The **pkgloader** package[*]

Michiel Helvensteijn

mhelvens+latex@gmail.com

April 29, 2018

Development of this package is organized at github.com/mhelvens/latex-pkgloader.
I am happy to receive feedback there!

---

1 Introduction

LaTeX can be extended by loading packages using a **\usepackage**, **\RequirePackage** or **\RequirePackageWithOptions** command. Similarly, documents load a document class using **\documentclass**, **\LoadClass** or **\LoadClassWithOptions**. Packages and classes can add definitions, change existing ones, or otherwise extend functionality of the language.

While the Turing-complete power of the TeX language is quite useful at times, it does make it all too easy for independent package authors to step on each others toes. CTAN is full of conceptually independent packages that cannot be loaded together, or break if they are not loaded in a specific order.

Yet, until now, there has been no automated package management to speak of. Document authors are usually told to avoid certain package combinations, or to load packages in some specific order. Occasionally, package authors patch their code to be aware of specific other packages, circumventing known conflicts. But this makes maintenance more difficult, because these package authors are 'mixing concerns'; they put code related to other packages into their own package. And it is all done in an ad-hoc fashion.

Enter pkgloader.

1.1 Package Description

Here is an example of main file for a LaTeX document which uses pkgloader:

```
1  \RequirePackage{pkgloader}
2      \documentclass{article}
3      \usepackage{algorithm}
4      \usepackage{hyperref}          } any order
5      \usepackage{float}
6  \LoadPackagesNow
7   ⋮                                 } optional
8  \begin{document} ... \end{document}
```

---

[*]This document corresponds to **pkgloader** v0.7.0, dated 2018/04/29.

The idea is to load `pkgloader` before loading any other package or class. It can then intercept all loading requests, analyze them and load them properly, taking this burden off the shoulders of the average document author.

Between the second and fifth line, the loading of all packages and classes is postponed. The **\LoadPackagesNow** command then loads the packages in some valid order. This also happens automatically upon reaching the end of the preamble. During this process, 'conflict resolving' code may also be run, meant specifically to make other packages work together properly. If the above code were compiled without `pkgloader`, the given package order would cause errors. The main advantage to this approach is that the complexity of dealing with package conflicts is moved to the `pkgloader` package and handled in a systematic manner.

If you are a document author, this may be all you need to know to use `pkgloader`. If you are interested in more advanced functionality, read on!

1.2 The **pkgloader** Area

\RequirePackage {`pkgloader`**} ... \LoadPackagesNow**

The `pkgloader` area is the area between **\RequirePackage{**`pkgloader`**}** and **\LoadPackagesNow**. Within it, the three traditional package-loading commands are 'hijacked', recording information rather than loading packages directly. Also, a **\Load** command is available, which offers more flexibility in regulating package loading behavior.

`pkgloader` accepts sets of rules coming from outside packages, though support is still somewhat limited. Any `pkgloader-`⟨*something*⟩`.sty` file can be loaded within the `pkgloader` area by passing ⟨*something*⟩ as a package option. The file `pkgloader-recommended.sty` is loaded this way by default. You can overwrite this by passing `recommended=false` as an option. For example:

```
1  \RequirePackage[recommended=false, my-better-rules]{pkgloader}
2     ⋮
3  \LoadPackagesNow
```

This area does not play by the recommended package loading rules, but uses the rules in `pkgloader-my-better-rules.sty` instead.

1.3 Package / Class Loading Requests

\usepackage
\RequirePackage
\RequirePackageW...s
\documentclass
\LoadClass
\LoadClassW...s

For requesting specific packages or classes inside the `pkgloader` area, just use the commands always used for this purpose: **\usepackage**, **\RequirePackage** and **\RequirePackageWithOptions**, as well as **\documentclass**, **\LoadClass** and **\LoadClassWithOptions**. Their syntax and effective semantics are the same as they have always been. Their effects are just delayed, reordered and perhaps modified by the active *package loading rules*.

1.4 Package Loading Rules

\Load

Each invocation if the **\Load** command sets up a rule about a class, package or packages, which are not necessarily ever loaded. These rules can come from any number of different sources. A central registry will be maintained together with `pkgloader` itself in the form of `pkgloader-recommended.sty`, specifying well-known conflicts

and resolutions. Individual package authors, however, can supply their own rules, as can document authors. Though ideally, for the average document author, things should 'just work'.

The **\Load** command expects the following syntax:

$$\langle load \rangle \quad ::= \quad \textbf{\textbackslash Load } \langle package \rangle \ \langle clause \rangle_1 \ \dots \ \langle clause \rangle_i$$

$\langle load \rangle$	::=	**\Load** $\langle package \rangle$ $\langle clause \rangle_1$ ... $\langle clause \rangle_i$
	|	**\Load** class $\langle package \rangle$ $\langle clause \rangle_1$ ... $\langle clause \rangle_i$
	|	**\Load error** $\langle clause \rangle_1$ ... $\langle clause \rangle_i$
$\langle package \rangle$	::=	**[** $\langle options \rangle$ **] {** $\langle pkg \rangle$ **} [** $\langle version \rangle$ **]**
$\langle clause \rangle$	::=	$\langle order \rangle$ | $\langle condition \rangle$ | $\langle reason \rangle$
$\langle order \rangle$	::=	**before {** $\langle pkg \rangle_1$ **,...,** $\langle pkg \rangle_j$ **}**
	|	**after {** $\langle pkg \rangle_1$ **,...,** $\langle pkg \rangle_j$ **}**
	|	**early**
	|	**late**
$\langle condition \rangle$	::=	**always** | **if loaded** | **if {** $\langle \phi \rangle$ **}**
$\langle \phi \rangle$	::=	$\langle pkg \rangle$ | $\langle \phi \rangle$ **&&** $\langle \phi \rangle$ | $\langle \phi \rangle$ **||** $\langle \phi \rangle$ | **!** $\langle \phi \rangle$ | **(** $\langle \phi \rangle$ **)**
$\langle reason \rangle$	::=	**because {** $\langle token{-}list \rangle$ **}**

$\langle pkg \rangle$ represents a package or class name. $\langle token{-}list \rangle$ should expand to a human-readable text without formatting.

if The $\langle condition \rangle$ clause determines under which package loading conditions any and all parts of a rule are invoked. Here is an example of the use of the $\langle condition \rangle$ clause:

```
1  \Load {res-ie-lst} if {inputenc && listings}
2  \Load {fixltx2e}    always
```

res-ie-lst (a fictional package built specifically to resolve the conflict between inputenc and listings) will be loaded if requested specifically, or if both inputenc and listings are loaded. The fixltx2e package is always loaded, as it was created to smooth over some mistakes in the LaTeX 2_ε core.

always The **always** keyword makes a rule unconditional. The **if loaded** directive makes
if loaded a rule conditional on its package already being loaded anyway. This can be used to order two packages only when they are being loaded by other means, and is actually the default behavior (in other words, **if loaded** really does nothing).

before The **before** and **after** keywords should be pretty straightforward. They can be
after used for things like:

```
1  \Load {xltxtra} after {graphicx}
```

which fixes the loading order between these two packages when they are both loaded.

early But the set of LaTeX packages is constantly growing, and it appears that some big
late packages should almost always be loaded early in the process, and others should almost always be loaded late. Therefore the **early** and **late** stages are provided as a fallback mechanism. If two packages are not related by an explicit application order, their loading order may still be decided by their relative stages: **early** before 'normal' before **late**. That way, conflicts may be avoided in a majority of cases. This is implemented with pkgloader-early and pkgloader-late stubs in the loading order graph.

The following example uses the $\langle order \rangle$ clause in addition to the $\langle condition \rangle$ clause:

```
1  \Load {res-ie-lst} if {inputenc && listings}
2                after {inputenc ,  listings}
3  \Load {fixltx2e} always early
```

An important observation about the loading order is that it might form a cycle when contradictory ordering rules are specified:

```
1  \Load {pkg1} after {pkg2}
2  \Load {pkg2} after {pkg1}
```

In practice this could happen if the authors of `pkg1` and `pkg2` independently discover a conflict, and both try to solve it by patching their code and having their own package be loaded last. `pkgloader` can provide a clear error message when this happens, allowing the two package authors to seek contact and collaborate on a solution.

error　　Now, about the **error** keyword. Initially all package combinations are valid. But if two packages are irredeemably incompatible, their combination can be made to trigger an error message by a command such as the following:

```
1  \Load error if {algorithms && pseudocode}
```

These two packages provide almost identical functionality and conflict on many command-names. It was generally agreed upon that they should never be loaded together. Document authors should simply choose one or the other.

because　　Finally, the ⟨*reason*⟩ clause can be used to supply a human-readable explanation of a rule. We finish the above examples by providing reasons:

```
1   \Load {res-ie-lst} if {inputenc && listings}
2                 after {inputenc ,  listings}
3       because {it allows the use of 1 byte unicode
4               characters in code listings}
5
6   \Load {fixltx2e} always early
7        because {it fixes some imperfections in LaTeX2e}
8
9   \Load error when {algorithms && pseudocode}
10       because {they provide almost identical functionality
11              and conflict on many command names}
```

In the future, reasons will be extracted automatically to generate documentation. For now, they are displayed with error messages related to the rule in question.

1.5 Status

So far, `pkgloader` seems to work as expected, but has not yet been tested as extensively as it should be. Therefore, bug-reports on the `pkgloader` issue tracker on Github would be most welcome. Also, lots more recommended package loading rules are needed.

I nonetheless decided to publish the package now, because I've been promising to do so for a while now:

<center>http://tex.stackexchange.com/questions/123174</center>

I hope that, with feedback and community collaboration, use of this package will become widespread and package authors will be able to work in a more modular fashion.

Future versions of `pkgloader` will be able to intelligently merge package options and to track packages loaded by other packages in order to better inform the user — perhaps even fix problems by carrying information into the next run through auxiliary files. But most of this will depend on feedback.

2 Implementation

We now show and explain the entire implementation from `pkgloader.sty`.

2.1 Package Info

First, the mandatory package meta-information:

```
1  \NeedsTeXFormat{LaTeX2e}
2  \RequirePackage{expl3}
3  \ProvidesExplPackage{pkgloader}{2018/04/29}{0.7.0}
4    {managing the options and loading order of LaTeX packages}
```

2.2 Required Packages

The following packages are required. Two standard `expl3`-related packages, one experimental package in `l3regex` and one user-contributed `expl3` package in `lt3graph`:

```
5  \RequirePackage{xparse}
6  \RequirePackage{l3keys2e}
7  \RequirePackage{l3regex}
8  \RequirePackage{lt3graph}
```

2.3 Package Code

We need two global data-structures. One to keep track of all packages that are known, one to keep track of the packages that are actually going to be loaded, and their order:

```
9   \prop_new:N  \g__pkgloader_known_pkg_prop
10  \graph_new:N \g__pkgloader_pkg_graph
```

We store pristine versions of the three package loading commands:

```
11  \cs_gset_eq:NN \__pkgloader_usepkg:wnw            \usepackage
12  \cs_gset_eq:NN \__pkgloader_RPkg:wnw             \RequirePackage
13  \cs_gset_eq:NN \__pkgloader_RPkgWithOptions:wnw \RequirePackageWithOptions
14  \cs_gset_eq:NN \__pkgloader_doccls:wnw           \documentclass
15  \cs_gset_eq:NN \__pkgloader_LCls:wnw             \LoadClass
16  \cs_gset_eq:NN \__pkgloader_LClsWithOptions:wnw \LoadClassWithOptions
```

And we define a command to clean up any and all commands that we change or introduce. It will be called when they are not needed anymore:

```
17  \tl_new:N \__pkgloader_cleanup_commands:
18  \tl_put_right:Nn \__pkgloader_cleanup_commands: {
19    \cs_gset_eq:NN \usepackage                  \__pkgloader_usepkg:wnw
20    \cs_gset_eq:NN \RequirePackage              \__pkgloader_RPkg:wnw
21    \cs_gset_eq:NN \RequirePackageWithOptions \__pkgloader_RPkgWithOptions:wnw
22    \cs_gset_eq:NN \documentclass               \__pkgloader_doccls:wnw
23    \cs_gset_eq:NN \LoadClass                   \__pkgloader_LCls:wnw
24    \cs_gset_eq:NN \LoadClassWithOptions        \__pkgloader_LClsWithOptions:wnw
```

```
25 }
```

This function globally registers a package loading rule, which can be created with either
the `\Load` command or any of the hijacked `\usepackage`-like commands:

```
26 \cs_new_protected:Nn \__pkgloader_register_rule:nnnnnnnnn {
27   % #1: package or class name
28   % #2: options
29   % #3: version
30   % #4: condition
31   % #5: compiled condition
32   % #6: packages to load before this one
33   % #7: packages to load after  this one
34   % #8: reason
35   % #9: command
```

If this package or class hasn't been seen before, register it and create a rule-counter for
it:

```
36   \prop_if_in:NnF \g__pkgloader_known_pkg_prop {#1} {
37     \prop_gput:Nnn \g__pkgloader_known_pkg_prop {#1} {}
38     \int_new:c {g__pkgloader_count_(#1)_int}
39   }
```

Increment the rule-counter:

```
40   \int_incr:c {g__pkgloader_count_(#1)_int}
```

Then, we set all properties of the

```
41   \tl_set:Nf \l_tmpa_tl {\int_use:c {g__pkgloader_count_(#1)_int}}
42   \tl_set:cn  {g__pkgloader_options_           (#1)_(\l_tmpa_tl)_tl} {#2}
43   \tl_set:cn  {g__pkgloader_version_           (#1)_(\l_tmpa_tl)_tl} {#3}
44   \tl_set:cn  {g__pkgloader_condition_         (#1)_(\l_tmpa_tl)_tl} {#4}
45   \tl_set:cn  {g__pkgloader_compiled_condition_(#1)_(\l_tmpa_tl)_tl} {#5}
46   \tl_set:cn  {g__pkgloader_predecessors_      (#1)_(\l_tmpa_tl)_tl} {#6}
47   \tl_set:cn  {g__pkgloader_successors_        (#1)_(\l_tmpa_tl)_tl} {#7}
48   \tl_set:cn  {g__pkgloader_reason_            (#1)_(\l_tmpa_tl)_tl} {#8}
49   \tl_set:cn  {g__pkgloader_command_           (#1)_(\l_tmpa_tl)_tl} {#9}
50   \bool_new:c {g__pkgloader_used_              (#1)_(\l_tmpa_tl)_bool}
51 }
```

These six macros are redefined to just register the loading information rather than load
the package or class immediately. The distinction between package and class is made by
prefixing the name with either `.sty` or `.cls` (which will be stripped off before the file is
actually loaded):

```
52 \RenewDocumentCommand {\usepackage} { o m o }
53   { \__pkgloader_usepackage_cmd:nnnnnn
54         {\usepackage} {#1} {#2.sty} {#3}
55         {pkgloader-cls-pkg.sty} {} }
56 \RenewDocumentCommand {\RequirePackage} { o m o }
57   { \__pkgloader_usepackage_cmd:nnnnnn
58         {\RequirePackage} {#1} {#2.sty} {#3}
```

```
59          {} {} }
60  \RenewDocumentCommand {\RequirePackageWithOptions} { o m o }
61    { \__pkgloader_usepackage_cmd:nnnnnn
62          {\RequirePackageWithOptions} {#1} {#2.sty} {#3}
63          {} {} }
64  \RenewDocumentCommand {\documentclass} { o m o }
65    { \__pkgloader_usepackage_cmd:nnnnnn
66          {\documentclass} {#1} {#2.cls} {#3}
67          {} {pkgloader-cls-pkg.sty} }
68  \RenewDocumentCommand {\LoadClass} { o m o }
69    { \__pkgloader_usepackage_cmd:nnnnnn
70          {\LoadClass} {#1} {#2.cls} {#3}
71          {} {pkgloader-cls-pkg.sty} }
72  \RenewDocumentCommand {\LoadClassWithOptions} { o m o }
73    { \__pkgloader_usepackage_cmd:nnnnnn
74          {\LoadClassWithOptions} {#1} {#2.cls} {#3}
75          {} {pkgloader-cls-pkg.sty} }
```

Storing this information is delegated to the `\__pkgloader_register_rule:nnnnnnnnn` function:

```
76  \cs_new:Nn \__pkgloader_usepackage_cmd:nnnnnn {
77    \__pkgloader_register_rule:nnnnnnnnn
78      {#3} {#2} {#4}                    % package name, options, version
79      {pkgloader-true.sty}              % condition
80      {\c_true_bool}                    % compiled condition
81      {#5} {#6}                         % predecessors, successors
82      {it~is~requested~by~the~author} % reason
83      {#1}                              % command
84  }
```

This is a sophisticated user-level command for manipulating package loading order and conditions. It has a 'non-standard' but convenient syntax, which scans for clauses rather than taking standard parameters:

```
85  \NewDocumentCommand {\Load} {} {
```

Initialize the variables used for storing given data:

```
86    \tl_clear:N        \l__pkgloader_load_extension_tl
87    \tl_clear:N        \l__pkgloader_load_options_tl
88    \tl_clear:N        \l__pkgloader_load_name_tl
89    \tl_clear:N        \l__pkgloader_load_version_tl
90    \clist_clear:N     \l__pkgloader_load_pred_clist
91    \clist_clear:N     \l__pkgloader_load_succ_clist
92    \tl_clear:N        \l__pkgloader_load_cond_tl
93    \tl_clear:N        \l__pkgloader_load_because_tl
94    \tl_clear:N        \l__pkgloader_load_cmd_tl
95    \bool_set_false:N \l__pkgloader_early_late_used_bool
```

Start scanning for input:

```
96    \__pkgloader_load_scan_ext_:w
```

```
97  }
```

This function checks if the `class` keyword is given.

```
98  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_ext_:w {
99    \peek_charcode_remove_ignore_spaces:NTF c {% % % class
100      \__pkgloader_load_scan_ext_c:w
101   }{ % % % % % % % % % % % % % % % % % % % % % % % % package details
102      \tl_set:Nn \l__pkgloader_load_extension_tl {.sty}
103      \tl_set:Nn \l__pkgloader_load_cmd_tl {\RequirePackage}
104      \__pkgloader_load_scan_pkg_:w
105    }
106  }
```

The `class` keyword indicates that this is a document class loading rule, rather than a package loading rule. We record this and then goes on to scan the details:

```
107  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_ext_c:w lass {
108    \tl_set:Nn \l__pkgloader_load_extension_tl {.cls}
109    \tl_set:Nn \l__pkgloader_load_cmd_tl {\LoadClass}
110    \clist_put_right:Nn \l__pkgloader_load_succ_clist {pkgloader-cls-pkg.sty}
111    \__pkgloader_load_scan_pkg_:w
112  }
```

The following function starts scanning for the name of the package central to this rule, as well as the options and minimum version proposed for it. It also checks if the `error` keyword is given.

```
113  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_pkg_:w {
114    \peek_charcode_remove_ignore_spaces:NTF e {% % % error
115      \__pkgloader_load_scan_pkg_e:w
116   }{\peek_charcode_remove_ignore_spaces:NTF [ {% % package options
117      \__pkgloader_load_scan_pkg_options:nw
118   }{ % % % % % % % % % % % % % % % % % % % % % % % package name
119      \__pkgloader_load_scan_pkg_:nw
120   }}
121  }
```

The `error` keyword can take the place of a package name, options and version. It is shorthand for the `pkgloader-error` file, and then jumps ahead to scanning clauses:

```
122  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_pkg_e:w rror {
123    \tl_set:Nn \l__pkgloader_load_name_tl {pkgloader-error.sty}
124    \__pkgloader_load_scan_clause_:w
125  }
```

This scans the options and goes ahead to scan the package name:

```
126  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_pkg_options:nw #1 ] {
127    \tl_set:Nn \l__pkgloader_load_options_tl {#1}
128    \__pkgloader_load_scan_pkg_:nw
129  }
```

This scans the package name (and adds the proper extension), peeks ahead for a minimum version, and otherwise goes on to scanning for clauses:

```
130  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_pkg_:nw #1 {
131    \tl_set:Nn \l__pkgloader_load_name_tl {#1}
132    \tl_put_right:NV
133        \l__pkgloader_load_name_tl
134        \l__pkgloader_load_extension_tl
135    \peek_charcode_remove_ignore_spaces:NTF [ {% % % % package version
136      \__pkgloader_load_scan_version:nw
137    }{ % % % % % % % % % % % % % % % % % % % % % % % % clauses
138      \__pkgloader_load_scan_clause_:w
139    }
140  }
```

This scans the version, and then goes ahead to scan for clauses:

```
141  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_version:nw #1 ] {
142    \tl_set:Nn \l__pkgloader_load_version_tl {#1}
143    \__pkgloader_load_scan_clause_:w
144  }
```

This is the start- and return-point used to scan for (additional) \Load clauses:

```
145  \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_:w {
146    \peek_charcode_remove_ignore_spaces:NTF a {
147      \peek_charcode_remove:NTF l { % % % % % % % % % always
148        \__pkgloader_load_scan_clause_al:w
149      }{\peek_charcode_remove:NTF f { % % % % % % % % % after
150        \__pkgloader_load_scan_clause_af:w
151      }{
152        \__pkgloader_load_end: a
153      }}
154    }{\peek_charcode_remove_ignore_spaces:NTF b { %
155      \peek_charcode_remove:NTF e {
156        \peek_charcode_remove:NTF c { % % % % % % % because
157          \__pkgloader_load_scan_clause_bec:w
158        }{\peek_charcode_remove:NTF f { % % % % % % before
159          \__pkgloader_load_scan_clause_bef:w
160        }{
161          \__pkgloader_load_end: be
162        }}
163      }{
164        \__pkgloader_load_end: b
165      }
166    }{\peek_charcode_remove_ignore_spaces:NTF e { % % % early
167      \__pkgloader_load_scan_clause_e:w
168    }{\peek_charcode_remove_ignore_spaces:NTF i { % % % if
169      \__pkgloader_load_scan_clause_i:w
```

```
170  }{\peek_charcode_remove_ignore_spaces:NTF l { % % % late
171     \__pkgloader_load_scan_clause_l:w
172  }{
173     \__pkgloader_load_end:
174  }}}}}
175 }
```

This processes the "always" clause, which loads this package conditional on the "pkgloader-true" package being loaded (which always is):

```
176 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_al:w ways {
177   \tl_put_right:Nn \l__pkgloader_load_cond_tl {~||~pkgloader-true}
178   \__pkgloader_load_scan_clause_:w
179 }
```

This processes the "after" clause:

```
180 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_af:w ter #1 {
181   \clist_map_inline:nn {#1} {
182     \clist_put_right:Nn \l__pkgloader_load_pred_clist {##1.sty}
183   }
184   \__pkgloader_load_scan_clause_:w
185 }
```

This processes the "because" clause:

```
186 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_bec:w ause #1 {
187   \tl_set:Nn \l__pkgloader_load_because_tl {#1}
188   \__pkgloader_load_scan_clause_:w
189 }
```

This processes the "before" clause:

```
190 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_bef:w ore #1 {
191   \clist_map_inline:nn {#1} {
192     \clist_put_right:Nn \l__pkgloader_load_succ_clist {##1.sty}
193   }
194   \__pkgloader_load_scan_clause_:w
195 }
```

This processes the "early" clause, which orders this package before the "pkgloader-early" stub:

```
196 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_e:w arly {
197   \bool_set_true:N \l__pkgloader_early_late_used_bool
198   \clist_put_right:Nn \l__pkgloader_load_succ_clist {pkgloader-early.sty}
199   \__pkgloader_load_scan_clause_:w
200 }
```

This processes the "if" clause, which may still be a manual condition or the "loaded" keyword:

```
201 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_i:w f {
202   \peek_charcode_remove_ignore_spaces:NTF l {
203     \__pkgloader_load_scan_clause_if_l:w
204   }{
205     \__pkgloader_load_scan_clause_if_:nw
206   }
207 }
```

This processes the "`if loaded`" clause, which uses this package being loaded as the condition for the rule being used:

```
208 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_if_l:w oaded {
209   \tl_put_right:Nn \l__pkgloader_load_cond_tl {~||~}
210   \tl_put_right:NV \l__pkgloader_load_cond_tl \l__pkgloader_load_name_tl
211   \__pkgloader_load_scan_clause_:w
212 }
```

This processes the "`if`" clause with a manual condition:

```
213 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_if_:nw #1 {
214   \tl_put_right:Nn \l__pkgloader_load_cond_tl {~||~#1}
215   \__pkgloader_load_scan_clause_:w
216 }
```

This processes the "`late`" clause, which orders this package after the "`pkgloader-late`" stub:

```
217 \cs_new_protected_nopar:Npn \__pkgloader_load_scan_clause_l:w ate {
218   \bool_set_true:N \l__pkgloader_early_late_used_bool
219   \clist_put_right:Nn \l__pkgloader_load_pred_clist {pkgloader-late.sty}
220   \__pkgloader_load_scan_clause_:w
221 }
```

This function processes the collected data and registers it:

```
222 \cs_new_protected_nopar:Nn \__pkgloader_load_end: {
```

We remove the leading " `||` " from the condition:

```
223     \tl_remove_once:Nn \l__pkgloader_load_cond_tl {~||~}
```

If no condition is given at all, the default is: "`if loaded`"

```
224     \tl_if_empty:NT \l__pkgloader_load_cond_tl
225       { \tl_set_eq:NN \l__pkgloader_load_cond_tl \l__pkgloader_load_name_tl }
```

We now take the condition and compile it to a `\bool_if:` kind of syntax. The original syntax is preserved to use in error messages and such:

```
226    \tl_set_eq:NN \l__pkgloader_load_compd_cond_tl \l__pkgloader_load_cond_tl
227    \regex_replace_all:nnN
228      { [^\&\|\(\)\!\s]+ }
229      { (\c{graph_if_vertex_exist_p:Nn}
230         \c{g__pkgloader_pkg_graph}\cB\{\0\.sty\cE\}) }
231      \l__pkgloader_load_compd_cond_tl
```

If no reason was given for this rule, it was obviously 'because of reasons':

```
232    \tl_if_empty:NT \l__pkgloader_load_because_tl
233      { \tl_set:Nn \l__pkgloader_load_because_tl {of~reasons} }
```

Having gathered and processed the data, the rule is registered:

```
234    \__pkgloader_register_rule:VVVVVVVVV
235      \l__pkgloader_load_name_tl
236      \l__pkgloader_load_options_tl
237      \l__pkgloader_load_version_tl
238      \l__pkgloader_load_cond_tl
239      \l__pkgloader_load_compd_cond_tl
240      \l__pkgloader_load_pred_clist
241      \l__pkgloader_load_succ_clist
242      \l__pkgloader_load_because_tl
243      \l__pkgloader_load_cmd_tl
```

```
244  }
245  \cs_generate_variant:Nn \tl_if_eq:nnF                          {VnF}
246  \cs_generate_variant:Nn \graph_gput_vertex:Nn                 {NV}
247  \cs_generate_variant:Nn \graph_gput_edge:Nnn                  {NnV,NVn}
248  \cs_generate_variant:Nn \seq_gput_right:Nn                    {NV}
249  \cs_generate_variant:Nn \__pkgloader_register_rule:nnnnnnnnn {VVVVVVVVV}
250  \tl_new:N    \l__pkgloader_load_extension_tl
251  \tl_new:N    \l__pkgloader_load_options_tl
252  \tl_new:N    \l__pkgloader_load_name_tl
253  \tl_new:N    \l__pkgloader_load_version_tl
254  \clist_new:N \l__pkgloader_load_pred_clist
255  \clist_new:N \l__pkgloader_load_succ_clist
256  \tl_new:N    \l__pkgloader_load_cond_tl
257  \tl_new:N    \l__pkgloader_load_because_tl
258  \tl_new:N    \l__pkgloader_load_cmd_tl
259  \bool_new:N  \l__pkgloader_early_late_used_bool
```

And here's the instruction to clean up the `\Load` command-name at the end:

```
260  \tl_put_right:Nn \__pkgloader_cleanup_commands:
261    { \cs_undefine:N \Load }
```

This function decides, based on all loaded rules and package requests, which packages, options and versions end up being loaded, and in which order.

```
262 \cs_new_protected:Nn \__pkgloader_select_packages: {
```

We first set up a graph to record accepted orderings:

```
263     \graph_clear:N \l__pkgloader_order_graph
```

We then start a loop that runs at least once, then repeats while additional package configurations are still being added to the set. Eventually the loop reaches a fixed point and terminates.

```
264     \bool_do_while:Nn \l__pkgloader_selection_changed_bool {
265       \bool_set_false:N \l__pkgloader_selection_changed_bool
```

Then first, for all possible package configurations (a nested loop, but not doubly indented because it feels like one loop):

```
266       \prop_map_inline:Nn \g__pkgloader_known_pkg_prop {%%%%%%%%%%%%    ##1
267       \int_step_inline:nncn {1} {1} {g__pkgloader_count_(##1)_int} {%% ####1
```

If the current configuration should be loaded but still isn't selected (nested conditional; but again, not indented):

```
268         \bool_if:cF {g__pkgloader_used_(##1)_(####1)_bool} {
269         \bool_if:vT {g__pkgloader_compiled_condition_(##1)_(####1)_tl} {
```

We mark the package configuration as being used:

```
270           \bool_set_true:c {g__pkgloader_used_(##1)_(####1)_bool}
```

We record the configuration in the main package graph, which maps each package to a clist of selected configurations:

```
271           \graph_get_vertex:NnNTF \g__pkgloader_pkg_graph {##1}
272                           \l__pkgloader_used_configs_tl {
273         \tl_if_empty:NTF \l__pkgloader_used_configs_tl
274           { \graph_gput_vertex:Nnf \g__pkgloader_pkg_graph {##1} {####1} }
275           { \graph_gput_vertex:Nnf \g__pkgloader_pkg_graph {##1}
276                           {\l__pkgloader_used_configs_tl, ####1} }
277         }   { \graph_gput_vertex:Nnf \g__pkgloader_pkg_graph {##1} {####1} }
```

In a separate graph, we record the associated (now activated) package loading orders. We don't do this in the main graph, because it may involve packages that are themselves not yet selected. These edges are later filtered and added to the main graph:

```
278           \graph_put_vertex:Nn \l__pkgloader_order_graph {##1}
279           \clist_map_inline:cn {g__pkgloader_predecessors_(##1)_(####1)_tl} {
280             \graph_put_vertex:Nn \l__pkgloader_order_graph {########1}
281             \graph_get_edge:NnnNTF \l__pkgloader_order_graph
282                   {########1} {##1} \l__pkgloader_used_configs_tl
283           { \graph_put_edge:Nnnn \l__pkgloader_order_graph
284                   {########1} {##1} {\l__pkgloader_used_configs_tl,####1} }
285           { \graph_put_edge:Nnnn \l__pkgloader_order_graph
286                   {########1} {##1}                              {####1} }
287           }
```

14

```
288         \clist_map_inline:cn {g__pkgloader_successors_(##1)_(####1)_tl} {
289           \graph_put_vertex:Nn \l__pkgloader_order_graph {########1}
290           \graph_get_edge:NnnNTF \l__pkgloader_order_graph
291                 {##1} {########1} \l__pkgloader_used_configs_tl
292             { \graph_put_edge:Nnnn \l__pkgloader_order_graph
293                   {##1} {########1} {\l__pkgloader_used_configs_tl,####1} }
294             { \graph_put_edge:Nnnn \l__pkgloader_order_graph
295                   {##1} {########1}                        {####1} }
296           }
```

We then mark the change, so a next iteration will be entered:

```
297           \bool_set_true:N \l__pkgloader_selection_changed_bool
```

```
298       }}
299     }}
300   }
```

We put the applicable proposed orderings into the graph of selected packages:

```
301   \graph_gput_edges_from:NN \g__pkgloader_pkg_graph \l__pkgloader_order_graph
```

If there is a cycle in the derived package loading order: ERROR

```
302   \graph_if_cyclic:NT \g__pkgloader_pkg_graph
303     {  \msg_fatal:nn {pkgloader} {cyclic-order}  }
```

Finally, we apply some default orderings where needed:

- If a package should not specifically go early or late, it goes inbetween; and

- if a package should not specifically go before a class, it goes after.

```
304   \graph_map_vertices_inline:Nn \g__pkgloader_pkg_graph {
305     \seq_if_in:NnF \g__pkgloader_system_packages_seq {##1} {
306       \graph_acyclic_if_path_exist:NnnF \g__pkgloader_pkg_graph
307           {##1} {pkgloader-early.sty} {
308         \graph_acyclic_if_path_exist:NnnF \g__pkgloader_pkg_graph
309             {pkgloader-late.sty} {##1} {
310           \graph_put_edge:Nnn \g__pkgloader_pkg_graph {pkgloader-early.sty} {##1}
311           \graph_put_edge:Nnn \g__pkgloader_pkg_graph {##1} {pkgloader-late.sty}
312       } }
313       \graph_acyclic_if_path_exist:NnnF \g__pkgloader_pkg_graph
314           {##1} {pkgloader-cls-pkg.sty} {
315         \graph_put_edge:Nnn \g__pkgloader_pkg_graph {pkgloader-cls-pkg.sty} {##1}
316   } } }
```

```
317 }
318 \cs_generate_variant:Nn \int_step_inline:nnnn   {nncn}
319 \cs_generate_variant:Nn \bool_if:nT             {vT}
320 \cs_generate_variant:Nn \withargs:nnn           {vvn}
321 \cs_generate_variant:Nn \graph_gput_vertex:Nnn {Nnf}
322 \graph_new:N \l__pkgloader_order_graph
323 \tl_new:N    \l__pkgloader_used_configs_tl
```

```
324 \bool_new:N  \l__pkgloader_selection_changed_bool
```

Now follows the user command to consolidate all package loading requests and do the 'right thing' (tm). Invoking this command ends the work of `pkgloader`.

```
325 \NewDocumentCommand {\LoadPackagesNow} {} {
```

We first select package configurations by their loading conditions:

```
326     \__pkgloader_select_packages:
```

Now to clean up after `pkgloader`, restoring and removing various command-names.

```
327     \__pkgloader_cleanup_commands:
```

Then, for all used packages, in topological order...

```
328     \graph_map_topological_order_inline:Nn \g__pkgloader_pkg_graph {
```

...load that package. Though note that this code is still quite incomplete, because it loads the first viable configuration. It should:

1. use the `WithOptions` version of the command if necessary,

2. allow custom merging schemes for options, and

3. use the latest required version.

```
329       \withargs:xn { \clist_item:nn{##2}{1} } {
330       \withargs:vvfvn {g__pkgloader_command_(##1)_(####1)_tl}
331                       {g__pkgloader_options_(##1)_(####1)_tl}
332                       {\__pkgloader_strip_extension:f{##1}}
333                       {g__pkgloader_version_(##1)_(####1)_tl} {
334         \IfValueTF {########2}
335           { \IfValueTF {########4}
336               { ########1 [########2] {########3} [########4] }
337               { ########1 [########2] {########3}             } }
338           { \IfValueTF {########4}
339               { ########1             {########3} [########4] }
340               { ########1             {########3}             } }
```
```
341       }
342     }
343   }
344 }
345 \cs_generate_variant:Nn \withargs:nn {xn}
346 \cs_generate_variant:Nn \withargs:nnnnn {vvfvn}
```

And it needs the following auxiliary function to strip filenames from their four character extension:

16

```
347 \cs_new:Npn \__pkgloader_strip_extension:f #1 {
348   \tl_reverse:f{
349     \tl_tail:f{\tl_tail:f{\tl_tail:f{\tl_tail:f{
350       \tl_reverse:n{#1}
351     }}}}
352   }
353 }
354 \cs_generate_variant:Nn \tl_reverse:n {f}
```

It's a bit clunky. Is there a substring function in `expl3` we could use that I don't know about?

And here's the instruction to clean up the `\LoadPackagesNow` command-name at the end:

```
355 \tl_put_right:Nn \__pkgloader_cleanup_commands:
356   { \cs_undefine:N \LoadPackagesNow }
```

```
357 \cs_gset_eq:NN \__pkgloader_begin_env:n \begin
358 \RenewDocumentCommand {\begin} {m} {
359   \tl_if_eq:nnT {#1} {document} {\LoadPackagesNow}
360   \__pkgloader_begin_env:n {#1}
361 }
362 \tl_put_right:Nn \__pkgloader_cleanup_commands:
363   { \cs_gset_eq:NN \begin \__pkgloader_begin_env:n }
```

Bootstrap `pkgloader` by inserting `pkgloader-true` in the graph directly, so all other packages can be inserted with rules, possibly using the `always` clause.

```
364 \graph_gput_vertex:Nn \g__pkgloader_pkg_graph {pkgloader-true.sty}
```

We keep a list of all pkgloader dummy packages:

```
365 \seq_new:N \g__pkgloader_system_packages_seq
366 \cs_generate_variant:Nn \seq_gset_from_clist:Nn {Nx}
367 \seq_gset_from_clist:Nx \g__pkgloader_system_packages_seq
368   {\tl_to_str:n
369     {pkgloader-true.sty,
370       pkgloader-false.sty,
371       pkgloader-early.sty,
372       pkgloader-late.sty,
373       pkgloader-cls-pkg.sty}}
```

We then register the core logical rules of `pkgloader`, regarding fundamental package 'stubs' like `pkgloader-false`, `pkgloader-error`, `pkgloader-early`, and so on.

```
374 \withargs:nn {of~the~mandatory~core~rules~of~pkgloader} {
375   \Load error if {pkgloader-false}              because {#1}
376   \Load {pkgloader-true}     always             because {#1}
377   \Load {pkgloader-early}    always             because {#1}
378   \Load {pkgloader-late}     always             because {#1}
```

17

```
379    \Load {pkgloader-cls-pkg} always                    because {#1}
380    \Load {pkgloader-early} before {pkgloader-late} because {#1}
381 }
```

We process the options passed to `pkgloader` as `.sty` files to be loaded before pkgloader does its thing. This should be used to define new `pkgloader` rules. Note, particularly, that any `\usepackage`-like command inside those `.sty` files is registered and processed by `pkgloader`; not loaded directly.

First, we define the functions used to handle an option.

```
382 \cs_new:Nn \__pkgloader_process_option:n
383    { \__pkgloader_process_option:nn {#1} {true} }
384 \cs_new:Nn \__pkgloader_process_option:nn
385    { \tl_if_eq:nnTF {#2} {true}
386          { \seq_put_right:Nn  \l__pkgloader_rule_packages_seq {#1} }
387          { \seq_remove_all:Nn \l__pkgloader_rule_packages_seq {#1} } }
```

The `recommended` rules are loaded unless explicitly turned off.

```
388 \seq_new:N \l__pkgloader_rule_packages_seq
389 \seq_put_right:Nn \l__pkgloader_rule_packages_seq {recommended}
```

Process the options to populate `\l__pkgloader_rule_packages_seq`.

```
390 \cs_generate_variant:Nn \keyval_parse:NNn {NNv}
391 \keyval_parse:NNv
392    \__pkgloader_process_option:n
393    \__pkgloader_process_option:nn
394    {opt@pkgloader.sty}
395 \seq_remove_duplicates:N \l__pkgloader_rule_packages_seq
```

Actually load the `.sty` files in `\l__pkgloader_rule_packages_seq`. Note that the actual file needs the `pkgloader-` prefix.

```
396 \seq_map_inline:Nn \l__pkgloader_rule_packages_seq
397    { \__pkgloader_RPkg:wnw {pkgloader-#1} }
```

Finally, we make a show of using the proper macros for LaTeX's benefit. If we don't, a LaTeX error is issued.

```
398 \DeclareOption*{}
399 \ProcessOptions\relax
```

Finally, here are the error messages this package can generate. First a simple error for cycles, which should be improved to show the cause of the cycle.

```
400 \msg_new:nnn {pkgloader} {cyclic-order}
401    {  There~is~a~cycle~in~the~requested~package~loading~order.  }
```

And the following is the error reported for certain package combinations that have been forbidden through an `error` rule.

```
402  \msg_new:nnnn {pkgloader} {illegal-combination}
403    {  A~combination~of~packages~fitting~the~following~condition~
404       was~requested:
405       \\\\\ \ \ \ #1\\\\
406       This~is~an~error~because~#2.  }
407    {  A~pkgloader~rule~was~requested~that~prohibits~the~logical~
408       combination\\above~for~the~specified~reason.~It~is~probably~
409       a~good~reason.  }
```

Change History

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

20

21