEnv [2] {% args: environment instance
```

```
  \md@check@star
  \@ifnextchar [ {%           ^^A for Emacs: ]
      \nc@includeenv{#1}{#2}%
    }{% ELSE
    \nc@includeenv{#1}{#2}[]%
  }%
}
\NewName{nc@includeenv} {#1#2[#3]} {% args: environment instance [prehook]
  \@ifnextchar [ {%      ^^A for Emacs: ]
      \nc@@includeenv {#1}{#2}{#3}%
    }{% ELSE
    \nc@@includeenv {#1}{#2}{#3}[]%
  }%
}
\NewName{nc@@includeenv} {#1#2#3[#4]} {% args: environment instance prehook [posthook]
  \begingroup
    \DisableAll
    \let\documentclass\GobbleOM
    \let\usepackage\GobbleOM
    \expandafter\def\csname end#1\endcsname {%
      \makeatletter
      % POSTHOOK
      \nc@radical@shutdown@aftergroup
    }%
    \expandafter\def\csname #1\endcsname {} % PREHOOK
  \endgroup
  \par
  \Expand \sc@star@nothing\In {%
    \include##1{#2}%
  }%
}

\NewName {nc@@includeenvskip} {#1#2#3[#4]} {% args: environment instance prehook [posthook]
  \begingroup
    \DisableAll
    \expandafter\def\csname end#1\endcsname {%
      \makeatletter
      % POSTHOOK
      \nc@radical@shutdown@aftergroup
    }%
    \expandafter\def\csname #1\endcsname {} % PREHOOK
    \long\def\documentclass ##1\begin{document}{%
      \begingroup
      \def\@currenvir{document}%
    }
  \endgroup
  \par
  #1%
}
```

14 Tag

The code in this section is processed when the `tag` package option is given (or, because the `tag` option is the default, when no package options are given.)

14.1 Writing to \@auxout

To do: *Might I need to do* `\let\protect\@unexpandable@protect` *instead of* `\noexpand`*, in the def of* `\\protected@writeaux`*?*

`\nc@writeaux@main` `\nc@protected@writeaux@main`

The `main` versions are exactly the same as what they replaced.

```
\newcommand\nc@writeaux@main {%
  \immediate\write\@auxout
}
\newcommand\nc@protected@writeaux@main {%
  \protected@write\@auxout
}
```

`\nc@writeaux@aux` `\nc@protected@writeaux@aux`

When you remove the `\immediate`, you have to expand whatever's in the argument at the time you invoke `\write`. `\IncludeName` and `\@percentchar`, and other exandables in `#2` will get expanded now. The `\@percentchar` and the `^^J`s are there because lines written to `\@auxout` must be on lines by themselves to satisfy BibTeX. The `^^J`s write newlines, and the `\@percentchar` eliminates a newline when the `aux` file is read in again later. Accommodating BibTeX requires special consideration several times below as well.

```
\newcommand\nc@writeaux@aux [1] {% args: write-text
  \eExecute {%
    \write\@auxout{\string\@auxtag{\IncludeName}{\@percentchar^^J#1^^J}}%
  }%
}
\newcommand\nc@protected@writeaux@aux [2] {% args: init-hook write-text
  \protected@write\@auxout{#1}{\string\@auxtag{\IncludeName}{\@percentchar^^J#2^^J}}%
}
```

`\@writeaux` `\@protected@writeaux`

We start with the `main` versions. We don't reserve the control sequences `\@writeaux` and `\protected@writeaux` because the hack to adapt other packages might have already defined it with `\providecommand`.

```
\let\@writeaux\nc@writeaux@main
\let\protected@writeaux\nc@protected@writeaux@main
```

14.2 Kernel redefinitions

`\@bibitem` `\@lbibitem` `\label` `\@citex` `\bibliography` `\nocite` `\addtocontents`

These are simple redefinitions of kernel functions. The changes are the substitutions for the writing commands described above.

```
\defcommand*\@bibitem [1] {%
  \item
  \if@filesw
    \@writeaux{\string\bibcite{#1}{\the\value{\@listctr}}}%
  \fi
  \ignorespaces
}
\DefName*{@lbibitem} {[#1]#2} {%
```

```
  \item[\@biblabel{#1}\hfill]%
  \if@filesw
    \begingroup
      \let\protect\noexpand
      \@writeaux{\string\bibcite{#2}{#1}}%
    \endgroup
  \fi
  \ignorespaces
}
\defcommand*\label [1] {%
  \@bsphack
  \protected@writeaux{}{\string\newlabel{#1}{{\@currentlabel}{\thepage}}}%
  \@esphack
}
\defcommand\addtocontents [2] {%
  \protected@writeaux
      {
       \let\label\Gobble
       \let\index\Gobble
       \let\glossary\Gobble
      }
      {\string\@writefile{#1}{#2}}%
}
\DefName*{@citex} {[#1]#2} {%
  \let\@citea\@empty
  \@cite {%
    \@for\@citeb:=#2\do {%
      \@citea
      \def\@citea{,\penalty\@m\ }%
      \edef\@citeb{\expandafter\@firstofone\@citeb}%
      \if@filesw
        \@writeaux{\string\citation{\@citeb}}%
      \fi
      \@ifundefined{b@\@citeb} {%
  \mbox{\reset@font\bfseries ?}%
  \G@refundefinedtrue
  \@latex@warning
    {Citation `\@citeb' on page \thepage \space undefined}%
}{% ELSE
        \hbox{\csname b@\@citeb\endcsname}%
      }%
    }%
  }{#1}% second arg to \@cite
}
\defcommand*\bibliography [1] {%
  \if@filesw
    \@writeaux{\string\bibdata{#1}}%
  \fi
  \@input@{\jobname.bbl}%
}
  \defcommand*\bibliographystyle [1] {%
  \ifx\@begindocumenthook\@undefined\else
    \expandafter\AtBeginDocument
  \fi
```

```
  {\if@filesw
    \@writeaux{\string\bibstyle{#1}}%
  \fi}%
}
\defcommand*\nocite [1] {%
  \@bsphack
  \@for\@citeb:=#1\do {%
    \edef\@citeb{\expandafter\@firstofone\@citeb}%
    \if@filesw
      \@writeaux{\string\citation{\@citeb}}%
    \fi
    \@ifundefined{b@\@citeb} {%
\G@refundefinedtrue
        \@latex@warning{Citation `\@citeb' undefined}%
      }{}%
    }%
  \@esphack
}
```

14.3 Checkpoints

\@writeckpt
\@wckptelt

The \@charlb, \@charrb, and \@percentchar stuff is to satisfy BibTeX (see above).

```
\defcommand*\@writeckpt [1] {%
  \if@filesw
    \write\@auxout{\string\@setckpt{#1}\@charlb\@percentchar}%
    {\let\@elt\@wckptelt
     \cl@@ckpt}%
    \write\@auxout{\@charrb}%
  \fi
}
\defcommand\@wckptelt [1] {%
  \write\@auxout{\string\setcounter{#1}{\the\@nameuse{c@#1}}}%
}
```

14.4 Including

\IncludeName
\ParentName
\nc@includename@<N>

```
\newcommand\IncludeName {%
  \@nameuse{nc@includename@\theIncludeDepth}%
}
\newcommand\ParentName {%
  \ifnum\value{IncludeDepth}= 0
    \jobname
    \FrankenWarning{newclude}{Requested name of parent of principle source}%
  \else
```

The incrementation of the IncludeDepth counter is local to the group.

```
    \begingroup
      \advance\c@IncludeDepth by \m@one
      \@nameuse{nc@includename@\theIncludeDepth}%
    \endgroup
  \fi
```

```
}
\NewName {nc@includename@0} {} {\jobname}
```

\nc@include
\nc@@include

To do: *dox*

```
\newcommand\nc@include [2][] {% args: hook filename
  \@ifnextchar [ {%]
      \nc@@include{#1}{#2}%
    }{% ELSE
    \nc@@include{#1}{#2}[]%
  }%
}
\NewName{nc@@include}{#1#2[#3]} {% args: prehook filename posthook
  \IfAllowed{#2} {%
      \nc@t@b           % surround the \include with something
      \stepcounter{IncludeDepth}%
      \DefName*{nc@includename@\theIncludeDepth} {} {#2}%
      \let\@writeaux\nc@writeaux@aux
      \let\protected@writeaux\nc@protected@writeaux@aux
```

Now execute the text of the optional argument to `\include`.

```
      #1%
      \@input@{#2.tex}%
      #3%
      \@writeckpt{#2}%
      \let\@writeaux\nc@writeaux@main
      \let\protected@writeaux\nc@protected@writeaux@main
```

We mustn't restore the counter before we have finished using it.

```
      \addtocounter{IncludeDepth}{\m@ne}%
      \nc@t@b           % surround the \include with something
    }{% ELSE
```

If the file is not allowed, we don't load it and do two things instead. We execute the part's checkpoint, then we write out the part's auxcommands and checkpoint again. We must handle the case when the auxcommands isn't defined; but the checkpoint will always be defined.

```
    \@ifundefined{cp@#2} {%
%       \DTypeout{No information on part [#2]!}%
      }{% ELSE
      \@nameuse{cp@#2}%
      \if@filesw%
        \nc@write@auxcommands{#2}%
        \nc@write@ckpt{#2}%
      \fi%                                                  if@filesw
    }%                                                      if@undefined
  }%                                                        IfAllowed
}%
```

\nc@write@auxcommands
\nc@write@ckpt

To do: *dox*

`\meaning` produces catcode 12's for all chars except spaces which are 10. Begin making definitions with `\catcode‘\^^M=12` (other).

```
\begingroup
\catcode‘\^^M=12 %% double percents mean they’re there only because of the catcode
%%
```

```
\Global\DefName*{nc@write@auxcommands} {#1} {% args: partname
  \@ifundefined{nc@auxcommands@#1} {%
    }{% ELSE
    \write\@auxout{\string\@auxtag{#1}\@charlb\@percentchar}%
    \EExpand*\csname nc@auxcommands@#1\endcsname\In {%
      \edef\nc@t@a {%
        \expandafter\strip@prefix\meaning ##1%
      }%
    }%
    \edef\nc@t@a {\expandafter\nc@strip@M\nc@t@a\@nil}%
%    \DTypeout{The auxcommands: \meaning\nc@t@a}%
    \begingroup %%
      \catcode`\^^M=12 % other
      \nc@for\nc@t@b:=\nc@t@a\do {%
%  \DTypeout{auxcommand ITEM: \meaning\nc@t@b}%
\EExpand\nc@t@b\In {%
  \write\@auxout{##1}%
}%
      }%
    \endgroup %%
    \write\@auxout{\@charrb}%
  }%
}%%
\Global\DefName*{nc@write@ckpt} {#1} {% args: partname
  \write\@auxout{\string\@setckpt{#1}\@charlb\@percentchar}%
  \EExpand*\csname cp@#1\endcsname\In {%
    \edef\nc@t@a {%
       \expandafter\strip@prefix\meaning ##1%
     }%
  }%
  \edef\nc@t@a {\expandafter\nc@strip@M\nc@t@a\@nil}%
  \begingroup %%
    \catcode`\^^M=12 % other
    \nc@for\nc@t@b:=\nc@t@a\do {%
%      \DTypeout{checkpoint ITEM: \meaning\nc@t@b}%
      \EExpand\nc@t@b\In {%
\write\@auxout{##1}%
      }%
    }%
  \endgroup %%
  \write\@auxout{\@charrb}%
}%%
```

\nc@for
\nc@forloop
\nc@iforloop

\nc@for is like the kernel's \@for but divides its list at ^^M_{12} instead of ,.

```
\Global\NewName{nc@for} {#1:=#2\do#3} {% FIX (what?)
  \expandafter   \def %%
    \expandafter \@fortmp %%
    \expandafter {#2}%
  \ifx\@fortmp\@empty \else %%
    \expandafter\nc@forloop#2^^M\@nil^^M\@nil\@@#1{#3}%
  \fi %%
}%%
\Global\NewName{nc@forloop} {#1^^M#2^^M#3\@@#4#5} {%
  \def#4{#1}%
```

```
  \ifx #4\@nnil \else %%
    #5%
    \def#4{#2}%
    \ifx #4\@nnil \else %%
      #5%
      \nc@iforloop #3\@@#4{#5}%
    \fi %%
  \fi %%
}%%
\Global\NewName{nc@iforloop} {#1^^M#2\@@#3#4} {%
  \def#3{#1}%
  \ifx #3\@nnil %%
    \expandafter\@fornoop %%
  \else %%
    #4%
    \relax %%
    \expandafter\nc@iforloop %%
  \fi %%
  #2\@@#3{#4}%
}%%
```

`\nc@strip@M` This strips a final `^^M`$_{12}$ from its argument.

To do: *I think this could be built in to* `\nc@for`.

```
\Global\NewName{nc@strip@M} {#1^^M\@nil} {#1}%%
```

Finish making definitions with `\catcode‘\^^M=12`.

```
\endgroup
```

`\@auxtag` `\@@auxtag` We both execute and save.

To do: *efficiency? check only once, then redefine auxtag?*

To do: *dox*

.

I could use `\EExpand\In` for clarity, but I go for efficiency on this crucial macro.

Begin making definitions with `\catcode‘\^^M=12` (other).

```
\begingroup
\catcode‘\^^M\active %% double percents mean they’re there only because of the catcode
%%
\Global\NewName*{@auxtag} {#1} {% args: partname
  \begingroup %%
    \catcode‘\^^M\active %%
    \@@auxtag{#1}%
}%%
\Global\NewName*{@@auxtag} {#1#2} {% args: partname auxcommands
    \@ifundefined {nc@auxcommands@#1} {%
\nc@toks@a={#2}%
      }{% ELSE
      \expandafter   \nc@toks@a %%
\expandafter \expandafter %%
\expandafter {\csname nc@auxcommands@#1\endcsname#2}%
    }%
    \expandafter\xdef\csname nc@auxcommands@#1\endcsname{\the\nc@toks@a}%
    #2%
  \endgroup %%
}%%
```

\@setckpt ***To do:*** *dox*

\@@setckpt

```
\Global\DefName*{@setckpt} {#1} {% args: partname
  \begingroup %%
    \catcode`\^^M\active %%
    \@@setckpt{#1}%
}%%
```

Finish making definitions with `\catcode`\^^M=12`.

```
\endgroup
```

The `\endgroup` terminates the change in catcode.

```
\newcommand*\@@setckpt [2] {% args: partname checkpoint
    \expandafter\gdef\csname cp@#1\endcsname{#2}%
  \endgroup
}
```

What this does is effectively remove all the tags. The end of document hook is processed before the closing processing of the `aux` files, during which checking for things like undefined references is done. At this point we do not need the tags.

```
\AtEndDocument {%
  \let\@auxtag\@secondoftwo
}
```

15 Allocate

The code in this section is processed when the `allocate` package option is given.

Warning: *This code has not been well tested yet. The output routine of LaTeX is very complicated, and unforseen problems might arise.*

\NextAux
\DynamicAux
\StaticAux

The macro `\NextAux` changes `\@auxout` to a new stream if one is available, and gives an error otherwise. The macro is implemented in dynamic and static ways which can be selected with `\DynamicAux` and `\StaticAux {`⟨*number of streams*⟩`}`. The number of streams can be from 2 to 16. The dynamic implementation is the default; I do not see much use for the static implementation at present. The `static` option is the equivalent of the declaration `\StaticAux{10}`. The `dynamic` selects the dynamic implementation.

`\StaticAux` can be invoked after `\DynamicAux`, but not the other way around (at least, the streams allocated by `\StaticAux` are not recovered). Macros which use `\NextAux` do not have to know whether the implementation is static or dynamic.

15.0.1 Wheels

The output streams are manipulated with the help of a data structure I call a *wheel*.

A *wheel* has 0 or more *spokes* and can be *rolled*. Each spoke is a TeX token, probably a control sequence name, and has an internal name. You can access the spoke at the 12 o'clock or "top" position of a wheel. In computerese, a wheel is a circularly linked list of tokens, and the operation of rolling moves a pointer along it in a certain direction by one element.

Wheels and operations on wheels are local.

\InitWheel
\DefWheel

You make a wheel either with `\InitWheel {`⟨*\csname*⟩`}`, which makes

⟨*csname*⟩ a wheel with no spokes, or `\DefWheel {`⟨*csname*⟩`}{`⟨*spokes*⟩`}`, which makes a wheel with ⟨*spokes*⟩ for spokes. The first spoke in ⟨*spokes*⟩ is the top, the second will be top after one roll, and the first will be top again after n rolls, if there are n spokes.

`\Roll` `\Top` `\AddSpokes`

Wheels are rolled by `\Roll {`⟨*wheel*⟩`}`. Spokes can be added to a wheel with `\AddSpokes {`⟨*wheel*⟩`}{`⟨*spokes*⟩`}`. When n spokes are added, the previous top will be at the top after n rolls. `\Top {`⟨*wheel*⟩`}` expands eventually to the top spoke, which then can further expand, and so on.

`\IfTop`

`\IfTop {`⟨*wheel*⟩`}{`⟨*spoke*⟩`}{`⟨*true clause*⟩`}{`⟨*false clause*⟩`}` compares the top of ⟨*wheel*⟩ with ⟨*spoke*⟩ using `\ifx`, and executes either ⟨*true clause*⟩ or ⟨*false clause*⟩ as appropriate. (The *newclude* package doesn't actually use this command; it's provided to "round out" the wheel data structure.)

Warning: *Don't put more than one token as the second argument to* `\IfTop`*.*

15.0.2 Preliminaries

We require the *afterpage* package. The intuitive justification is that `\write`s are delayed until the current page is shipped out. We need to keep an output stream open until its last `\write` has been actually handled; after that, the stream can be made available again.

```
\RequirePackage{afterpage}
```

`\nc@aux@wheel`

We use the wheel structure to handle both the static case and the dynamic case. The spokes of the wheel are macros `\nc@auxout@`⟨n⟩. Their first-level expansion is ⟨n⟩, a positive integer from 0 to 15. Each spoke has two corresponding macros. `\nc@auxout@`⟨n⟩`@stream` is a stream name allocated by `\newwrite`. `\nc@auxout@`⟨n⟩`@inuse` is a global boolean which is `true` iff the corresponding stream is currently in use.

```
\InitWheel\nc@aux@wheel
```

`\nc@count`

We need an internal counter. Notice that the stream numbers used in the auxwheel start at 0, so the stream associated with with the numeral 4 is the fifth stream.

```
\newcounter{nc@count}
```

`\nc@aux@wheel@size`

`\nc@aux@wheel@size` is a pseudo-counter that holds the present size of the aux wheel. In the static case it never changes and is set only for consistency.

```
\ReserveCS\nc@aux@wheel@size
```

`\NextAux`

```
\ReserveCS\NextAux
```

The kernel allocates two streams for the include system, `\@mainaux` and `\@partaux`. The auxwheel is initialized with these two streams. The first, corresponding to the principle source, is always marked in use.

To do: *Reserve the stream names.*

```
\newboolean{@nc@auxout@1@inuse@}

\ReserveName{nc@auxout@1}
\NewName*{nc@auxout@1} {} {1}

\ReserveName{nc@auxout@1@stream}
\expandafter\let\csname nc@auxout@1@stream\endcsname\@partaux
```

\nc@init@aux@wheel We initialize the wheel with the first spoke.

```
\newcommand\nc@init@aux@wheel {%
  \EExpand\csname nc@auxout@1\endcsname\In {%
    \AddSpokes\nc@aux@wheel##1%
  }
}
```

15.0.3 Static allocation

\StaticAux nonpositive numbers are treated the same as 1.

To do: *bounds check; the counter's max should be one less than the number, since we have stream 0.*

```
\newcommand\StaticAux [1] {%
  \def\nc@aux@wheel@size {#1}
  \setcounter{nc@count}{2}
  \nc@init@aux@wheel
  \@whilenum \value{nc@count} <= \nc@aux@wheel@size
    \do {%
```

First define the macros that make up the wheel itself to be their spoke numbers.

```
     \eExpand*\thenc@count\In {%
       \NewName*{nc@auxout@\thenc@count} {} {%
          ##1%
       }%
     }
```

Next allocate the corresponding stream.

```
     \EExpand\csname nc@auxout@\thenc@count@stream\endcsname\In {%
       \@nameuse{newwrite}##1%
     }
```

Next create the corresponding flag (they start `false`).

```
     \provideboolean{@nc@auxout@\thenc@count @inuse@}
```

Now add the spoke itself.

```
     \EExpand\csname nc@auxout@\thenc@count\endcsname\In {%
       \ReserveCS#1%
       \AddSpokes\nc@aux@wheel##1%
     }
     \stepcounter{nc@count}
   }
  \def\NextAux {%
    \Roll\nc@aux@wheel
    \@nameuse{if@nc@auxout@\Top\nc@aux@wheel @inuse@}%
      \MonsterError{newclude} {%
        You can't nest \protect\include this deeply!%
      }%
    \else
      \Elet\@auxout\csname nc@auxout@\Top\nc@aux@wheel @stream\endcsname
    \fi
  }%
}
```

15.0.4 Dynamic allocation

\DynamicAux
\nc@addnewauxstream

```
\newcommand\DynamicAux {%
  \def\nc@aux@wheel@size {1}
  \nc@init@aux@wheel
  \def\NextAux {%
    \Roll\nc@aux@wheel
    \@nameuse{if@nc@auxout@\Top\nc@aux@wheel @inuse@}%
      \nc@addnewauxstream
    \fi
```

Either the top spoke was not in use, or we have added a fresh spoke at the top – so the top spoke represents what we want.

```
    \Elet\@auxout\csname nc@auxout@\Top\nc@aux@wheel @stream\endcsname
    \typeout{NextAux has just set auxout to stream
            \the\csname nc@auxout@\Top\nc@aux@wheel @stream\endcsname.
             We are using spoke number
            \csname nc@auxout@\Top\nc@aux@wheel\endcsname.}
    }%
}
```

It works out that the new spoke should have a spoke number one greater than the current wheel size. We use the `nc@count` counter to find this number.

```
\newcommand\nc@addnewauxstream {%
  \setcounter{nc@count}{\nc@aux@wheel@size}%
  \stepcounter{nc@count}%
  \typeout{Allocating another spoke (spoke number \thenc@count)}%
```

First we add the spoke itself, then initialize the corresponding objects.

```
  \EExpand*\csname nc@auxout@\thenc@count\endcsname\In {%
    \AddSpokes\nc@aux@wheel##1%
  }%
  \EExpand*\thenc@count\In {%
    \DefName*{nc@auxout@##1} {} {##1}%
    \provideboolean{@nc@auxout@##1@inuse@}%
    \def\nc@aux@wheel@size {##1}%
    \EExpand*\csname nc@auxout@##1@stream\endcsname\In {%
      \@nameuse{newwrite}####1%
    }%
  }%
}
\DynamicAux
```

15.0.5 Including

\nc@include

The only way I see how to set the `inuse` flag to `false` at the proper time is to use the *afterpage* package. What I would really like is to `\write` something with side effects.

```
\newcommand\nc@include [2][] {%
  \if@filesw
    \immediate\write\@mainaux{\string\@input{#2.aux}}%
  \fi
  \@tempswatrue
  \if@partsw
```

```
    \@tempswafalse
    \edef\reserved@b{#2}%
    \@for\reserved@a:=\@partlist\do
      {\ifx\reserved@a\reserved@b\@tempswatrue\fi}%
  \fi
  \if@tempswa
    \stepcounter{IncludeDepth}%
```

`\nc@t@c` is going to preserve the current `aux` stream number to restore `\@auxout`, in case there is a nested `\include`.

```
    \edef\nc@t@c {%
      \the\@auxout
    }%
    \if@filesw
      \NextAux
      \openout\@auxout #2.aux
      \write\@auxout{\relax}%
      \expandafter\global
        \csname @nc@auxout@\Top\nc@aux@wheel @inuse@true\endcsname
```

`\nc@include@finish@<N>` The next line defines the macro `\nc@include@finish@`⟨*n*⟩ to close the output stream that is presently open. We have an interesting task here of getting certain unique information to macros after the `\@input` when we might end up recursing during the `\@input`. To do this, we immediately expand the variables we need and store them in a macro which will *not* be altered by a recursion of `\include`. We have set up the `IncludeDepth` counter to allow us to define such a macro, which is unique to *this* instance of `\include`.

Warning: *The macro names* `\nc@include@finish@`⟨*n*⟩ *where* ⟨*n*⟩ *is an integer are overwritten, that is, they are not allocated in a safe way.*

The following lines are intended to make this definition, where `<D>` represents the current value of `IncludeDepth`, `<P>` represents the spoke number of the current top of `\nc@aux@wheel`, and `<S>` represents the stream number for the current part, i.e., the current value of `\@auxout`, and `<X>` represents the stream number that was current before `\include` got called (this is saved in `\nc@t@c`).

```
\def\nc@include@finish@<D> {%
  \closeout<S>%
  \global\chardef\@auxout=<X>%
  \afterpage{\global\@nc@auxout@<P>@inuse@false}
}
```

```
      \EExpand\theIncludeDepth\In {%                    ##1
      \EExpand\the\@auxout\In {%                        ####1
        \DefName{nc@include@finish@##1} {} {%
          \closeout####1%
          \global\chardef\@auxout=\nc@t@c
          \typeout{Restored auxout to stream number
                           \nc@t@c \space (old: \the\@auxout)}
          \typeout{executing afterpage}%
          \EExpand\csname @nc@auxout@\Top\nc@aux@wheel
                          @inuse@false\endcsname\In {% ########1
            \afterpage{%
              \typeout{Finishing.  auxout is now \the\@auxout;  current spoke
```

```
                    is \csname nc@auxout@\Top\nc@aux@wheel\endcsname\space
                     with stream number
                  \the\csname nc@auxout@\Top\nc@aux@wheel @stream\endcsname
                    }%
              \global########1%
            }%                                                    Afterpage
          }%                                                      EExpand
        }%                                                        DefName
      }}%                                                         EExpand
    \fi %                                                         \if@filesw
    \nc@t@b           % surround the \include with something
```

Now execute the text of the optional argument to `\include`. Notice that if we change to a new `aux` file, we should do it before the optional argument. This is important so that sectioning commands will appear in the right order. If the sectioning command were to write to `\@mainaux`, then it would come *after* the whole included `aux` file, instead of before it.

```
    #1%
    \@input@{#2.tex}%
    \@writeckpt{#2}%
    \if@filesw
      \csname nc@include@finish@\theIncludeDepth\endcsname
    \fi
    \nc@t@b           % surround the \include with something
```

We mustn't restore the counter before we have finished using it.

```
    \addtocounter{IncludeDepth}{\m@ne}%
```

If the file is excluded by the `\includeonly` command, we don't load it and execute the file's checkpoint instead.

```
  \else
    \@nameuse{cp@#2}%
  \fi
}
```

15.0.6 Checkpoints

`\@writeckpt` `\@wckptelt`

We must redefine the macros which write the checkpoints. `\@auxout` is substituted for `\@partaux`; I think this change should be in the kernel anyway! And we remove the `\immediate`s.

```
\defcommand\@writeckpt [1] {%
  \if@filesw
    \write\@auxout{\string\@setckpt{#1}\@charlb}%
    \begingroup
      \let\@elt\@wckptelt
      \cl@@ckpt
    \endgroup
    \write\@auxout{\@charrb}%
  \fi
}
\defcommand\@wckptelt [1] {%
  \protected@write\@auxout{}{\string\setcounter{#1}{\the\@nameuse{c@#1}}}%
}
```

15.0.7 Wheels

\InitWheel
\Roll
\IfTop
\Top
\AddSpokes

A wheel is implemented as a macro. The tokens of its first-level expansion are the spokes, the top being the first.

```
\newcommand\InitWheel [1] {% args: wheel
  \InitCS#1%
}
\newcommand\Roll [1] {% args: wheel
  \edef #1{%
    \expandafter\nc@roll #1\nc@llor
  }%
}
\ReserveCS\nc@llor
\NewNameDef{nc@roll} {#1\nc@llor} {%
  \@cdr#1\@nil\@car#1\@nil
}
\newcommand\Top [1] {% args: wheel
  \E@car #1\@nil
}
\newcommand\IfTop [4] {% args: wheel token true false
  \EExpand#1\In {%
    \edef\nc@t@b {%
      \expandafter\noexpand\@car##1\@nil
    }%
  }%
```

At this point, the first-level expansion of `\nc@t@b` is a single token, the top of the wheel. We `\let \nc@t@a` to this token.

```
  \Elet\nc@t@a\nc@t@b
  \let\nc@t@b #2%
  \ifx\nc@t@a\nc@t@b
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
\newcommand\AddSpokes [2] {% args: wheel spokes
  \EExpand#1\In {%
    \def #1{#2##1}%
  }%
}
```

16 Benign packages

\DeclareFormattingPackage
\nc@formatting@packages

```
\newcommand\DeclareFormattingPackage [1] {%
  \addto@macro\nc@formatting@packages{,#1}%
}
\newcommand\nc@formatting@packages {times,helvetic}
```

Index

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

T

U

V

W

X

Z