The **typed-checklist** package*

Richard Grewe
r-g+tex@posteo.net
2022-05-29

**Abstract**

The main goal of the **typed-checklist** package is to provide means for typesetting checklists in a way that stipulates users to explicitly distinguish checklists for goals, for tasks, for artifacts, and for milestones – i.e., the type of checklist entries. The intention behind this is that a user of the package is coerced to think about what kind of entries he/she adds to the checklist. This shall yield a clearer result and, in the long run, help with training to distinguish entries of different types.

1 **Motivation and Disambiguation**

The development of this package was driven with two goals in mind:

1. having a package with which one can easily typeset checklists and in a way that separates content from layout;

2. having a thinking tool that helps distinguishing between goals and tasks.

The first goal felt natural to me since from time to time I manage checklists in **LaTeX** documents, mostly because I like it when the result looks typeset nicely. The second goal arose from an observation about some of my own checklists as well as checklists created by others: Quite frequently, the checklists mixed goals and tasks or had goals formulated as tasks and vice versa. As a consequence, the checklists were formulated unnecessarily unclear and were more difficult to understand by others.

This package approaches particularly the second goal by providing checklists with a *type*. A checklist of a particular type shall then only contain entries of this type.

While the package allows one to define custom checklist types (see Section 4), it comes with four basic types: Artifact, Goal, Milestone, and Task. In this documentation, the terms “artifact”, “goal”, “milestone”, and “task” will be used along the lines of the following definitions (highlights added):

---

*This document corresponds to **typed-checklist** v2.1, dated 2022/05/28. The package is available online at [http://www.ctan.org/pkg/typed-checklist](http://www.ctan.org/pkg/typed-checklist) and [https://github.com/Ri-Ga/typed-checklist](https://github.com/Ri-Ga/typed-checklist).*
artifact:  – “An object made or shaped by human hand.” (Wiktionary)
goal:  – “An observable and measurable end result having one or more objectives to be achieved within a more or less fixed timeframe.” (BusinessDictionary.com)
        – “the end toward which effort is directed” (Merriam-Webster)
        – “The object of a person’s ambition or effort; an aim or desired result” (Oxford Dictionaries)
        – “A result that one is attempting to achieve.” (Wiktionary)
milestone:  – “An important event [...] in the life of some project” (Wiktionary)
task:  – “a usually assigned piece of work often to be finished within a certain time” (Merriam-Webster)
        – “A piece of work done as part of one’s duties.” (Wiktionary)

We could connect the four terms as follows. Typically, the “piece of work” that constitutes a task is performed for achieving some goal. One can also say that a goal serves as a reference point for why and how one should perform certain tasks. A goal can be that a particular artifact or set of artifacts is available at some point in time. A milestone is a group of goals whose achievement is of importance for something bigger. These connections suggest that nesting different types of checklists is reasonable – and it is supported by the typed-checklist package.

2  Recommendations for Structuring Checklists

The typed-checklist package allows checklists of different types as well as of identical types to be nested. That is, within a checklist, another checklist can be placed. The following list discusses some combinations of nested checklist types and provides some recommendations of what types could be nested for particular purposes and what types should better not be nested.

1. tasks in goals ................................................................. ✓
   This nesting combination could be used for listing tasks whose accomplishment would lead to the satisfaction of the superordinated goal.

2. goals in goals ................................................................. ✓
   This nesting combination could be used for explicitly listing sub-goals (and sub-sub-goals and ...) to a goal. That is, using this nesting combination you can express the result of breaking down goals into sub-goals. Used reasonably, this nesting should be used in a way that the sub-goals, when achieved, yield the superordinated goal to be achieved (at least with high probability and/or to a significant extent).

3. tasks in tasks ................................................................. ✓
   This nesting combination could be used for listing all sub-tasks to a task. That is, using this nesting combination you can express the result of breaking down tasks into sub-tasks.
4. **goals in milestones** 
This nesting combination could be used for listing all goals that must be achieved, at a particular date, for calling a milestone achieved.

5. **artifacts in milestones** 
This nesting combination could be used for listing all artifacts that must exist, at a particular date, for calling a milestone achieved.

6. **goals in tasks** 
This nesting lacks a clearly recognizable meaning. The use of this kind of nesting might be an indicator for a misunderstanding of goals or tasks, or it might be the result of too vague formulations of goals or tasks that do not reveal that something is wrong in the planning.

7. **milestones in milestones** 
A milestone, as cited, is an important event. Having sub-milestones would blur the notion of important events by introducing multiple levels of important events. Instead of nesting milestones, one could nest goals or artifacts in milestones to express intermediate stages of a milestone.

### 3 Basic Usage

The following example demonstrates a basic use of the package.

```latex
\documentclass{article}
\usepackage{typed-checklist}
\begin{document}
\begin{CheckList}{Goal}
  \Goal{open}{I have a trendy haircut}
  \begin{CheckList}{Task}
    \Task{done}{find a hairdresser}
    \Task{started}{make an appointment}
    \Task{open}{go to the hairdresser}
  \end{CheckList}
  \Goal{achieved}{I have a typed checklist}
\end{CheckList}
\end{document}
```

The example contains a checklist for goals and the first goal contains a checklist for tasks. Checklist entries have a status and a description. In the typeset result, the checklist type is reflected by a basic symbol (an empty circle for a goal and an empty box for a task) that is decorated depending on the status (e.g., with a check mark). The entry’s description is shown next to the symbol.

#### 3.1 Checklists

Checklists are created via the CheckList environment. The \texttt{(type)} parameter determines the type of all checklist entries in the environment. The \texttt{typed-checklist}
package comes with four predefined types: Goal, Task, Artifact, and Milestone. Each of the types comes with a macro of the same name as the type. With this macro, the entries of the checklist can be created.

The \langle options \rangle can be a comma-separated list of \langle key \rangle=\langle value \rangle pairs. Table 1 on page 8 shows the keys and possible values that can be set.

Defaults for checklist options can also be specified globally, either through package options or through the \CheckListSet macro.

\CheckListSet\{\langle options-list \rangle\}
This macro takes a comma-separated \langle options \rangle list and sets these options for all subsequent checklists.

A checklist can be viewed as a list of entries (even if the layout is actually tabular). The macros for creating the entries are described next.

\subsection{Checklist Entries}

\Goal\{\langle options \rangle\}\{\langle status \rangle\}\{\langle description \rangle\}
Inside a checklist of type Goal, the \Goal macro specifies a goal. Every goal comes at least with a \langle description \rangle and a \langle status \rangle. The \langle description \rangle can, technically, be anything that is displayable in the given checklist layout. However, for the purpose of a meaningful checklist, the \langle description \rangle should be a clear description of a goal in a full sentence\footnote{Incomplete sentences tend to be less clear.}. The \langle status \rangle parameter selects the most recent known status of the goal. This parameter can assume any of the following values\footnote{See Section 4.1 to find out how custom states can be defined}:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>achieved</td>
<td>This value specifies that the goal has been achieved. Depending on how the \langle description \rangle was formulated, this might mean that in the respective situation the \langle description \rangle is a true statement.</td>
</tr>
<tr>
<td>dropped</td>
<td>This value specifies that the goal was a goal once but is no longer a goal that shall be pursued. This value allows one to preserve historical information about a checklist.</td>
</tr>
<tr>
<td>unclear</td>
<td>This value specifies that the goal somehow exists but is not yet clear enough to those who pursue the goal (or: who typeset the checklist) for actually pursuing the goal.</td>
</tr>
<tr>
<td>open</td>
<td>This value specifies the negation of all aforementioned values. That is, the goal is clear but neither achieved yet nor dropped.</td>
</tr>
</tbody>
</table>

The \langle options \rangle allow one to specify further details about the goal. The \langle options \rangle must be a possibly empty, comma-separated list of \langle key \rangle=\langle value \rangle pairs. The \langle key \rangle must be one of the following values\footnote{See Section 4.2 to find out how custom \langle key \rangles can be defined}:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>who</td>
<td>This option declares who is responsible for making sure the checklist entry is addressed. Remember to put the value in curly braces if it contains commas.</td>
</tr>
</tbody>
</table>

\footnote{Incomplete sentences tend to be less clear.}
This option declares a deadline for the checklist entry, i.e., a date until which the entry must be addressed at latest. The format for specifying deadlines is determined by the checklist options `input-dates` and `strict-dates`.

This option declares a label name for the checklist entry. This is analogous to the \label macro of \LaTeX. The entry’s label is displayed next to the entry. A reference to a labeled checklist entry can be made using the \ref macro of \LaTeX.

\Task [(options)]{(status)}{(description)}

Inside a checklist of type Task, the \Task macro specifies a task. Every task comes at least with a \textit{(description)} and a \textit{(status)}. The \textit{(description)} can, technically, be anything that is displayable in the given checklist layout. However, for the purpose of a meaningful checklist, the \textit{(description)} should be a clear description of a task in a full sentence, possibly in imperative form. The \textit{(options)} parameter can be set as documented for the \Goal macro on page 5. The \textit{(status)} parameter selects the most recent known status of the task. This parameter can assume any of the following values:

- open: This value specifies that the task is still planned but has not yet been started.
- dropped: This value specifies that the task was originally planned but is no longer part of the plan.
- unclear: This value specifies that the task itself or its current status is unclear.
- started: This value specifies that someone has started to perform the task, but has not finished yet.
- done: This value specifies that someone has accomplished the task. Depending on the clarity and level of detail of the \textit{(description)}, whether accomplishing the task yielded a meaningful outcome might be more or less subjective to the person who accomplished the task.

\Artifact [(options)]{(status)}{(description)}

Inside a checklist of type Artifact, the \Artifact macro specifies an artifact. Every artifact comes at least with a \textit{(description)} and a \textit{(status)}. The \textit{(description)} can, technically, be anything that is displayable in the given checklist layout. However, for the purpose of a meaningful checklist, the \textit{(description)} should be a clear identification of the artifact and its required attributes. The \textit{(status)} parameter selects the most recent known status of the artifact. This parameter can assume any of the following values:

- missing: This value specifies that the artifact is missing yet.
- dropped: This value specifies that the artifact was originally planned but is no longer part of the plan.
unclear  This value specifies that the artifact itself or its current status is unclear.
incomplete  This value specifies that some non-negligible parts of the artifact exist but the artifact does not yet exist in its final form.
available  This value specifies that the artifact exists and available.

\Milestone\{⟨options⟩\}\{⟨status⟩\}\{⟨description⟩\}

Inside a checklist of type \Milestone, the \Milestone macro specifies a milestone. Every milestone comes at least with a ⟨description⟩ and a ⟨status⟩. The ⟨description⟩ can, technically, be anything that is displayable in the given checklist layout. However, for the purpose of a meaningful checklist, the ⟨description⟩ should be a clear identification of what has to exist or must have been fulfilled. The ⟨status⟩ parameter selects the most recent known status of the milestone. This parameter can assume any of the following values:

open  This value specifies that the milestone has not yet been achieved.
achieved  This value specifies that the milestone has been achieved.

3.3 Comprehensive Example

The example in Listing 1 on page 9 shows the use of nested checklists and the use of various checklist and entry options. The example deliberately mixes different date formats for the sake of demonstration, but this should normally be avoided as it reduces legibility.

4 Customized Checklists

The \texttt{typed-checklist} package comes with a set of layouts, checklist types, checklist entry states, and checklist entry options. These together shall provide everything needed for typesetting even checklists with complex structures. When the default is not enough, you can use the macros described in this section for creating your own layouts, types, states, and options.

4.1 Defining Checklist Types and Entry States

\CheckListAddType\{(type)\}\{⟨symbol⟩\}

Using this macro, you can add a new checklist type. The name of the type, i.e., the name that can be used as argument to the CheckList environment, is specified by ⟨type⟩. The basic symbol of entries belonging to this checklist type will be ⟨symbol⟩ (e.g., an empty box or circle). All status-symbols (see Section 4.1) are drawn on top of ⟨symbol⟩. Note that the \texttt{typed-checklist} package uses this macro also for creating each of the four default checklist types.

\CheckListAddStatus\{(types)\}\{⟨status⟩\}\{⟨isclosed⟩\}\{⟨symbol⟩\}
<table>
<thead>
<tr>
<th>Key</th>
<th>Description and Possible Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout</td>
<td>This selects the default checklist layout. Allowed values are all known layout names, including the predefined ‘list’, ‘table’, ‘hidden’. In list layout, each entry is a list item. In table layout, each entry is a row and the checklist is a table (see Section 6.2 for how to change which table environment is used). The hidden layout does not display the checklist and its entries.</td>
<td>list</td>
</tr>
<tr>
<td>input-dates</td>
<td>This option specifies the format of deadlines that checklist entries expect. Allowed values are ‘d.m.y’, ‘m/d/y’, and ‘y-m-d’ – with the intuitive meaning.</td>
<td>d.m.y</td>
</tr>
<tr>
<td>output-dates</td>
<td>This option specifies the format in which deadline dates are displayed. Allowed values are: ‘d.m.y’, ‘m/d/y’, ‘y-m-d’: These format dates in the indicated order of day, month, and year. ‘d.m.yy’, ‘m/d/yy’, ‘yy-m-d’: These are analogous to their counterparts with a single ‘y’, but use a two-digit display of the year (i.e., the century is stripped away). ‘d.m.’, ‘m/d’, ‘m-d’: These format dates in the indicated order, showing only month and day of the month. ‘same’: With same, deadlines are output in the same format in which they are specified. ‘datetime’: With datetime, the datetime2 package is used for displaying deadlines. The package must be loaded manually.</td>
<td>same</td>
</tr>
<tr>
<td>strict-dates</td>
<td>This option specifies whether deadlines must adhere to the input date format (as specified via the input-dates key) or can deviate. Allowed values are ‘true’ and ‘false’.</td>
<td>false</td>
</tr>
</tbody>
</table>

Table 1: Options for CheckList environments (and \CheckListSet)
Y1K problems are resolved.

Y2K problems are resolved.                      (John)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
<th>Who</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>(Task i) Repair all programs</td>
<td>John</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Just turn off all computers, if Task i fails</td>
<td>Mankind</td>
<td>December 31, 1999</td>
</tr>
</tbody>
</table>

Y65K problems are resolved.

(Task ii) Build Y65K-proof time machine.

Use time machine from Task ii if problem persists.

Listing 1: Comprehensive checklist example

```latex
\DTMsetregional% remember \usepackage{datetime2}
\CheckListSet{strict-dates,input-dates=d.m.y,output-dates=same}
\begin{CheckList}{Goal}
  \Goal[deadline=31.12.999]{achieved}{Y1K problems are resolved.}
  \Goal[who=John,deadline=31.12.1999]{open}{Y2K problems are resolved.}
\end{CheckList}
\begin{CheckList}{Task}
  \Task[who=John,label=Fix1]{started}{Repair all programs}
  \Task[who=Mankind,deadline=31.12.1999]{open}{Just turn off all computers, if \ref{Fix1} fails}
\end{CheckList}
\begin{CheckList}{Goal}
  \Goal[deadline=31.12.65535]{unclear}{Y65K problems are resolved.}
\end{CheckList}
\begin{CheckList}{Task}
  \Task[deadline=$\approx 2500AD$,label=TM]{open}{Build Y65K-proof time machine.}
  \Task[deadline=31.12.65535]{open}{Use time machine from \ref{TM} if problem persists.}
\end{CheckList}
\end{CheckList}
```
Using this macro, you can add a new checklist entry status for selected checklist types. The name of the status to define is specified by the \textit{status} argument. The checklist types to which the status is added, are provided by the \textit{types} argument, a comma-separated list. The \textit{symbol} is \LaTeX{} code of a symbol that is put on top of the checklist type's symbol. The \textit{isclosed} parameter must be one of \texttt{true} or \texttt{false}. A value of \texttt{true} indicates that the status of the entry corresponds to the entry being closed. This particularly means that no warning will be shown if the deadline of an entry with this status is passed. A value of \texttt{false} for \textit{isclosed} indicates that the \textit{status} corresponds to the entry not yet being closed. Note that the \texttt{typed-checklist} package uses this macro also for creating the provided states of the four default checklist types.

\textbf{Example}   The following example shows how to define a ‘bug’ type.

\begin{example}
\begin{itemize}
\item program crashes when started after 31.12.65535
\item progress bar flawed when duration above 136.2 years . . . . . . . . . . . . \texttt{(C++ Team)}
\item help screen crashes when F1 is pressed . . . . . . . . . . . . . . . . . \texttt{(Test Team)}
\item fancy splash screen missing
\end{itemize}

\begin{verbatim}
\CheckListAddType{Bug}{\textcolor{lightgray}{\FourStar}}
\CheckListAddStatus{Bug}{new}{false}{\textcolor{red}{\FourStar}}
\CheckListAddStatus{Bug}{assigned}{false}{\textcolor{yellow!75!red}{\FourStar}}
\CheckListAddStatus{Bug}{resolved}{true}{\textcolor{green}{\FourStar}}
\CheckListAddStatus{Bug}{closed}{true}{\Checkmark}
\begin{CheckList}{Bug}
    \Bug{new}{program crashes when started after 31.12.65535}
    \Bug[who=C++ Team]{assigned}{progress bar flawed when duration above 136.2 years}
    \Bug[who=Test Team]{resolved}{help screen crashes when F1 is pressed}
    \Bug{closed}{fancy splash screen missing}
\end{CheckList}
\end{verbatim}
\end{example}

\section{Defining Checklist Layouts}

Using this macro, you can add a new checklist layout. The \texttt{(begin)} and \texttt{(end)} part is similar to a \LaTeX{} \texttt{\newenvironment}. The \texttt{(fields)} must be a comma-separated list of field names. A field name can be one of the following:

1. the name of an entry property (e.g., ‘status’, ‘description’, ‘deadline’, or ‘who’),
2. the concatenation of multiple entry properties, separated by a ‘+’ (e.g., ‘deadline+status’), or
3. a fresh name that does not correspond to an entry property.

When one or multiple entry properties are referenced in a field name (cases 1 and 2), then the \texttt{(code)} argument to \texttt{\CheckListDefineFieldFormat} gets the properties’ values as arguments when invoked.

\begin{verbatim}
\CheckListDeclareLayout{{(name)}}{{(fields)}}{(begin)}{(end)}
\CheckListDefineFieldFormat{{(layout)}}{{(field)}}{{(code)}}
\end{verbatim}
After the new type has been added, for each field in the comma-separated \langle fields \rangle, this macro must be used to define how a field is formatted. The \langle code \rangle can take one or more arguments. If the \langle field \rangle does not contain a ‘+’, the \langle code \rangle can take one argument, through which the value of the respective entry property is passed to \langle code \rangle. If \langle field \rangle concatenates multiple property names with a ‘+’, then the number of arguments equals the number of names in \langle field \rangle and the properties are passed in the given order.

\CheckListExtendLayout\{(name)\}\{(base)\}\{(fields)\}

Using this macro, you can extend an existing checklist layout. Afterwards, the layout \langle name \rangle is available. This layout takes the \langle begin \rangle and \langle end \rangle code from the \langle base \rangle layout. Moreover, all fields defined by the \langle base \rangle layout can be used in the \langle fields \rangle parameter of the new layout. However, additional fields can be defined and the format of the fields for the new layout can be overwritten via \CheckListDefineFieldFormat.

**Auxiliary Macros**  
The following macros can be used in the definition of field formats.

\CheckListStatusSymbol\{(status)\}

The macro expands to the defined symbol for the given \langle status \rangle, i.e., the overlay between the checklist type’s base symbol and the entry status’ symbol.

\CheckListSigned[\langle core \rangle]\{(text)\}

The macro displays \langle text \rangle in a right-aligned fashion with a dotted leader to \langle text \rangle. This is similar to the display of page numbers in some table of content formats. The display takes place only if \langle text \rangle is non-empty. If \langle core \rangle is given, \langle core \rangle is instead used in the emptiness check.

\CheckListDefaultLabel\{(label)\}

This macro sets \langle label \rangle as the label for the current entry, based on the default checklist counter. It corresponds to a \refstepcounter on the checklist counter and a subsequent \label\{(label)\}.

\CheckListDisplayDeadline\{(status)\}\{(deadline)\}

This macro displays \langle deadline \rangle depending on the given entry’s \langle status \rangle and the current date. Internally, for highlighting the deadline, the following macro is used, which can be redefined with \renewcommand to change the deadline highlighting.

\CheckListHighlightDeadline\{(closed?)\}\{(passed?)\}\{(deadline)\}

This macro formats \langle deadline \rangle depending on whether the corresponding checklist entry is \langle closed? \rangle (true or false) and whether \langle deadline \rangle has already \langle passed? \rangle (true or false).

**Example**  
The following example shows how to define an alternative list format.
1. Y1K problems are resolved. .......................... 31.12.999
   (a) John: Repair all programs
   (b) Mankind: Just turn off all computers, if Task 2a fails ....... 31.12.1999

\begin{CheckList}[layout=enumlist]{Goal}
  \Goal[deadline=31.12.999]{achieved}{Y1K problems are resolved.}
  \Goal[who=John,deadline=31.12.1999]{open}{Y2K problems are resolved.}
\end{CheckList}

\begin{CheckList}{Task}
  \Task[who=John,label=Fix1]{started}{Repair all programs}
  \Task[who=Mankind,deadline=31.12.1999]{open}{Just turn off all computers, if Task~\ref{Fix1} fails}
\end{CheckList}

4.3 Adding Entry Options

Checklist entries can be augmented by more than the default fields. Values for these additional fields can be specified as entry options.

\CheckListAddEntryOption{{\textit{name}}}{{\textit{default}}}

This macro introduces a new entry option named \textit{name} and with the given \textit{default} value. The newly introduced option can then be provided to a checklist entry in the same way as the pre-defined options “who” and “label”.

When an entry option is defined, by default it is not displayed. Hence, when introducing a new entry option, one should consider defining a new checklist layout that makes use of the entry option.

The following example shows how to extend a layout for incorporating a custom-defined priority field.

11


5 Filtering Checklists

Filtering out certain checklist entries based on their properties can help keeping the focus on the relevant entries. For this purpose, typed-checklist allows one to specify filtering code.

5.1 Setting Basic Filters

\begin{CheckListFilterClosed}\{\textit{types}\}\end{CheckListFilterClosed}

This macro sets up a filter that hides all checklist entries whose status is closed. Through the optional \textit{types} argument, a comma-separated list of checklist types can be specified to which the filter shall be applied. By default, the filter is applied to all defined checklist types.

\begin{CheckListFilterClosed}\{\textit{types}\}\end{CheckListFilterClosed}

\begin{CheckListFilterValue}\{\textit{field}\}\{\textit{value}\}\end{CheckListFilterValue}

This macro sets up a filter that hides all checklist entries whose \textit{field} has a value that is unequal \textit{value}. 

\begin{CheckListFilterValue}\{\textit{field}\}\{\textit{value}\}\end{CheckListFilterValue}
This macro sets up a filter that filters out checklist entries by their deadline. Only those entries are preserved whose deadline is before (if \langle comp \rangle equals ‘\(<\)’), equal (if \langle comp \rangle equals ‘\(=\)’), or after (if \langle comp \rangle equals ‘\(>\)’) the given \langle date \rangle. The \langle date \rangle must be in the format selected for input dates (see the input-dates option). If \langle filter-inv \rangle is true, then checklist entries whose deadline does not obey the format for input dates are filtered out. Otherwise, if \langle filter-inv \rangle is false, these checklist entries are not filtered out.

\begin{checklist}
\Task[who=John, deadline=09.11.1989]{open}{John’s task}
\Task[who=Mary, deadline=01.01.2019]{open}{Mary’s task}
\Task[deadline=TBD]{open}{Other task (first time)}
\\end{checklist}

5.2 Combining and Resetting Filters

When multiple filter macros are used, the filters are applied one after another to each checklist entry until a filter filters out the entry. Consequentially, all applied filters are combined conjunctively, i.e., only those checklist entries are displayed that satisfy all filters.

When two filters are set up that affect the exact same fields of checklist entries (of the same type), then only the last of these filters becomes effective. The following example demonstrates this as well as the conjunction of filters.

\begin{checklist}
\Task[who=John, deadline=09.11.1989]{open}{John’s task}
\Task[who=Mary, deadline=01.01.2019]{open}{Mary’s task}
\Task[deadline=TBD]{open}{Other task (first time)}
\\end{checklist}
5.3 The Generic Filter Interface

Filters can also be set up programmatically.

```
\CheckListSetFilter[(types)]{(fields)}{(filter-code)}
```

This macro sets up the \textit{(filter-code)} for a set of \textit{(fields)}. The \textit{(fields)} must be given as a ‘+’-separated list of field names, e.g., “status+who”. The \textit{(filter-code)} may contain as many positional parameters (#1, \ldots) as there are fields names in \textit{(fields)}. When a checklist entry is about to be displayed, the \textit{(filter-code)} is evaluated, obtaining as arguments the entry’s field values. By default (without any filter set up), all entries are displayed. To disable the display of an entry, the \textit{(filter-code)} can use \texttt{\togglefalse{display}}. If \textit{(types)} are given (as a comma-separated list), then the \textit{(filter-code)} is applied only to checklists of a type in the list.

Examples for how to use the macro can be found in the implementation, e.g., of the macros \texttt{\CheckListFilterClosed} and \texttt{\CheckListFilterValue}.

6 Checklists and Other Packages

6.1 asciilist

The \texttt{typed-checklist} package can be combined with the \texttt{asciilist} package in the sense that a checklist can be defined within an AsciList environment. The \texttt{typed-checklist} package provides a syntax for this when the package is loaded with the
with\texttt{withAsciiList=true} option. The syntax is illustrated with the following snippet, a transformed version of the example in Section 3.3:

\begin{AsciiList}[\texttt{GoalList,TaskList}]{-,*}
- \texttt{achieved[deadline=31.12.999]}: No Y1K problems
- \texttt{open[who=John,deadline=31.12.1999]}: No Y2K problems
* \texttt{started[who=John,label=Fix2]}: Repair programs
* \texttt{open[who=Mankind,deadline=31.12.1999]}: Just turn off all computers, if Task \texttt{iii} fails
- \texttt{unclear[deadline=31.12.9999]}: No Y10K problems
\end{AsciiList}

For each checklist type \texttt{(type)} (added by \CheckListAddType), an \texttt{AsciiList} environment \texttt{(type)List} is automatically created.

Note that currently, a checklist entry in an \texttt{AsciiList} environment must fit into a single line or each except for the last line is ended with a percent char (as in the above example). Note also that the table layout does not work within an \texttt{AsciiList} environment.

### 6.2 Table Packages

The table layout by default uses the \texttt{xltabular} package for layouting the tables. The default can be changed through the \texttt{tablepkg} package option. The following values are available:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ltablex}</td>
<td>This option uses the \texttt{ltablex} package.</td>
</tr>
<tr>
<td>\texttt{tabularx}</td>
<td>This option uses the \texttt{tabularx} package from the L\LaTeX{} core. When using this table type, keep in mind that \texttt{tabularx} tables must fit onto a single page.</td>
</tr>
<tr>
<td>\texttt{xltabular}</td>
<td>This option uses the \texttt{xltabular} package, a successor of \texttt{ltablex}.</td>
</tr>
</tbody>
</table>

### 7 Related Packages

The following L\LaTeX{} packages provide related functionalities to the \texttt{typed-checklist} package.

\texttt{todo}: The package allows for typesetting “to-dos”, i.e., tasks in some sense, in a simple way with customizable display. The three main conceptual differences between \texttt{todo} and \texttt{typed-checklist} are:
1. todo does not distinguish between different types (such as goals and tasks);
2. todo does not allow one to provide a status for a to-do and rather assumes that done to-dos are simply removed from the document;
3. todo aims at specifying tasks for the document into which the to-dos are placed, while typed-checklist aims at typesetting checklists whose entries are for more general kinds of projects.

easy-todo:
The package is similar in spirit to the todo package and shares the main differences to the typed-checklist package.

todonotes:
The package is similar in spirit to todo and easy-todo, but provides more formatting options for the to-dos.

pgfgantt:
The package allows one to create Gantt charts, i.e., graphical displays of activities and milestones with a focus on time frames. The package allows one to structure the activities into groups. In that sense, there are certain similarities between the packages. The main conceptual difference to typed-checklist is the form of presentation (time-centric Gantt chart vs. text-centric lists).

8 Limitations and Future Work

- In twoside documents, deadlines are currently displayed in the left margin on even pages. The default layout (list) does not look good then. This should be repaired. The same problem is with checklist entry labels, which are displayed on the other side.
- In deadlines, the full year (including century) must be provided for the colored highlighting to work. Future versions could check for a two-digit year and automatically prepend “20” for the century.
- The package automatically adds the pre-defined checklist types and states, which might have two draws for some users: firstly, this adds a dependency on symbol packages, which might not work well together with some fonts; secondly, some users might prefer other definitions of the standard checklist types. To improve the situation, the package could offer an option for disabling the definition of the standard checklist types. Concerning the symbols packages, typed-checklist could also reduce the set of used packages or even draw all symbols itself.
- The package displays checklist entries in the ordering in which they are listed in the LATEX sources. Automatic sorting of checklist entries, for instance by deadline or future fields like priority/importance, might make the package even more useful for bigger checklists. The implementation of the feature could be done for example as discussed on stackexchange.
9 Pre-defined Checklist Types and States

\begin{CheckList}{Goal}
\Goal{open}{open goal}
\Goal{dropped}{dropped goal}
\Goal{unclear}{unclear goal}
\Goal{achieved}{achieved goal}
\end{CheckList}

\begin{CheckList}{Task}
\Task{open}{open task}
\Task{dropped}{dropped task}
\Task{unclear}{unclear task}
\Task{started}{started task}
\Task{done}{done task}
\end{CheckList}

\begin{CheckList}{Artifact}
\Artifact{missing}{missing artifact}
\Artifact{dropped}{dropped artifact}
\Artifact{unclear}{unclear artifact}
\Artifact{incomplete}{incomplete artifact}
\Artifact{available}{available artifact}
\end{CheckList}

\begin{CheckList}{Milestone}
\Milestone{open}{open milestone}
\Milestone{achieved}{achieved milestone}
\end{CheckList}

Goals
- open goal
- dropped goal
- unclear goal
- achieved goal

Tasks
- open task
- dropped task
- unclear task
- started task
- done task

Artifacts
- missing artifact
- dropped artifact
- unclear artifact
- incomplete artifact
- available artifact

Milestones
- open milestone
- achieved milestone
10 Implementation

10.1 Basic Package Dependencies

We use the \texttt{xkeyval} package for declaring package options as well as for option lists of entry types.

\begin{verbatim}
\RequirePackage{xkeyval}
\end{verbatim}

We use the \texttt{etoolbox} package for simpler handling of lists.

\begin{verbatim}
\RequirePackage{etoolbox}
\end{verbatim}

We use colors for deadlines, for instance.

\begin{verbatim}
\RequirePackage{xcolor}
\end{verbatim}

10.2 Options

10.2.1 Checklist Options

In the following, we define the possible options for a checklist.

\begin{verbatim}
define@key[tchklst]{GlobalListOptions}{layout}{% \ifinlist{#1}{tchklst@ChecklistLayouts}{}{% PackageError{typed-checklist}{{#1} not a known checklist layout} \{Known layouts are:\forlistloop{ }{tchklst@@CheckListLayouts}}}% \def\tchklst@@layout{#1}
\end{verbatim}

\begin{verbatim}
define@key[tchklst]{GlobalListOptions}{input-dates}{% \ifinlist{#1}{tchklst@@InputDateFormats}{}{% PackageError{typed-checklist}{{#1} not a known input date format} \{Known formats are:\forlistloop{ }{tchklst@@InputDateFormats}}}% \letcs\tchklst@inputdate@order{tchklst@dateorder@#1}% \letcs\tchklst@inputdate@sep{tchklst@dateformat@sep@#1}
\end{verbatim}

\begin{verbatim}
define@key[tchklst]{GlobalListOptions}{output-dates}{% \ifinlist{#1}{tchklst@@OutputDateFormats}{}{% PackageError{typed-checklist}{{#1} not a known output date format} \{Known formats are:\forlistloop{ }{tchklst@@OutputDateFormats}}}% \letcs\tchklst@@dateoutput@use{tchklst@dateoutput@use@#1}
\end{verbatim}

\begin{verbatim}
define@boolkey[tchklst]{GlobalListOptions}{strict-dates}[true]{% \ifbool{tchklst@GlobalListOptions@strict-dates}{\let\tchklst@@faileddate=tchklst@DateFailStrict}{\let\tchklst@@faileddate=tchklst@DateFailLax}
\end{verbatim}

10.2.2 Checklist Entry Options

The \texttt{CheckListAddEntryOption{⟨option⟩}{⟨default⟩}} macro declares a new \texttt{⟨option⟩} that can be used when defining checklist entries. An option always comes with a \texttt{⟨default⟩} value.

\begin{verbatim}
\CheckListAddEntryOption{⟨option⟩}{⟨default⟩}
\end{verbatim}

\begin{verbatim}
\newcommand*\CheckListAddEntryOption[2]{% \define@cmdkey[tchklst]{Entry}{#1}{#2} \presetkeys[tchklst]{Entry}{#1}{#2}
\end{verbatim}
In the following, we define a basic default set of possible options for a checklist entry.

\CheckListAddEntryOption{who}{}
\CheckListAddEntryOption{deadline}{}
\CheckListAddEntryOption{label}{}

10.3 Setting Options Globally

\CheckListSet{⟨options-list⟩} sets global options for the typed-checklist package.

\newcommand\CheckListSet[1]{\setkeys[tchklst]{GlobalListOptions}{#1}}

\CheckListDefaultLayout{⟨layout⟩} macro sets the default layout for all Checklist environments that do not set the layout option explicitly. This macro is obsoleted by the \CheckListSet macro introduced in v2.0 of the package.

\newcommand*\CheckListDefaultLayout[1]{\CheckListSet{layout={#1}}}

10.4 Checklist Types

In the following, we implement the existing types of checklists as well as the macros for declaring new types.

\tchklst@ChecklistTypes collects the list of known checklist types. Initially, the list is empty.

\newcommand*\tchklst@ChecklistTypes{}

\CheckListAddType{⟨type⟩}{⟨symbol⟩} adds a new checklist type with name ⟨type⟩ to the list of known checklist types. The basic symbol of entries belonging to this checklist type will be ⟨symbol⟩ (e.g., an empty box or circle).

\newcommand*\CheckListAddType[2]{% Add new type to existing list, if the type is not already known.
  \ifinlist{#1}{\tchklst@ChecklistTypes}{% PackageError{typed-checklist}{Checklist type '#1' already defined}{}
  \listadd\tchklst@ChecklistTypes{#1}%;
  \csdef{tchklst@ChecklistTypeSym@#1}{#2};%
  Create an initially empty list of possible states that entries of the type can have, and an empty list of filters for the type.
  \csdef{tchklst@ChecklistStates@#1}{}
  \csdef{tchklst@ChecklistFilters@#1}{}
  Finally, invoke all hooks for new types of checklists.
  \def\do##1{##1{#1}}%
  \dolistloop\tchklst@@addtype@hooks}
This is an etoolbox list of single-argument macros for hooking into the registration of new checklist types.

\newcommand*{\tchklst@@addtype@hooks}{}

The \tchklst@IntroduceTypeHook{⟨cmd⟩} macro introduces ⟨cmd⟩ for all existing checklist types (first code line) as well as for all checklist types defined afterwards (second code line).

\newcommand*{\tchklst@IntroduceTypeHook}[1]{%  
\forlistloop{#1}{\tchklst@ChecklistTypes}{%  
\listgadd{\tchklst@@addtype@hook}{#1}}%

The \tchklst@aux@OargAfter{⟨macro-use⟩}{⟨opt-arg⟩} macro inserts an optional argument, ⟨opt-arg⟩, into a ⟨macro-use⟩, where the ⟨macro-use⟩ may have multiple mandatory arguments but no optional argument. The ⟨opt-arg⟩ is optional, i.e., if it is not provided, then ⟨macro-use⟩ is taken as is.

Example use: \tchklst@aux@OargAfter{\cite{foo}}[page 9] would expand first to \tchklst@aux@OargAfter@ii{page 9}\cite{foo} and, finally, to \cite[page 9]{foo}.

\newcommand\tchklst@aux@OargAfter[1]{%  \@ifnextchar[{	chklst@aux@OargAfter@i{#1}}{#1}}%
\long\def\tchklst@aux@OargAfter@i#1[#2]{%  \tchklst@aux@OargAfter@ii{#2}{#1}}%
\newcommand\tchklst@aux@OargAfter@ii[2]{%  #2[#1]}%

The \tchklst@CheckType{⟨type⟩} is a convenience macro for checking whether the checklist type ⟨type⟩ is defined. This macro yields an error with a simple message if ⟨type⟩ is not defined.

\newcommand*{\tchklst@CheckType}[1]{%  \ifinlist{#1}{\tchklst@ChecklistTypes}{}{%  \PackageError{typed-checklist}{Unknown checklist type ‘#1’}{%  \Unknown checklist type ‘#1’}{%  \Known types are: \forlistloop{}{\tchklst@ChecklistTypes}}}%

10.5 Checklist Entry States

In the following, we implement the existing status possibilities of the individual checklist types as well as macros for declaring a new status.

\CheckListAddStatus{⟨types⟩}{⟨status⟩}{⟨isclosed⟩}{⟨symbol⟩} macro declares a new ⟨status⟩ for a given comma-separated list of checklist ⟨types⟩. The ⟨symbol⟩ is \LaTeX code of a symbol that is put on top of the checklist type's symbol. The ⟨isclosed⟩ parameter must be one of true or false. A value of true indicates that the status of the entry corresponds to the entry being closed. This particularly means that no warning will be shown if the deadline of an entry with this status is passed. A value of false for ⟨isclosed⟩ indicates that the ⟨status⟩ corresponds to the entry not yet being closed.

\newcommand*{\CheckListAddStatus}[4]{%
We loop over all the checklist \textit{types} given.

In the following line, the actual type parameter is added last by the \texttt{forcsvlist} macro.

\begin{verbatim}
{\tchklst@AddStatus{#2}{#3}{#4}}%
{#1}}% 
\end{verbatim}

\texttt{\tchklst@AddStatus} has the same parameters (in different ordering) and intention as the \texttt{\CheckListAddStatus} macro, except that it assumes a single \textit{type} instead of a type list. This macro is used internally by \texttt{\CheckListAddStatus}.

Some argument checking up front.

\begin{verbatim}
\newcommand*{\tchklst@AddStatus}[4]{%
Some argument checking up front.
\tchklst@CheckType{#4}%
\ifinlistcs{#1}{tchklst@ChecklistStates@#4}{% 
\PackageError{typed-checklist}{% #4-checklist state ‘#1’ already defined}{}}{}% 
\listcsadd{tchklst@ChecklistStates@#4}{#1}%
\end{verbatim}

\texttt{\tchklst@CheckTypeStatus} is a convenience macro for checking whether the checklist entry status \textit{status} is defined for checklist type \textit{type}. This macro yields an error with a simple message if \textit{status} is not defined.

\begin{verbatim}
\newcommand*{\tchklst@CheckTypeStatus}[2]{% 
\ifinlistcs{#2}{tchklst@ChecklistStates@#1}{}{% 
\PackageError{typed-checklist}{}{Unknown #1-checklist entry status ‘#2’}{}{Known states are:\forlistcsloop{ }{tchklst@ChecklistStates@#1}}}}
\end{verbatim}

\texttt{\CheckListStatusSymbol} is a convenience macro for obtaining the symbol for a particular \textit{status} of the current checklist’s type.

\begin{verbatim}
\newcommand*{\CheckListStatusSymbol}[1]{% 
\tchklst@symbolcombine{\csuse{tchklst@sym@\tchklst@cur@type @#1}}% \csuse{tchklst@ChecklistTypeSym@\tchklst@cur@type}}
\end{verbatim}

\texttt{\tchklst@symbolcombine} macro combines two symbols, \textit{symbol1} and \textit{symbol2}.

\begin{verbatim}
\newcommand*{\tchklst@symbolcombine}[2]{% 
\setbox0\hbox{#2}\
\copy0\llap{\hbox to \wd0{\hss\smash{#1}\hss}}}
\end{verbatim}

\texttt{\CheckListIfClosed} macro expands to \texttt{\texttt{iiftrue}}, if the \textit{status} of an entry in the current checklist is a “closed” one (see the documentation for \texttt{\CheckListAddStatus} for details). Otherwise, the macro expands to \texttt{\texttt{iffalse}}.
\newcommand*{\CheckListIfClosed}[1]{% 
\csname if\csname tchklst@isclosed@tchklst@cur@type @#1\endcsname\endcsname
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi}

10.6 Checklist Layouts

The \tchklst@ChecklistLayouts collects the list of known checklist layouts. Initially, the list is empty.

\newcommand*{\tchklst@ChecklistLayouts}{%

\CheckListDeclareLayout

The \CheckListDeclareLayout\{\langle name\rangle\}\{\langle fields\rangle\}\{\langle begin\rangle\}\{\langle end\rangle\} macro declares a new checklist layout with the given \langle name\rangle. At the begin and end of the checklist, the \langle begin\rangle and, respectively, \langle end\rangle code is executed. The \langle fields\rangle parameter must be a comma-separated list of field names. The fields will be displayed for each checklist entry in the order given by \langle fields\rangle, where the format for the display must be declared using \CheckListDefineFieldFormat.

\newcommand*{\CheckListDeclareLayout}[4]{% Add new layout to existing list, if the layout is not already known.
\ifinlist{#1}{\tchklst@ChecklistLayouts}{% 
\PackageError{typed-checklist}{Checklist layout '#1' already declared}{}%}
\listadd\tchklst@ChecklistLayouts{#1}()% Save the \langle fields\rangle list of the new layout.
\csdef{tchklst@ChecklistLayoutFields@#1}{}% Save the \langle begin\rangle and \langle end\rangle code of the new layout.
\csdef{tchklst@ChecklistLayoutBegin@#1}{#3}%
\csdef{tchklst@ChecklistLayoutEnd@#1}{#4}}%

\CheckListExtendLayout

The \CheckListExtendLayout\{\langle name\rangle\}\{\langle base\rangle\}\{\langle fields\rangle\} macro declares a new checklist layout, \langle name\rangle, which inherits existing \langle fields\rangle as well as the \langle begin\rangle and \langle end\rangle code from a given \langle base\rangle layout.

\newcommand*{\CheckListExtendLayout}[3]{% Inherit all fields defined by the \langle base\rangle layout.
\def\do##1{% 
\ifcsdef{tchklst@ChecklistFormat@#2@##1}{% 
\csletcs{tchklst@ChecklistFormat@#1@##1}{tchklst@ChecklistFormat@#2@##1}}{%}
\dolistcsloop{tchklst@ChecklistLayoutFields@#2}%;}
}%
The \CheckListDefineFieldFormat{⟨layout⟩}{⟨field⟩}{⟨code⟩} macro defines the ⟨code⟩ to be used for displaying the given ⟨field⟩ (or fields) in a checklist of the given ⟨layout⟩. Multiple fields can be displayed by specifying ⟨field⟩ in the form ⟨field⟩₁ + ... + ⟨field⟩ₙ.

\begin{verbatim}
\newcommand{\CheckListDefineFieldFormat}[3]{%\end{verbatim}
\begin{verbatim}
  \chklst@deffieldmacro{tchklst@ChecklistFormat@#1@#2}{#2}{#3}}%
\end{verbatim}

The \tchklst@deffieldmacro{⟨csname⟩}{⟨fields⟩}{⟨code⟩} defines a command with name ⟨csname⟩ whose number of arguments equals the number of ‘+’-separated elements in ⟨fields⟩. The command then expands to ⟨code⟩, which can refer to the respective number of positional parameters.

\begin{verbatim}
\newcommand{\tchklst@deffieldmacro}[3]{%\end{verbatim}
\begin{verbatim}
  \begingroup
  Get number of properties (‘+’-separated) in ⟨field⟩ into @tempcnta.
  \@tempcnta=0\relax
  \def\do##1\{\advance\@tempcnta by 1\relax\}%
  \tchklst@dopsvlist{#2}%
  Next, use the above number for determining the number of arguments of the defined formatting macro, i.e., the number of positional parameters permitted in ⟨code⟩. The macro is first \undefined’d such that \newcommand always succeeds.
  \edef\do{\endgroup
           \csundef{#1}%
           \noexpand\newcommand\expandonce{\csname #1\endcsname}%
           \[the\@tempcnta]{\unexpanded{#3}}}%
  \do}%
\end{verbatim}

The \tchklst@usefieldmacro{⟨use-cmd⟩}{⟨csname⟩}{⟨fields⟩} macro takes the current values of the fields in the ‘+’-separated ⟨fields⟩ and applies them in the given order to ⟨csname⟩. This application is performed directly, if ⟨use-cmd⟩ is left at its default, or is otherwise provided as an argument to ⟨use-cmd⟩.

\begin{verbatim}
\newcommand{\tchklst@usefieldmacro}[3][\@firstofone]{%\end{verbatim}
\begin{verbatim}
  \begingroup
  \expandafter\def\expandafter\tchklst@@cmd\expandafter{\csname #2\endcsname}
  \def\do##1\{\eappto\tchklst@@cmd{\csexpandonce{cmdtchklst@Entry@##1}}\}
  \tchklst@dopsvlist{#3}%
  \expandafter\def\expandafter\tchklst@@cmd\expandafter{\expandafter{\tchklst@@cmd}}%
  \preto\tchklst@@cmd{\endgroup#1}%
  \tchklst@@cmd}
\end{verbatim}

The \tchklst@CheckLayout{⟨layout⟩} is a convenience macro for checking whether the checklist layout ⟨layout⟩ is defined. This macro yields an error with a simple message if ⟨layout⟩ is not defined. If a command is provided for ⟨layout⟩, it is expanded.

\begin{verbatim}
\newcommand{\tchklst@CheckLayout}[1]{%\end{verbatim}
\begin{verbatim}
  \xifinlist{#1}{\tchklst@ChecklistLayouts}{}{%\end{verbatim}
\begin{verbatim}
    \PackageError{typed-checklist}...
\end{verbatim}
\end{verbatim}
10.7 Entry Filters

The `CheckListSetFilter{⟨types⟩}{⟨fields⟩}{⟨code⟩}` macro defines a filter for the given `⟨fields⟩` of all types in the comma-separated list `⟨types⟩`. The filtering code is `⟨code⟩`, which may use positional parameters.

```latex
\newcommand*{CheckListSetFilter}[3][*]{
  \ifstrequal{#1}{*}{
    \forlistloop{\tchklst@SetFilter{#2}{#3}}{\tchklst@ChecklistTypes}
  }{
    \forcsvlist{\tchklst@SetFilter{#2}{#3}}{#1}
}
```

The `tchklst@SetFilter{⟨fields⟩}{⟨code⟩}{⟨type⟩}` macro defines a filter for a single type.

```latex
\newcommand*{tchklst@SetFilter}[3]{
  \tchklst@CheckType{#3}
  \ifinlistcs{#1}{tchklst@ChecklistFilters@#3}{}
  {\listcsadd{tchklst@ChecklistFilters@#3}{#1}}
  \tchklst@deffieldmacro{tchklst@CheckListFilter@#3@#1}{#1}{#2}
```

The `CheckListFilterValue{⟨types⟩}{⟨field⟩}{⟨value⟩}` macro filters out all checklist entries whose `⟨field⟩` is unequal `⟨value⟩`, by using an \ifstrequal comparison.

```latex
\newcommand*{CheckListFilterValue}[3][*]{
  \CheckListSetFilter[#1]{#2}
  \ifstrequal{##1}{#3}{}{\togglefalse{display}}
```

The `CheckListFilterClosed{⟨types⟩}{⟨field⟩}{⟨value⟩}` macro filters out all checklist entries whose status is closed, by using `CheckListIfClosed`.

```latex
\newcommand*{CheckListFilterClosed}[1][*]{
  \CheckListSetFilter[#1]{status}
  \CheckListIfClosed{##1}{\togglefalse{display}}{}}
```

The `CheckListFilterDeadline{⟨types⟩}{⟨comp⟩}{⟨refdate⟩}{⟨filter-inv⟩}` macro filters out all checklist entries whose deadline does not satisfy the comparison against `⟨refdate⟩` by operator `⟨comp⟩` (‘<’, ‘=’, ‘>’). The argument `⟨filter-inv⟩` must be either true or false and specifies whether deadlines that do not match the selected input deadline format are filtered out (true) or not (false).

```latex
\newcommand*{CheckListFilterDeadline}[4][*]{
  First, pre-parse `⟨refdate⟩` such that it need not be parsed for each checklist entry.
  \bgroup
  \def\do##1##2##3##4{\egroup
    Use the internal `tchklst@DateCompare` macro to perform the date comparison based on the pre-parsed date of the `\do{⟨year⟩}{⟨month⟩}{⟨day⟩}` macro arguments.
  \CheckListSetFilter[#1]{deadline}
  \tchklst@DateCompare{####1}{#2}{##1}{##2}{##3}
  {\togglefalse{display}}
  \egroup
```
If parsing \langle \textit{refdate} \rangle fails, we always fail like with strict input date parsing: Setting up a filter with an invalid date would not make sense. 

The \texttt{\CheckListFilterReset(\langle \textit{types} \rangle)} macro resets the filters for all checklist types in the comma-separated list \langle \textit{types} \rangle. If \langle \textit{types} \rangle is omitted or equals '*', then the filters for all checklist types are reset. 

The \texttt{\tchklst@ResetFilter(\langle \textit{type} \rangle)} macro removes all filters (i.e., for all fields) from checklists of the given \langle \textit{type} \rangle.

\section*{10.8 Checklist and Entry Definition}

The \texttt{\CheckList(\langle \textit{options} \rangle)(\langle \textit{type} \rangle)} environment declares a new checklist.

We check whether the provided \langle \textit{type} \rangle is known. 

Parse and check the options. 

We store the type, layout, and fields of the checklist for use inside the list. 

The following line declares the macro for the checklist entries, for example the \texttt{\Goal} macro for the \langle \textit{type} \rangle \texttt{Goal}. 

Start and end the actual checklist environment as defined by the layout. 

The \texttt{\tchklst@entry(\langle \textit{options} \rangle)(\langle \textit{status} \rangle)(\langle \textit{description} \rangle)} macro defines a checklist entry with a given \langle \textit{status} \rangle, a given \langle \textit{description} \rangle, and possibly particular
⟨options⟩ (a comma-separated list of key-value pairs). See Section 10.2.2 for the list of available options.

First check for a valid status. There is no need to check for a valid type, because the surrounding CheckList environment already does this.

Parse the options.

Save status and description such that they can be accessed just like the options.

Now iterate through all filters for the current type until one filter turns the local display toggle to false.

Show the fields of the entry in the order they were given. The whole entry is first collected in a macro ⟨chkklst@entry⟩, such that individual field display code cannot leave the current \LaTeX group (e.g., by advancing to the next table cell in table layout) and thereby void the entry option macros.

The \chkklst@dopsvlist{⟨list⟩} parses a ‘+’-separated list.

The \CheckListSigned{⟨core⟩}{⟨text⟩} macro is taken from Knuth’s TeXbook with minor spacing modifications. See also http://tex.stackexchange.com/a/13761. The added optional ⟨core⟩ is the reference for checks whether ⟨text⟩ is empty: In case of emptiness, nothing is shown by the macro. If ⟨core⟩ is omitted, ⟨text⟩ itself is used in the emptiness check.
10.9 Deadlines

The following code implements the parsing of deadlines and for comparing deadlines against the current date.

\CheckListDisplayDeadline \The \CheckListDisplayDeadline \{\langle \text{status} \rangle\}\{\langle \text{deadline} \rangle\} formats a \langle \text{deadline} \rangle dependent on the \langle \text{status} \rangle and the current date.

\CheckListIfClosed \% \begin{macrocode}
\checkclist@ifafter{#1}{#2}{#3}
\appto\checkclist@@args{{true}}\appto\checkclist@@args{{false}}
\CheckListHighlightDeadline \expandafter\CheckListHighlightDeadline \checkclist@@args
\end{macrocode}

\CheckListHighlightDeadline \The \CheckListHighlightDeadline \{\langle \text{closed?} \rangle\}\{\langle \text{passed?} \rangle\}\langle \text{deadline} \rangle macro highlights the given \langle \text{deadline} \rangle based on the two Boolean (‘true’ or ‘false’) arguments \langle \text{done?} \rangle (whether the respective checklist entry has been completed) and \langle \text{passed?} \rangle (whether the deadline has passed). One can \renewcommand this macro to change the deadline highlighting.
The following auxiliary macro just swaps the first two arguments, \textit{⟨text⟩} and \textit{⟨sep⟩} of \texttt{\thklst@splitapply} such that the now first argument \textit{⟨sep⟩} can be expanded more easily.

\newcommand*{\thklst@splitapply@i}[2]{\thklst@splitapply{#2}{#1}}

The \texttt{\thklst@ifafter{⟨y⟩}{⟨m⟩}{⟨d⟩}{⟨iftrue⟩}{⟨iffalse⟩}} macro performs the check whether the current date is after the date specified by \textit{⟨y⟩}, \textit{⟨m⟩}, and \textit{⟨d⟩}. If this is the case, the macro expands to \textit{⟨iftrue⟩}, otherwise to \textit{⟨iffalse⟩}. Credits for this code go to \url{http://tex.stackexchange.com/questions/41404/how-to-make-time-dependent-code/}.

\newcommand*{\thklst@ifafter}[3]{\ifnum\the\year	wo@digits\month	wo@digits\day% #2\else#3\fi}

The \texttt{\CheckListDateCompare{⟨date⟩}{⟨comp⟩}{⟨refdate⟩}{⟨iftrue⟩}{⟨iffalse⟩}{⟨iffail⟩}} macro compares \textit{⟨date⟩} against \textit{⟨refdate⟩} using the operator \textit{⟨comp⟩}. The latter must be one of <, =, and >. If the dates fulfill the comparison, the macro expands to \textit{⟨iftrue⟩}. If the dates do not fulfill the comparison, the macro expands to \textit{⟨iffalse⟩}. Finally, if one of \textit{⟨date⟩} and \textit{⟨refdate⟩} are not recognized as dates, the macro expands to \textit{⟨iffail⟩}.

\newcommand{\CheckListDateCompare}[6]{\bgroup\def\do##1##2##3##4\egroup\tchklst@DateCompare{#1}{#2}{##1}{##2}{##3}{#4}{#5}{#6}}

\newcommand{\tchklst@DateCompare}[8]{\bgroup\def\do##1##2##3##4\egroup\tchklst@DateCompare{##1}{##2}{##3}{##4}{##5}{##6}{#7}{#8}}

\CheckListParseDate The \texttt{\CheckListParseDate{⟨date⟩}{⟨cmd⟩}{⟨fail⟩}} macro parses \textit{⟨date⟩} according to the selected date input format. If the parsing succeeds, the macro expands to
\texttt{\texttt{cmd}}\langle\textbf{year}\rangle\langle\textbf{month}\rangle\langle\textbf{day}\rangle\langle\textbf{date}\rangle \ (i.e., \texttt{cmd} must take four arguments). Otherwise, the macro expands to \texttt{\texttt{fail}}\langle\textbf{date}\rangle \ (i.e., \texttt{\texttt{fail}} must take one argument).

253 \newcommand\CheckListParseDate[3]{% 254 \expandafter\tchklst@splitapply@i\expandafter{\tchklst@inputdate@sep} 255 \mbox{(#1)}}

Dates have three components and all must be positive numbers.

Before expanding to \texttt{cmd}, reorder the parsed date components according to the ordering of the selected date input format.

The following argument is applied to whichever of the two previous arguments \texttt{tchklst@splitapply@i} expands to.

The following set of macros is for registering date input and date output formats.

\texttt{\texttt{tchklst@InputDateFormats}}  \texttt{\texttt{tchklst@OutputDateFormats}}

\texttt{tchklst@registerdateinputfmt} \texttt{tchklst@registerdateoutputfmt}
The following \numexpr looks a bit unfamiliar but it computes the modulo, given that integer division rounds the result.

\newcommand\tchklst@splitapply[6]{%
\def\tchklst@split@@rec##1##2##3#2##4\relax{% 
Check whether \textit{cond} holds for \langle prefix \rangle first. 
\ifnumgreater{##1}{0}{#4}{##3}{#2} \relax %}

The remainder of this section defines generic auxiliary macros for deadline parsing.
If \langle suffix \rangle is empty, then \langle text \rangle contained too few components and, hence, expand to \langle fail \rangle. Otherwise recurse.

\{\ifstrempty{##4}
\{\#6\}
{\tchklst@split@@rec{##1-1}{##2{##3}}##4\relax}\}%

Otherwise, if \langle k \rangle = 0, and \langle suffix \rangle is empty, then \langle text \rangle indeed contains \langle n \rangle components and \langle prefix \rangle is appended to \langle cmd \rangle as the last component. If \langle suffix \rangle is nonempty, expand to \langle fail \rangle.

\{\ifstrempty{##4}
\{##2{##3}\}
{#6}\%

If \langle cond \rangle does not hold, expand to \langle fail \rangle.

\tchklst@ifPositive{\langle text \rangle}{\langle iftrue \rangle}{\langle iffalse \rangle}
macro expands to \langle iftrue \rangle if \langle text \rangle is a positive number and expands to \langle iffalse \rangle otherwise (i.e., if \langle text \rangle is not a number or not positive). The code of the macro is taken from Donald Arseneau’s cite package.

\newcommand*{\tchklst@ifPositive}[1]{\ifcat _\ifnum\z@<0#1_\else A\fi
\expandafter\@firstoftwo \else \expandafter\@secondoftwo \fi}

10.10 Default Checklist Types and States

We use some packages for the default symbols in the checklist.

\RequirePackage{bbding}

The following line makes sure that the bbding font is actually loaded, by simply putting a particular symbol into a box and then forgetting the box again (via the grouping). This addresses the case that the bbding symbols are used inside an \import* or \subimport* of the import package: In this case, the font would be attempted to be loaded only inside the ‘import’ and could then no longer be found (producing “No file Uding.fd”).

\AtBeginDocument{\setbox0\hbox{\Checkmark}}

The following provides the default set of checklist types.

\CheckListAddType{Goal}{$\bigcirc$}
\CheckListAddType{Task}{{\small\Square}}
\CheckListAddType{Artifact}{{\large$\bigtriangleup$}}
\CheckListAddType{Milestone}{\FiveStarOpen}

The following provides the default set of status possibilities.

\CheckListAddStatus{Goal,Task,Milestone}{open}{false}\%
\CheckListAddStatus{Goal}{dropped}{true}{\tiny\XSolid}\%
\CheckListAddStatus{Task}{dropped}{true}{\small\XSolid}\%
\CheckListAddStatus{Goal}{unclear}{false}{\footnotesize ?}\%
\CheckListAddStatus{Task}{unclear}{false}\%
\CheckListAddStatus{Artifact}{unclear}{false}\%
The following provides the default set of checklist layouts.

### 10.11 Default Checklist Layouts

The list layout is based on a description environment with a slightly modified vertical and horizontal spacing.

```latex
\CheckListDeclareLayout{list}{status,label,description, who,deadlines+status,END}%
{\bgroup\topsep=\medskipamount\itemsep=0pt\itemize\@newlistfalse}%
{\global\@newlistfalse\enditemize\egroup}
```

The checklist entry starts with the status symbol, which opens up a new list item.

```latex
\CheckListDefineFieldFormat{list}{status}{}
\CheckListDefineFieldFormat{list}{label}{%\ifstrempty{#1}{}{%\CheckListDefaultLabel{#1}%\ifbool{inner}{%\mbox{\small(#1)}%\nobreak\hskip 0pt plus50pt\allowbreak\hskip 0pt plus-50pt\relax}{}%\leavevmode\reversemarginpar\marginpar{%\textcolor{gray}{\underbar{\hbox to \hsize{\normalfont\textcolor{black}{\ref{#1}}\hfil}}}}}%\CheckListDefineFieldFormat{list}{description}{%\ignorespaces #1\relax}%\CheckListDefineFieldFormat{list}{who}{%\CheckListSigned[#1]{\textit{(#1)}}}%
```

Show the label in the reverse margin, with some nice layout.

```latex
\ifstrempty{#1}{}{%\CheckListDefaultLabel{#1}%%%%%%%%%%%%%%%%%%%%\ifbool{inner}{}{\leavevmode\reversemarginpar\marginpar{%\textcolor{gray}{\underbar{\hbox to \hsize{\normalfont\textcolor{black}{\ref{#1}}\hfil}}}}}%
```

Show the description, with leading spaces removed.

```latex
\ifbool{inner}{}{\mbox{\small(#1)}}%\nobreak\hspace{0pt plus50pt}\allowbreak%\hspace{0pt plus-50pt}\relax
```

Show the responsible person(s), if the who option is given in \textit{⟨options⟩}.

```latex
\CheckListDefineFieldFormat{list}{who}{%\CheckListSigned[#1]{\textit{(#1)}}}%
```

Show the responsible person(s), if the who option is given in \textit{⟨options⟩}.
Show the deadline of the entry in the margin, if the deadline option is given in \textit{(options)}.

357 \CheckListDefineFieldFormat{list}{deadline+status}{\% \\
358 \ifstrempty{#1}{}{\normalmarginpar\margintext{#1}}\% \\
359 \begin{quotation}
The following \texttt{\unskip} prevents \texttt{\marginnote} from breaking an overfull margin text at its very beginning, which meant that the margin text would vertically be placed below the actual entry (see also \texttt{https://tex.stackexchange.com/questions/117695/}).
\end{quotation}
360 \unskip
361 \CheckListDisplayDeadline{#2}{#1}\}
\end{quotation}

End the display of one checklist entry. \textit{(options)}.

362 \CheckListDefineFieldFormat{list}{END}{\% \\
363 \parfillskip=0pt \finalhyphendemerits=0 \endgraf}

10.11.2 \textbf{hidden}

The hidden layout completely hides the checklist and all its entries. We add the status field only to ignore spaces after each entry.

363 \CheckListDeclareLayout{hidden}{dummy}{\ignorespaces}{\ignorespaces}
364 \CheckListDefineFieldFormat{hidden}{dummy}{\ignorespaces}

10.11.3 \textbf{table}

The table layout formats the checklist as a table, one row per checklist entry. The NC field just inserts the column separator.

365 \CheckListDeclareLayout{table}\% \\
366 {newline,status,NC,label,description,NC,who,NC,deadline+status}\% \\
367 {\% \\
368 \tchklst@@begintab\hline}
369 \gdef\tchklst@@newline{\hline\tchklst@@endhead}
370 \gdef\tchklst@@newline{\hline}
371 \tcbf{Status} & \tcbf{Description} & \\
372 \tcbf{Who} & \tcbf{Deadline} & \\
373 \{\tchklst@@endtab}
374 \CheckListDefineFieldFormat{table}{newline}{\tchklst@@newline}
375 \CheckListDefineFieldFormat{table}{status}{\CheckListStatusSymbol{#1}}
376 \CheckListDefineFieldFormat{table}{label}{\% \\
377 \ifstrempty{#1}{}\% \\
378 \leavevmode\CheckListDefaultLabel{#1}\% \\
379 \mbox{\small\ref{#1}}\% \\
380 \nobreak\hskip Opt plus50pt\allowbreak \\
381 \hskip Opt plus-50pt\relax}
382 \CheckListDefineFieldFormat{table}{description}{\ignorespaces #1}
The following macros define the package-specific table code.

\CheckListDefinieFieldFormat{table}{deadline+status}{\% 
\ifstrempty{#1}{}{\CheckListDisplayDeadline{#2}{#1}}}
\CheckListDefinieFieldFormat{table}{who}{#1}
\CheckListDefinieFieldFormat{table}{NC}{&}

The following three macros specify how the xltabular package is initialized (i.e., how the package is loaded) and how table environments are started and, respectively, ended.

\newcommand{\chklst@inittab@xltabular}{{\RequiroPackage{array,xltabular}}}
\newcommand{\chklst@begintab@xltabular}{{\setlength{\extrarowheight}{0.5ex}\def{\chklst@@endhead}{\endhead}}}
\chklst@inittab@xltabular
\chklst@begintab@xltabular
\chklst@endtab@xltabular

The following modifies the internal end code of xltabular such that \endxltabular can be the first token as required while there is still the horizontal line.

\preto{\XLT@ii@TX@endtabularx}{\toks@\expandafter{\the{\toks@}\chklst@@newline}}%\tabularx{\linewidth}{|c|X|l|r|}}%\preto{\TX@endtabularx}{\toks@\expandafter{\the{\toks@}\chklst@@newline}}%\tabularx{\linewidth}{|c|X|l|r|}}%\newcommand{\chklst@endtab@xltabular}{\endxltabular}
\chklst@inittab@tabularx
\chklst@begintab@tabularx
\chklst@endtab@tabularx

The following is analogous to its counterpart in xltabular.

\preto{\TX@endtabularx}{\toks@\expandafter{\the{\toks@}\chklst@@newline}}%\tabularx{\linewidth}{|c|X|l|r|}}%\newcommand{\chklst@endtab@tabularx}{\endtabularx}
\chklst@inittab@ltablex
\chklst@begintab@ltablex
\chklst@endtab@ltablex

The following fixes a bug in ltablex, see https://tex.stackexchange.com/a/197000/132738.

\newcommand{\chklst@inittab@ltablex}{{\RequiroPackage{ltablex}}}
\newcommand{\chklst@begintab@ltablex}{\end{tabularx}}
\newcommand{\chklst@endtab@ltablex}{\endgroup}
\chklst@inittab@ltablex
\chklst@begintab@ltablex
\chklst@endtab@ltablex

The following fixes a bug in typed-checklist, see https://tex.stackexchange.com/a/197000/132738.

\Patchcmd{\TX@endtabularx}{\end{tabularx}}{\end{tabularx}\endgroup}{}{\PackageError{typed-checklist}{{Could not apply code patch to ltablex' package.}}}
10.12 Package Options

10.12.1 Package-Only Options

The `withAsciilist` option enables support for the `asciilist` package.

```latex
\define@boolkey[tchklst]{PackageOptions}[tchklst@]{withAsciilist}[true]{}
```

The `tablepkg` option specifies which table package is used for layouting checklists in the table layout.

```latex
\define@choicekey[tchklst]{PackageOptions}{tablepkg}{[val]
  \letcs\tchklst@@inittab{tchklst@inittab@[val]}
  \letcs\tchklst@@begintab{tchklst@begintab@[val]}
  \letcs\tchklst@@endtab{tchklst@endtab@[val]}
}
```

The `onecounter` option specifies whether a single counter shall be used for all entry labels, no matter the entry types, or whether one counter per entry type shall be used.

```latex
\define@boolkey[tchklst]{PackageOptions}[tchklst@]{onecounter}[true]{}
```

10.12.2 Processing Options

Set option defaults and then load the given options.

```latex
\ExecutableOptionsX[tchklst]<PackageOptions,GlobalListOptions>{%
  withAsciilist=false,
  tablepkg=xltabular,
  onecounter=true,
  layout=list,
  input-dates=d.m.y,
  output-dates=same,
  strict-dates=false,
%
\ProcessOptionsX[tchklst]<PackageOptions,GlobalListOptions>\relax
```

10.12.3 Labels

The \CheckListDefaultLabel{⟨label⟩} macro puts the given ⟨label⟩ and ensures that this ⟨label⟩ is based on the right checklist entry counter.

```latex
\newcommand*{\CheckListDefaultLabel}[1]{%
  \ifstrempty{#1}{}
  \ifbool{tchklst@onecounter}{
    \refstepcounter{tchklst@entryID}
  }{
    \refstepcounter{tchklst@entryID@tchklst@cur@type}
  }
  \label{#1}}
```

The \tchklst@NewEntryCounter{⟨at⟩}{⟨type⟩} creates a new counter for checklist entries and defines the format for displaying counter values. The counter is named tchklst@entryID{⟨at⟩}{⟨type⟩}.

\tchklst@NewEntryCounter
If the package shall use a single counter for all entries then define the counter
and how counter values are displayed here.
\iftchklst@onecounter
  \tchklst@NewEntryCounter[]{}
\else
  Otherwise, register the creation of a new type-specific counter in the hook for new
  checklist types.
  \tchklst@IntroduceTypeHook{\tchklst@NewEntryCounter[@]}
\fi

10.12.4 `asciilist`

If the package is loaded with `asciilist` support...
\iftchklst@withAsciilist
  First, we load the package.
  \RequirePackage{asciilist}
  \tchklst@ChkListEntry
\fi

The `\tchklst@ChkListEntry{⟨item-macro⟩}{⟨content⟩}` macro can be used as
a parameter to `\AsciiListEndArg` of the `asciilist` package in order to allow for
checklist entries in an AsciiList.
\newcommand*{\tchklst@ChkListEntry}{⟨item-macro⟩}{⟨content⟩}{⟨status+opts⟩}{⟨descr⟩}
\tchklst@RegisterAsciiTypeEnv The `\tchklst@RegisterAsciiTypeEnv{⟨type⟩}` registers an `asciilist` environment
for the given checklist `{type}`.
\newcommand*{\tchklst@RegisterAsciiTypeEnv}{⟨type⟩}{}
## Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1</td>
<td>General: Initial version</td>
</tr>
<tr>
<td>v0.2</td>
<td>General: Better handling of empty “who”</td>
</tr>
<tr>
<td>v0.3</td>
<td>General: Added deadline and label support</td>
</tr>
<tr>
<td>v0.4</td>
<td>General: Added “dropped” tasks</td>
</tr>
<tr>
<td>v0.4b</td>
<td>General: Fix package dependencies (xcolor)</td>
</tr>
<tr>
<td>v0.5</td>
<td>General: Added “dropped” artifacts</td>
</tr>
<tr>
<td>v0.6</td>
<td>General: Indication of closed checklist entries</td>
</tr>
<tr>
<td>v1.0</td>
<td>General: First documented version</td>
</tr>
<tr>
<td>v1.1</td>
<td>General: Added definable layouts</td>
</tr>
<tr>
<td>v1.1b</td>
<td>General: Fix for more comprehensible error messages when end of environment is forgotten</td>
</tr>
<tr>
<td>v1.1c</td>
<td>General: Added milestone checklists</td>
</tr>
<tr>
<td>v1.2</td>
<td><code>\CheckListAddEntryOption</code>: Added <code>\CheckListAddEntryOption</code> macro</td>
</tr>
<tr>
<td></td>
<td><code>\CheckListExtendLayout</code>: Enabled extensible layouts</td>
</tr>
<tr>
<td>v1.2b</td>
<td><code>\CheckListDefaultLayout</code>: Enabled setting default checklist layouts</td>
</tr>
<tr>
<td>v1.3</td>
<td>General: Support for combining checklists with <code>asciilist</code></td>
</tr>
<tr>
<td>v1.3b</td>
<td>General: Removed dependency on <code>paralist</code> package</td>
</tr>
<tr>
<td>v1.3c</td>
<td><code>\tchklst@RegisterAsciiTypeEnv</code>: Enabled use of optional arguments for <code>asciilist</code> environments</td>
</tr>
<tr>
<td>v1.3d</td>
<td>General: Fixed symbol for dropped tasks</td>
</tr>
<tr>
<td>v1.4</td>
<td>General: Added display of labels to table layout</td>
</tr>
<tr>
<td></td>
<td>Eliminated MnSymbol dependency</td>
</tr>
<tr>
<td></td>
<td>Robustified label display in inner mode</td>
</tr>
<tr>
<td></td>
<td>Robustified use of <code>bbding</code> package</td>
</tr>
<tr>
<td>v1.5</td>
<td>General: Add nobreak to ‘who’ field in ‘list’ layout</td>
</tr>
<tr>
<td></td>
<td>Improve left alignment of entry text in list layout</td>
</tr>
<tr>
<td></td>
<td>Raggedright for labels in case of a narrow list</td>
</tr>
<tr>
<td></td>
<td>Raggedright for labels in case of narrow table display</td>
</tr>
<tr>
<td>v1.5b</td>
<td>General: Fix for list layout (changed to itemize)</td>
</tr>
<tr>
<td>v1.5c</td>
<td>General: Fix vertical placement of deadlines in narrow margins</td>
</tr>
<tr>
<td>v2.0</td>
<td><code>\CheckListDisplayDeadline</code>: More flexibility for deadline parsing and deadline printing</td>
</tr>
<tr>
<td></td>
<td><code>\CheckListHighlightDeadline</code>: Show future deadlines of completed entries also in green</td>
</tr>
<tr>
<td></td>
<td><code>\CheckListSet</code>: Macro added</td>
</tr>
<tr>
<td></td>
<td><code>\CheckListSetFilter</code>: Added customizable filtering feature</td>
</tr>
<tr>
<td></td>
<td><code>\tchklst@entry</code>: Simplified field display through <code>\tchklst@usefieldmacro</code></td>
</tr>
<tr>
<td></td>
<td><code>\tchklst@ifafter</code>: Reversed argument list</td>
</tr>
<tr>
<td></td>
<td><code>\tchklst@symbolcombine</code>: Ignore height of first argument</td>
</tr>
<tr>
<td></td>
<td>Switched combining order</td>
</tr>
<tr>
<td></td>
<td>General: Add package option for using per-type entry counters</td>
</tr>
</tbody>
</table>
Allow empty list-type checklists 33 v2.1
Made "who" column left-aligned in table layout 34
Package option ‘tablepkg’ added 36

Indexes

Symbols
\@dblarg 206
\@firstofone 125
\@firstoftwo 89, 235, 250, 308
\@gobble 162
\@ifnextchar 53
\@newlistfalse 339, 340
\@secondoftwo 91, 237, 251, 308
\@tempcnta 117, 118, 123
\@undefined 456, 457, 458, 459
\/uni2423 349, 381
\advance 118
\allowbreak 348, 380
\appto 201, 221, 223
\ArrowBoldRightStrobe 328, 333
\AsciiListEndArg 464
\AsciiListRegisterEnv 461
\AtBeginDocument 310

A
\begin 214
\begingroup 116, 126, 187
\bseries 322
\bgroup 157, 240, 245, 339
\bigcirc 311
\bigtriangleup 313

B
\begin 214
\begingroup 116, 126, 187
\bseries 322
\bgroup 157, 240, 245, 339
\bigcirc 311
\bigtriangleup 313

C
\CheckList 462
\CheckList (environment) 5, 173
\CheckListAddEntryOption 12, 27, 30, 31, 32
\CheckListAddStatus 10, 63, 315, 316, 317, 318, 319, 321, 324, 325, 327, 329, 331, 332, 334, 335
\CheckListAddType 8, 38, 311, 312, 313, 314
\CheckListCompare 239
\CheckListDeclareLayout 10, 94, 104, 337, 363, 365
\CheckListDefaultLabel 11, 345, 378, 434
\CheckListDefaultLayout 35
\CheckListDefineFormat 11, 113, 341, 343, 353, 355, 357, 361, 364, 374, 375, 376, 382, 383, 385, 386
\CheckListDisplayDeadline 11, 212, 360, 384
\CheckListExtendDeadline 11, 103
\CheckListFilterClosed 13, 153
\CheckListFilterDeadline 14, 156
\CheckListFilterReset 15, 165
\CheckListFilterValue 13, 150
\CheckListHighlightDeadline 11, 224, 226
\CheckListIfClosed 86, 155, 220
\CheckListParseDate 163, 213, 243, 252, 253
\CheckListSet 5, 33, 36
\CheckListSetFilter 15, 141, 151, 154, 159
\CheckListSigned 11, 206, 356
\CheckListStatusSymbol 11, 80, 342, 375
\Checkmark 310, 324, 329, 334
\cmdtchklst@Entry@description 191
\cmdtchklst@Entry@status 190
\copy 85
\csdef 43, 44, 45, 99, 101, 102, 172
\cexpandonce 130
\year \hspace{1cm} 233 \hspace{1cm} \text@ \hspace{1cm} 307