The newclude LaTeX package

A new system for including files (Frankenstein's backbone)

Matt Swift <swift@alum.mit.edu>

virhuiai@qq.com 翻译

Version: 2 Date: 1999/11/02

Documentation revision: 1999/11/02

摘要

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.

Newclude 是一个向后兼容的重新实现 LaTeX 文件包含系统的工具。它的主要新特性包括：（1）不需要在被包含的文件前后使用`\clearpage` 命令，（2）可以嵌套使用`\include` 命令，（3）提供在文件包含前后执行的钩子。通过使用单个 `aux` 文件，Newclude 实现了前两个特性。

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*.

仍在开发中，但已经在许多情况下非常有用的是新命令，它们包括独立的 LaTeX 文件的部分内容，这些文件也可以单独处理（即，包含`\documentclass`，`\begin{document}` 等的文件）。*Newclude* 吸收并取代了以前的 *includex* 包。

目录

第一部分　Discussion
讨论

1　Introduction
介绍

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.
我们称可能通过在`\include` 家族中使用的命令包含到另一个文档中的文件为*部分*。当一个部分在特定的处理运行中实际被包含时，我们称其为*已包含部分*，而当它未被包含时，我们称其为*未包含部分*。请注意，未包含的部分*并不*等同于从未成为`\include` 家族中的命令包含候选文件的文件。

The *newclude* package adds these features to the standard LaTeX inclusion system:
newclude 宏包为标准的 LaTeX 包含系统添加了以下功能：

1. Hooks `\AtBeginInclude` and `\AtEndInclude` are provided.
   提供了钩子命令 `\AtBeginInclude` 和 `\AtEndInclude` 。

2. Optional arguments to `\include` and friends override current values of `\AtBeginInclude` and `\AtEndInclude` .
   `\include` 和其它相关命令的可选参数可以覆盖当前 `\AtBeginInclude` 和 `\AtEndInclude` 的值。

3. `\include*` is like `\include` but with arbitrary commands rather than `\clearpage` s surrounding the part.
   类似于 `\include` ，但是其包含的部分不是由 `\clearpage` 命令围绕而成的，而是由任意命令组成的。

4. `\include` and friends can be nested.
   `\include` 和其它相关命令可以嵌套使用。

5. `\includeall` cancels the effect of `\includeonly` .
   `\includeall` 命令可以取消 `\includeonly` 命令的作用。

6. `\IfAllowed` ⟨*filename*⟩ is a new conditional that branches, depending on what has been declared in an `\includeonly` .
   `\IfAllowed` ⟨*filename*⟩ 是一个新的条件语句，其执行分支取决于在 `\includeonly` 命令中声明的内容。

7. Commands `\includedoc` etc. include a part that can be processed independently. These features are in development.
   命令 `\includedoc` 等可以包含可以独立处理的部分。这些功能正在开发中。

Newclude accepts three mutually-exclusive package options, with `tag` the default when no package option is given.
Newclude 接受三个互斥的包选项，当没有给出包选项时，`tag` 是默认选项。

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.
使用 `simple` 选项加载 *newclude* 只提供功能 1和 2。如果您不使用这些新功能中的任何一个，标准的 LaTeX 和 *newclude* 包含系统将表现相同，除了某些与`\include` 的新可选参数解析相关的不寻常和良性的奇怪情况，这些情况在该命令的文档中进行了讨论。

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.
选项 `tag` 和 `allocate` 各自使用不同的方法实现了上述所有功能。每种方法都引入了与标准 LaTeX 不同的差异，下面在第 4.2节和第 4.3节中进行了讨论。如果我发现如何使一种方法永远不劣于另一种方法，我将从包中删除另一个选项。

2 Usage

使用方法

\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 .
\include [⟨prehook⟩]{⟨filename⟩}[⟨posthook⟩] 的用法类似于标准的 LaTeX 命令 \include ，但它支持嵌套，并且当给出两个钩子参数时，它们的内容会在每次包含时插入到部分的开头和结尾，覆盖当前的 \AtBeginInclude 和 \AtEndInclude 的值。

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

Warning: *可选参数中的右方括号（] 们）必须用花括号括起来，以避免混淆参数解析器。*

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).*

Warning: *如果在\include 命令的必需参数⟨filename⟩（可带可选的空格）之后紧跟着一个左方括号（[），那么它将被视为⟨posthook⟩参数的起始位置。如果您想在该位置放置实际的左括号字符，您必须在其前面加上一些东西来终止 TeX 对可选参数的搜索，例如\relax 、或段落分隔符（显式或隐式）。*

The commands \AtBeginInclude ⟨tokens⟩ and \AtEndInclude ⟨tokens⟩ are analagous to standard LaTeX's commands \AtBeginDocument ⟨tokens⟩ and \AtEndDocument ⟨tokens⟩.
命令\AtBeginInclude ⟨tokens⟩和\AtEndInclude ⟨tokens⟩类似于标准的 LaTeX 命令\AtBeginDocument ⟨tokens⟩和\AtEndDocument ⟨tokens⟩。

FIX: multiple instances concatenate?
修复：多个实例连接?

FIX give name to what's held by atbegininclude so that an override can mention it
FIX 将在 atbegininclude 中保存的内容命名，以便覆盖可以提及它。

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.
当给`\include` 添加可选参数⟨*prehook*⟩时，其内容将替代用`\AtBeginInclude` 指定的内容，仅用于该次包含。同样地，⟨*posthook*⟩将替代用`\AtEndInclude` 指定的内容，仅用于该次包含。

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:
举个例子，将`\chapter` 声明放在⟨*prehook*⟩参数中，可以让章节名称以及可选的相应 LaTeX 标签保留在包含文件中，而不是被包含的文件中：

```
\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.
⟨*posthook*⟩ 参数可以用于限定或撤销在 ⟨*prehook*⟩ 或包含文件中的声明。例如，可以在 `\include` 前后简单地添加这些声明，而不会产生不良影响。FIX: 更好的例子。

```
\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

`\IncludeSurround`

`\DefaultIncludeSurround`

has been specified with \AtEndInclude .
\include*[⟨prehook⟩]{⟨filename⟩}[⟨posthook⟩] 和 \include 类似，但省略了通常包含的 \clearpage ，它们被替换为 \IncludeSurround ，默认为 \DefaultIncludeSurround 。\IncludeSurround 的内容被插入到 ⟨prehook⟩ 或使用 \AtBeginInclude 指定的任何内容之前，以及 ⟨posthook⟩ 或使用 \AtEndInclude 指定的任何内容之后。

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` *.*

在\include* 之后会插入一个空格，除非紧接着有一个\%来抑制它。加上被包含文件中的尾随空格，这可能会导致不必要的空格。因此，\DefaultIncludeSurround 被初始化为\par。当用户必须显式更改\IncludeSurround 以实现从主文件到包含文件的完全平滑流时，如果他们发现问题，他们更有可能咨询此文档。包和类的编写者在更改\DefaultIncludeSurround 时应考虑到这一困难。

\includeonly The \includeonly command is reimplemented, but its usage and behavior is the same as the standard LaTeX version.
这个命令\includeonly 被重新实现了，但是它的用法和行为与标准的 LaTeX 版本相同。

\includeall The \includeall command cancels the effect of any \includeonly command presently in effect.
\includeall 命令取消任何当前生效的 \includeonly 命令的影响。

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.)
如果你使用\includeonly 命令，每个文件单独一行，添加或删除需要包含的文件时，注释掉它们的行非常容易，但注释掉整个\includeonly 命令就变得

很麻烦了。然而，当你想要处理整个文档时，取消注释单个`\includeall` 命令非常容易。(或者可以在调用 LaTeX 命令行时插入`\includeall` 命令，等等。)

3 Experimental features
实验特性

`\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 nth instance of the environment within the file, or further specify the environment to be extracted in some other way.
`\includeenv` 命令用于将出现在 ⟨*filename*⟩ 中的单个 LaTeX 环境的内容包含进来。该环境由其名称（⟨*environment*⟩）和文件中该环境的一个实例（⟨*instance*⟩）来指定。目前，⟨*instance*⟩ 参数被忽略，因此它将始终是第一个出现在 ⟨*filename*⟩ 中的名称为 ⟨*environment*⟩ 的 LaTeX 环境的内容。未来，⟨*instance*⟩ 参数可以用于指定文件中环境的第 n 个实例，或以其他方式进一步指定要提取的环境。

FIX: right now they're required; skip text up to documentclass OR the target, then branch?
修复：目前它们是必需的；跳过文档类或目标之前的文本，然后分支？

Good preamble syntactic sugar:
良好的前导语法糖：
`\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.*

您可以在主要的辅助文件中插入一个\usepackage 并正确加载它。如果我们发现一个不是格式化包的\usepackage ，那么一种策略是在（主要的）辅助文件中插入相应的\usepackage ，然后在导言部分后退出。

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.*

你不能通过宏参数处理和语法糖跳过逐字文本。这意味着必须使用逐字方法进行重大的跳过重新实现。

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.
所包含的文件可以（但不是必须）拥有自己的\documentclass 命令和\begin{document} . . . \end{document} 对。\includeenv 通过处理前导文本（如果存在），跳过指定环境的开头之前的文本，处理环境的内容，然后跳过所包含部分的其余部分，提取指定的环境。

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*⟩.
请注意，虽然\begin{document} . . . \end{document} 对可能不严格限定一个 LaTeX 环境，但由于它看起来与环境完全一样，因此您仍然可以将⟨*environment*⟩设置为document，以提取⟨*filename*⟩的 document"环境" 的内容。

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.
当你有使用这个命令的冲动时,请考虑以下问题。也许你打算忽略的 \usepackage 对处理部分内容是必要的。也许它与已在顶层加载的包冲突。也许两者都是!当然，对于定义命令的命令，如 \newcommand ，人们期望在包中找到它们，也是同样的情况。

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:
LaTeX 源文件设计中存在一个与导言部分功能相关的深层问题。导言部分包含决定下面文档格式的声明。不幸的是，无法区分以下两种情况：

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
   声明表明某些标记将出现在文档中，这些标记在 LaTeX 内核中未定义，或者使用不同的语法。

2. declarations that describe how a certain instance of the document should be formatted
   声明描述文档的某个实例应该如何格式化。

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.
第一类示例包括`\usepackage{url}`和`FIXexample`,第二类示例包括`\usepackage{times}`和`FIX`。当您想要将文档或其中的一部分包含在另一个文档中时，有必要进行区分，以便可以处理类别（1）中的声明并忽略类别（2）中的声明。

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.
采用有关前言使用的公约可以克服这个设计问题，但对于那些前言不遵守公约的遗留文件，它并不能解决问题。在前言中包含类别（1）声明的遗留文件必须进行修改或使用额外的命令进行特殊适配。

The convention I suggest is to `\usepackage{preamble}`. `\beginmarkup \endmarkup` . FIX. Can we arrange to load
我建议的惯例是使用 `\usepackage{preamble}`, `\beginmarkup \endmarkup` FIX。我们能否安排加载?

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.)
当 \includeenv 在被包含的部分中遇到 \usepackage 命令时，它会查看 \usepackage 命令中的参数中的包，如果该包尚未加载并且不在已知的完全属于类别 (2) 的包列表中，则会发出警告。(请参见下面的 \DeclareFormattingPackage 命令。)

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.
当然，\documentclass 命令也是一个类别 (1) 声明。目前，如果 \includeenv 检测到被包含文档中的 \documentclass 命令的参数与包含文档中的 \documentclass 命令的参数不同，它将发出警告并继续执行。将来，我希望通过使 \includeenv 针对特定的参数组合采取特定的操作来使此行为更加智能化。例如，如果被包含文档的类别暗示使用了父类别中未定义的标记，合适的操作是定义缺失的标记命令。另一方面，类别为 *report* 和类别为 *article* 的文档不 (我认为) 声明不同的标记，因此在这种情况下不应发出警告。

\includeenv* \includeenv* is analogous to \include*, that is, it surrounds the included part with \IncludeSurround rather than \clearpage .
\includeenv* 类似于\include*，即它使用\IncludeSurround 环绕包含的部分，而不是\clearpage 。

\includedoc \includedoc [⟨*prehook*⟩]{⟨*file name*⟩}[⟨*posthook*⟩] is shorthand for(是一种
\includedoc* 简写形式，等同于) \includeenv [⟨*prehook*⟩]{⟨*filename*⟩}{⟨*document*⟩}{⟨⟩}
[⟨*posthook*⟩].

`\includedoc*` is analogous(类似) to `\includeenv*`.

4 Options
选项

4.1 Simple
简单选项

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.
如果给定了 `simple` 选项，则提供的唯一新功能是钩子（参见上文中的特性1和 2）。与标准的 LaTeX 一样，`\clearpage` 将包围`\include`，嵌套的`\include` 会导致错误。与标准的 LaTeX 不同的是，*Newclude* 在扫描可能的可选参数时会有所不同。

4.2 Tag
标签

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:
`tag` 选项会让 LaTeX 只使用一个 `aux` 文件。这个选项是默认的，并且效果很好。我意识到与内核的包含系统有以下两个不同之处：

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.
果在处理文档的某个部分时停止了 LaTeX 进程，则从该点开始在 `aux` 文件中通常存储的所有信息都将丢失。在内核系统中，再次处理文档两次会恢复以前生成的所有部分的 `aux` 信息

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.。
如果 LaTeX 总是在 `\nonstopmode` 下调用（例如通过 AUC-TeX ），那么只有在出现灾难性错误导致即使 `\nonstopmode` 也无法继续处理时，才会出现这种差异。

2. Other packages and classes that redefine kernel commands that write to `\@auxout` will cause problems.
其他重新定义内核命令并写入 `\@auxout` 的包和类会导致问题。

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:
第一个差异必须被接受。第二个差异可以根据情况逐个解决，通过特别编写与这些包和类的兼容性代码。我打算这样做。以下是我所知道的这些包和类的列表：

⟨*none so far*⟩ If you discover any more for this list, please write me!
如果您发现了更多的内容，请写信给我！

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.
现在，将其他软件包修改为与 *newclude* 兼容也非常容易。请参见下面的第 6节，其中包括相关内核命令的列表。

4.3 Allocate
分配

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.
第二种方法（`allocate` 选项）是我第一次尝试的解决方案，直到我确定它在任何情况下都没有优势，它将继续作为一个选项。

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.
`allocate` 选项会使 LATEX 在需要时为每个部分动态分配 TEX 输出流。处理部分时分配流，并在该部分最后一页被弹出后回收。与旧系统一样，每个部分都会创建单独的 `aux` 文件。此实现的限制是 TEX 仅拥有 16 个输出流。每个命令`\tableofcontents` 、`\listoffigures` 、`\listoftables` 、`\makeglossary` 和`\makeindex` 都会使用一个输出流。剩余的流（减去包和类所需的流）可用于包含系统。如果有 n 个流可用，则可能的嵌套级别为 $n-1$，减去出现在同一页上的最大部分数量。例如，如果有 10 个可用流并且部分从不出现在同一页上（旧行为需要`\clearpage` ），则可能有 8 个嵌套级别（比旧系统多 8 个，而不是 7 个）。可以使用相同的方程式计算可能贡献到同一页的最大部分数量。注意：TEX 的分页算法会向前查找，直到有足够的材料填充一页。必须将所有新打开的 `aux` 文件都计算为对所讨论的页面的贡献，即使一些后来的文件在选择分页后实际上不会对页面产生贡献。

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.
allocate 解决方案本身有两种实现方式。系统要么从一开始就保留一定数量的输出流，要么在需要时动态地申请和释放它们。动态解决方案是默认的。目前我不认为静态解决方案有太多用处。如果动态系统申请了后续需要的流，则只是一个问题，即 *newclude* 或其他功能是否会发出关于无法再分配流的错误信号。

5 Programmers' interface
程序员接口

\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*⟩.
`\IfAllowed {`⟨*部件名称*⟩`}{`⟨*真值*⟩`}{`⟨*假值*⟩`}`，如果⟨*part-name*⟩在待包含文件列表中，则执行⟨*true*⟩，否则执行⟨*false*⟩。如果没有列表，则执行⟨*true*⟩。

\IncludeName `\IncludeName` expands to the name of the part currently being processed. In the toplevel source file, it will expand to `\jobname` .
`\IncludeName` 展开为当前正在处理的部分的名称。在顶层源文件中，它将展开为 `\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).
`\ParentName` 展开为包含当前正在处理的部分的名称。在顶层源文件中，展开`\ParentName` 将生成一个警告并展开为`\jobname` （ `\IncludeName` 也是如此)。

FIX: root source file? toplevel? master? principle source? glossary!
修复：根源文件？顶层文件？主文件？原则性文件？词汇表！

\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.
`\DeclareFormattingPackage {`⟨ *包名*⟩`}` 声明⟨ *包名*⟩为仅用于格式声明的包，即使用它的效果完全限于类别（2）。如果格式包出现在由`\includeenv` `\usepackage` ，不会发出警告。格式包的一个例子是 *times* 包。没有提供区分使用或不使用某些包选项的情况的工具，因此，除非无论传递哪些选项都是如此，否则不要将包声明为格式包。

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.
如果您给我发送格式化包的名称，我将在 *newclude* 的下一个版本中包括它们。与此同时，您可以在`newclude.cfg` 中声明它们。如果您愿意，也可以为您本地的格式化包执行相同的操作。将一个包声明为格式化包多次不会造成任何损害。

`\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.
`\Disable {`⟨*tokens*⟩`}` 提供了一种在使用 `\includeenv` 和相关命令时忽略其他命令的方法。如果你想让不带参数的宏 `\foo` 在某些部分完全被忽略，可以在包含你想要影响的部分之前随时发出命令 `\Disable{\let\foo\relax}`。如果 `\foo` 接受一个必选参数，则应写作 `\let\foo\Gobble`。如果 `\foo` 接受一个可选参数和一个必选参数，则应写作 `\let\foo\GobbleOM`。等等。对于其他示例，请参见 *moredefs* 包中的吞噬命令（ *newclude* 要求），或编写自己的命令。

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.

\Disable 的参数被累计并由命令\DisableAll 执行，在包含它时执行一个包含部分的组中。

There is no way to undo the effect of issuing a \Disable command.
没有办法撤销发出\Disable 命令的影响。

6 How to play nicely with *newclude* 如何与 *newclude* 友好地玩要

To adapt a package or class for use with the tag option of
为了使用 tag 选项，需要适应一个包或类。*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
}
```

第二部分 Implementation

7 Version control

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

These definitions must be the first ones in the file.
这些定义必须是文件中的第一个。

```
\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).
如果我们正在从\ProcessDTXFile 命令（请参见 *compsci* 包）加载此文件，则\JusTLoaDInformatioN 将被定义；否则我们假设它不是（这就是为什么有个" 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.)
如果我们从\ProcessDTXFile 加载，我们希望加载\DoXPackageS 中列出的软件包（需要为此文件排版文档），然后退出。否则，我们将以正常方式将此文件用作软件包，因此不执行任何操作。\DoXPackageS （如果有的话）在 dtx 文件中声明，如果您正在阅读此软件包的排版文档，将出现在其上方。（顺便说一句，使用空参数或\relax \usepackage 是可以的。）

```
\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.
现在我们检查是否为 LaTeX2e 并声明 LaTeX 包。

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

8 Review of the kernel's inclusion system 内核包含系统的回顾

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.
对于*主源文件*，会写入一个 aux 文件到磁盘中，对于每个被包含的*部分*，也会写入一个 aux 文件。之所以要为每个部分分别写入，是为了保留上次包含该部分时的信息，即使该部分在`\includeonly` 中被排除。假设一个部分被处理一次，在下一次运行中，它的名称被从`\includeonly` 列表中移除。即使这次运行不处理该部分，它的 aux 文件仍然会被读取，因为上次运行时被`\include` 的任何部分的 aux 文件都会被读取。但是，在这次运行中，该部分的信息不会被重新生成，因为该部分不会被处理。主 aux 文件在每次运行时都会重新

创建，因此如果该信息驻留在主 aux 文件中，它就会丢失。

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.
为了处理多个 aux 文件的写入，内核使用了 TEX 的两个输出流。当一个例程向辅助文件写入时，它会写入到 \@auxout，它是一个 其值要么是 \@mainaux，即主源文件的 aux 文件，要么是 \@partaux，即所有部分的 aux 文件。

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.
遇到\include 命令时，在决定是否实际加载该部分之前，内核会向\@mainaux 写入一个命令，以加载该部分的 aux 文件。主 aux 文件由\document 加载，因此在处理主源文件时，每次都会读取*所有*的 aux 文件。

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.
如果一个部分实际上被加载了，那么会在这个部分的 aux 文件中写入一个检查点，其中包含了计数器的快照（即所有 LATEX 计数器的值的记录）。在下一次运行时，如果这个部分实际上没有被加载，那么它的 aux 文件中的信息已经被 \document 处理过了。处理检查点会定义一个宏，当调用它时会恢复计数器状态。当 \include 实际上没有加载一个部分时，它会调用这个检查点宏来改变当前计数器状态。

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 state-

ments in a principle source will bring havoc on the page numbers?
这个系统有优点也有缺点。在某些情况下，保持参考文献、引用、交叉引用和页码的最新状态非常有用，但有时结果可能令人困惑，因为检查点没有记录。（也许在第二版的 LaTeX 手册中会有解决方法。）除了阅读代码或通过艰难的方式找出，有谁能猜到在主要源文件中重新排列两个“停用”的\include ?

9 Discussion of *newclude*'s inclusion system
newclude 包中的包含系统讨论

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 \writes . Then we can close \@partaux immediately and reopen \@mainaux.
简单地删除包含部分周围的\clearpage 会导致延迟执行\write 命令的问题。假设一个以\write 命令结束的部分在页面中间结束，而另一个\write 出现在主要源代码中立即（或很快）之后的包含中。第一个必须写入\@partaux，而第二个必须写入\@mainaux。如果我们在第一个\write 仍然挂起（即在当前页面已经被输出之前）时关闭\@partaux，那么\write 将会被写入到一个关闭的流中，因此将会转到日志文件和终端。\clearpage 通过刷新所有挂起的\writes 解决了这个问题。然后我们可以立即关闭\@partaux 并重新打开\@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.
成功删除\clearpage 可以通过两种方式实现，一是整个文档只使用一个辅助文件，二是分配额外的输出流，以便在所有对\@partaux 的\write 完成后，延

迟关闭它直到当前页面已被输出。

10 Package initialization
包初始化

```
\RequirePackage{moredefs}

\InitCS\sc@t@a
\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.
上述选项处理会在给出 `simple` 选项时加载文件 `simple.sto`。加载完毕后，处理过程停止。当没有给出 `simple` 选项时，*newclude* 包的代码会在第 12 节继续执行。

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
常见

The code in this section is common to the `tag` and `allocate` options.
本节中的代码适用于 `tag` 和 `allocate` 选项。

`\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.
对于嵌套的`\include` ,我们需要一些方式来区分它们,因此我们使用`IncludeDepth` 计数器来跟踪嵌套深度。

```
\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?
我认为，为列表中的每个包含部分定义一个宏比为每个部分可能搜索两次列表更有效。对于使整个过程更有效的其他意见呢?

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*⟩.
我们使用通常的 LaTeX 技巧，未定义的控制序列与`\relax` 相等。空控制序列*不相同*。应该跟随⟨*true clause*⟩，然后是⟨*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.
这是主要用户命令。临时变量`\nc@t@b` 包含了实际包含文件周围的内容。

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

```
120     \nc@include
121   }%
122 }
```

13 Experimental common
实验共同点

\Disable
\DisableAll

This allows the disabling hacks.

```
123 \ReserveCS\DisableAll
124 \newcommand\Disable [1] {%
125   \g@addto@macro\DisableAll{#1}%
126 }
```

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.
我们首先考虑如何停止输入文件。想法是让部分的`\end{document}` 命令调用`\endinput` 。但是，有一个问题是当你不希望它们出现时，`\end{document}` 之后一行的字符会被插入。为了克服这个限制，我们需要做一些工作。

\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!
我们将向这个宏添加一堆命令，旨在将所有东西及其相关的内容都设置为注释的`\catcode` 。这将是一种蛮力的方法！

```
127 \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.
首先，记录一条消息，说明我们即将做一些疯狂的事情。如果出现问题，这可能会有所帮助。

```
128 \addto@macro\nc@radical@shutdown {%
129   \MonsterInfo{newclude}
130     {\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. . . .
现在我们开始添加\catcode 命令。这前两个命令可能是多余的；但以防万一有人改变了事情 . . . 。

```
\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.
接下来，我们定义一个命令，稍后将在循环中使用。

```
\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:
以下列表包含除了这三个特殊处理的键盘字符之外的所有字符：%#。第一个已经是注释，我们马上处理第二个。在下面的列表中，每个字符都被\catcode d 为注释。

```
\@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.
我们单独添加 #，因为将其放入刚刚使用的列表中很棘手或不可能。

```
\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.

我们用\endinput 结束宏。这必须在所有先前的内容之后，否则，TEX 会立即读取到行尾，使用常规的类别码。这是一个好理论，但我不确定它是否必要。

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

c@radical@shutdown@aftergroup

We need to use \nc@radical@shutdown this way.
我们需要这样使用 \nc@radical@shutdown。

```
\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
标签

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.)
当给出 `tag` 包选项时（或者因为 `tag` 选项是默认选项，当没有给出任何包选项时），本节中的代码将被处理。

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
```

```
296       \@writeaux{\string\citation{\@citeb}}%
297     \fi
298     \@ifundefined{b@\@citeb} {%
299 \G@refundefinedtrue
300         \@latex@warning{Citation `\@citeb' undefined}%
301       }{}%
302     }%
303   \@esphack
304 }
```

14.3 Checkpoints

\@writeckpt
\@wckptelt

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

```
305 \defcommand*\@writeckpt [1] {%
306   \if@filesw
307     \write\@auxout{\string\@setckpt{#1}\@charlb\@percentchar}%
308     {\let\@elt\@wckptelt
309      \cl@@ckpt}%
310     \write\@auxout{\@charrb}%
311   \fi
312 }
313 \defcommand\@wckptelt [1] {%
314   \write\@auxout{\string\setcounter{#1}{\the\@nameuse{c@#1}}}%
315 }
```

14.4 Including

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

```
316 \newcommand\IncludeName {%
317   \@nameuse{nc@includename@\theIncludeDepth}%
318 }
319 \newcommand\ParentName {%
320   \ifnum\value{IncludeDepth}= 0
321     \jobname
322     \FrankenWarning{newclude}{Requested name of parent of principle source}%
323   \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%
```

```
425       \nc@iforloop #3\@@#4{#5}%
426     \fi %%
427   \fi %%
428 }%%
429 \Global\NewName{nc@iforloop} {#1^^M#2\@@#3#4} {%
430   \def#3{#1}%
431   \ifx #3\@nnil %%
432     \expandafter\@fornoop %%
433   \else %%
434     #4%
435     \relax %%
436     \expandafter\nc@iforloop %%
437   \fi %%
438   #2\@@#3{#4}%
439 }%%
```

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

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

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

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

```
441 \endgroup
```

`\@auxtag` We both execute and save.

`\@@auxtag`

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).

```
442 \begingroup
443 \catcode`\^^M\active %% double percents mean they're there only because of the catcode
444 %%
445 \Global\NewName*{@auxtag} {#1} {% args: partname
446   \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
\@@setckpt

To do: *dox*

```
\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
分配

The code in this section is processed when the `allocate` package option is given.
当给出 `allocate` 软件包选项时，本节中的代码将被处理。

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.

```
475 \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.

```
476 \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%
```

```
552     }%
553   }%
554 }
555 \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.

```
556 \newcommand\nc@include [2][] {%
557   \if@filesw
558     \immediate\write\@mainaux{\string\@input{#2.aux}}%
559   \fi
560   \@tempswatrue
561   \if@partsw
562     \@tempswafalse
563     \edef\reserved@b{#2}%
564     \@for\reserved@a:=\@partlist\do
565       {\ifx\reserved@a\reserved@b\@tempswatrue\fi}%
566   \fi
567   \if@tempswa
568     \stepcounter{IncludeDepth}%
```

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

```
569     \edef\nc@t@c {%
570       \the\@auxout
571     }%
572     \if@filesw
573       \NextAux
574       \openout\@auxout #2.aux
575       \write\@auxout{\relax}%
576       \expandafter\global
577         \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}
}
```

```
578     \EExpand\theIncludeDepth\In {%                  ##1
579     \EExpand\the\@auxout\In {%                      ####1
580       \DefName{nc@include@finish@##1} {} {%
581         \closeout####1%
582         \global\chardef\@auxout=\nc@t@c
583         \typeout{Restored auxout to stream number
584                           \nc@t@c \space (old: \the\@auxout)}
585         \typeout{executing afterpage}%
586         \EExpand\csname @nc@auxout@\Top\nc@aux@wheel
587                         @inuse@false\endcsname\In {% ########1
588           \afterpage{%
589             \typeout{Finishing.  auxout is now \the\@auxout;  current spoke
590                    is \csname nc@auxout@\Top\nc@aux@wheel\endcsname\space
591                     with stream number
592                  \the\csname nc@auxout@\Top\nc@aux@wheel @stream\endcsname
593                    }%
594             \global########1%
595           }%                                                Afterpage
596         }%                                                  EExpand
```

```
597         }%                                                    DefName
598       }}%                                                     EExpand
599     \fi %                                                     \if@filesw
600     \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.

```
601     #1%
602     \@input@{#2.tex}%
603     \@writeckpt{#2}%
604     \if@filesw
605       \csname nc@include@finish@\theIncludeDepth\endcsname
606     \fi
607     \nc@t@b           % surround the \include with something
```

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

```
608     \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.

```
609   \else
610     \@nameuse{cp@#2}%
611   \fi
612 }
```

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.

```
613 \defcommand\@writeckpt [1] {%
614   \if@filesw
615     \write\@auxout{\string\@setckpt{#1}\@charlb}%
616     \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
良性软件包

\DeclareFormattingPackage
\nc@formatting@packages

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