The **typed-checklist** package*

Richard Grewe
r-g+tex@posteo.net
2019-01-15

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** v2.0, dated 2019/01/11. 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

```
\begin{CheckList}[(options)]{(type)}
\end{CheckList}
```

Checklists are created via the CheckList environment. The (type) parameter
determines the type of all checklist entries in the environment. The typed-checklist
package comes with four predefined types: \textit{Goal}, \textit{Task}, \textit{Artifact}, and \textit{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 \textit{⟨options⟩} can be a comma-separated list of \textit{⟨key⟩}=\textit{⟨value⟩} pairs. Table 2 on page 7 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 \texttt{\CheckListSet} macro.

\texttt{\CheckListSet{⟨options-list⟩}}

This macro takes a comma-separated \textit{⟨options⟩} 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.

### 3.2 Checklist Entries

\texttt{\Goal{⟨options⟩}{⟨status⟩}{⟨description⟩}}

Inside a checklist of type \textit{Goal}, the \texttt{\Goal} macro specifies a goal. Every goal 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 goal in a full sentence\textsuperscript{1}. The \textit{⟨status⟩} parameter selects the most recent known status of the goal. This parameter can assume any of the following values\textsuperscript{2}:

<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 \textit{⟨description⟩} was formulated, this might mean that in the respective situation the \textit{⟨description⟩} 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 \textit{⟨options⟩} allow one to specify further details about the goal. The \textit{⟨options⟩} must be a possibly empty, comma-separated list of \textit{⟨key⟩}=\textit{⟨value⟩} pairs. The \textit{⟨key⟩} must be one of the following values\textsuperscript{3}:

<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>

\textsuperscript{1}Incomplete sentences tend to be less clear.

\textsuperscript{2}See Section 4.1 to find out how custom states can be defined

\textsuperscript{3}See Section 4.2 to find out how custom \textit{⟨key⟩}s can be defined.
**deadline** 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`.

**label** 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 `(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 description of a task in a full sentence, possibly in imperative form. The `(options)` parameter can be set as documented for the `\Goal` macro on page 4. The `(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 `(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 `(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 the artifact and its required attributes. The `(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\[\langle options\rangle\\{\langle status\rangle\}\{\langle description\rangle\}\]

Inside a checklist of type\ Milestone, the \Milestone\ macro specifies a milestone. Every milestone 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 identification of what has to exist or must have been fulfilled. The \langle status\rangle 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 8 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\{\langle type\rangle\}\{\langle symbol\rangle\}

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 \langle type\rangle. The basic symbol of entries belonging to this checklist type will be \langle symbol\rangle (e.g., an empty box or circle). All status-symbols (see Section 4.1) are drawn on top of \langle symbol\rangle. Note that the \texttt{typed-checklist} package uses this macro also for creating each of the four default checklist types.

\CheckListAddStatus\{\langle types\rangle\}\{\langle status\rangle\}\{\langle isclosed\rangle\}\{\langle symbol\rangle\}

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 \langle status\rangle argument. The
<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 2: Options for CheckList environments (and \CheckListSet)
Y1K problems are resolved. 31.12.999
Y2K problems are resolved. ...................................................(John) 31.12.1999

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
<th>Who</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>Repair all programs</td>
<td>John</td>
<td>31.12.999</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. 31.12.65535

- (Task ii) Build Y65K-proof time machine. ≈ 2500AD
- Use time machine from Task ii if problem persists.

Listing 1: Comprehensive checklist example
checklist types to which the status is added, are provided by the \texttt{⟨types⟩} argument, a comma-separated list. The \texttt{⟨symbol⟩} is \LaTeX\ code of a symbol that is put on top of the checklist type’s symbol. The \texttt{⟨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 \texttt{⟨isclosed⟩} indicates that the \texttt{⟨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.

Example  The following example shows how to define a ‘bug’ type.

```latex
\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}
```

4.2 Defining Checklist Layouts

\CheckListDeclareLayout{⟨name⟩}{⟨fields⟩}{⟨begin⟩}{⟨end⟩}

Using this macro, you can add a new checklist layout. The \texttt{⟨begin⟩} and \texttt{⟨end⟩} part is similar to a \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.

\CheckListDefineFieldFormat{⟨layout⟩}{⟨field⟩}{⟨code⟩}

After the new type has been added, for each field in the comma-separated \texttt{⟨fields⟩}, this macro must be used to define how a field is formatted. The \texttt{⟨code⟩} 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{\langle name \rangle}{\langle base \rangle}{\langle fields \rangle}

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{\langle status \rangle}

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]{\langle text \rangle}

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{\langle label \rangle}

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{\langle label \rangle}.

\CheckListDisplayDeadline{\langle status \rangle}{\langle deadline \rangle}

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{\langle closed? \rangle}{\langle passed? \rangle}{\langle deadline \rangle}

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.
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{⟨name⟩}{⟨default⟩}

This macro introduces a new entry option named ⟨name⟩ and with the given ⟨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.
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

\CheckListFilterClosed [(types)]

This macro sets up a filter that hides all checklist entries whose status is closed.
Through the optional (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.

\CheckListFilterClosed
\begin{CheckList}{Task}
\Task{open}{Open task}
\Task{started}{Started task}
\Task{done}{Done task}
\Task{dropped}{Dropped task}
\end{CheckList}

\CheckListFilterValue [(types)] {⟨field⟩} {⟨value⟩}

This macro sets up a filter that hides all checklist entries whose ⟨field⟩ has a value
that is unequal ⟨value⟩.

\CheckListFilterValue(who)(John)
\begin{CheckList}{Task}
\Task[who=John]{open}{John’s task}
\Task[who=Mary]{open}{Mary’s task}
\end{CheckList}
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\text{comp}\rangle equals ‘<’), equal (if \langle\text{comp}\rangle equals ‘=’), or after (if \langle\text{comp}\rangle equals ‘>’) the given \langle\text{date}\rangle. The \langle\text{date}\rangle must be in the format selected for input dates (see the input-dates option). If \langle\text{filter-inv}\rangle is true, then checklist entries whose deadline does not obey the format for input dates are filtered out. Otherwise, if \langle\text{filter-inv}\rangle is false, these checklist entries are not filtered out.

```latex
\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.

```latex
\begin{CheckList}
\Task[who=John]{done}{John’s task}
\Task[who=Mary]{done}{Mary’s task}
\Task[who=Mary]{open}{Mary’s open task}
\end{CheckList}
```

Filters are local to the \LaTeX group in which they are set up. In particular, if a filter is set up inside an environment, then the filter is no longer effective after the environment.
\begin{CheckList}{Goal}
\Goal[who=Mary]{achieved}{Mary's goal}
\begin{CheckList}{Task}
\CheckListFilterValue{who}{Mary}
\Task[who=Mary]{done}{Mary's task}
\Task[who=John]{done}{John's task}
\end{CheckList}
\Goal[who=John]{achieved}{John's goal}
\end{CheckList}
\CheckListFilterReset
\begin{CheckList}{Goal}
\Goal[who=Mary]{achieved}{Mary's goal}
\end{CheckList}
\CheckListFilterReset
\begin{CheckList}{Task}
\CheckListFilterValue{who}{John}
\Task[who=John]{open}{John's task}
\Task[who=Mary]{open}{Mary's task}
\end{CheckList}
\begin{CheckList}{Task}
\CheckListFilterReset
\begin{CheckList}{Task}
\CheckListFilterValue{who}{John}
\Task[who=John]{open}{John's task}
\Task[who=Mary]{open}{Mary's task}
\end{CheckList}
\end{CheckList}
\CheckListFilterReset

5.3 The Generic Filter Interface

Filters can also be set up programmatically.

\CheckListSetFilter[(\textit{types})]{(\textit{fields})}{(\textit{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, …) 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{\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 Asciilist environment. The \texttt{typed-checklist} package provides a syntax for this when the package is loaded with the
withAsciilist=true option. The syntax is illustrated with the following snippet, a transformed version of the example in Section 3.3:

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

For each checklist type \textit{⟨type⟩} (added by \texttt{\CheckListAddType}), an \texttt{AsciiList} environment \textit{⟨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{tabu} package for layouting the tables. The default can be changed through the \texttt{tablepkg} package option. The following values are available:

| \texttt{ltablex} | This option uses the \texttt{ltablex} package. |
| \texttt{tabu} | This option uses the \texttt{tabu} package, which is the default. That is, specifying this option does not change the package behavior. |
| \texttt{tabularx} | This option uses the \texttt{tabularx} package from the \LaTeX{} core. When using this table type, keep in mind that \texttt{tabularx} tables must fit onto a single page. |
| \texttt{xltabular} | This option uses the \texttt{xltabular} package, a successor of \texttt{ltablex}. |

Note: In the future, the default might change if the \texttt{tabu} package remains unmaintained.

### 7 Related Packages

The following \LaTeX{} packages provide related functionalities to the \texttt{typed-checklist} package.
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 todo and 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 xkeyval package for declaring package options as well as for option lists of entry types.

\RequirePackage{xkeyval}

We use the etoolbox package for simpler handling of lists.

\RequirePackage{etoolbox}

We use colors for deadlines, for instance.

\RequirePackage{xcolor}

10.2 Options

10.2.1 Checklist Options

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

\define@key[tchklist]{GlobalListOptions}{layout}{\%}
\ifinlist{#1}{\tchklist@ChecklistLayouts}{}{\PackageError{typed-checklist}{`#1' not a known checklist layout}{Known layouts are:\forlistloop{ }{\tchklist@@CheckListLayouts}}}\def\tchklist@@layout{#1}
\define@key[tchklist]{GlobalListOptions}{input-dates}{\%}
\ifinlist{#1}{\tchklist@@InputDateFormats}{}{\PackageError{typed-checklist}{`#1' not a known input date format}{Known formats are:\forlistloop{ }{\tchklist@@InputDateFormats}}}\letcs\tchklist@inputdate@order{tchklist@dateorder@#1}
\letcs\tchklist@inputdate@sep{tchklist@dateformat@sep@#1}
\define@key[tchklist]{GlobalListOptions}{output-dates}{\%}
\ifinlist{#1}{\tchklist@@OutputDateFormats}{}{\PackageError{typed-checklist}{`#1' not a known output date format}{Known formats are:\forlistloop{ }{\tchklist@@OutputDateFormats}}}\letcs\tchklist@@dateoutput@use{tchklist@dateoutput@use@#1}
\define@boolkey[tchklist]{GlobalListOptions}{strict-dates}[true]{\%}
\ifbool{tchklist@GlobalListOptions@strict-dates}{\let\tchklist@@faileddate=tchklist@DateFailStrict}{\let\tchklist@@faileddate=tchklist@DateFailLax}

10.2.2 Checklist Entry Options

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

\CheckListAddEntryOption{⟨option⟩}{⟨default⟩}
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

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

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

\CheckListDefaultLayout

The \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

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

\newcommand*\tchklst@ChecklistTypes{}

\CheckListAddType

The \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}%
Save the symbol of the new type.
\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 \texttt{\tchklst@IntroduceTypeHook\{⟨cmd⟩\}} macro introduces \texttt{⟨cmd⟩} for all existing checklist types (first code line) as well as for all checklist types defined afterwards (second code line).

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

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

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

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

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

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

\section{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 \texttt{⟨status⟩} for a given comma-separated list of checklist \texttt{⟨types⟩}. The \texttt{⟨symbol⟩} is \LaTeX code of a symbol that is put on top of the checklist type's symbol. The \texttt{⟨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 \texttt{⟨isclosed⟩} indicates that the \texttt{⟨status⟩} corresponds to the entry not yet being closed.

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

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

\forcsvlist

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

Some argument checking up front.

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

\tchklst@symbolcombine{⟨symbol1⟩}{⟨symbol2⟩} macro combines two symbols, ⟨symbol1⟩ and ⟨symbol2⟩.

\CheckListIfClosed The \CheckListIfClosed{⟨status⟩}{⟨iftrue⟩}{⟨iffalse⟩} macro expands to ⟨iftrue⟩, if the ⟨status⟩ of an entry in the current checklist is a “closed” one (see the documentation for \CheckListAddStatus for details). Otherwise, the macro expands to ⟨iffalse⟩.
10.6 Checklist Layouts

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

\newcommand*{\tclst@ChecklistLayouts}{}

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}{\tclst@ChecklistLayouts}{% PackageError{typed-checklist}{% Checklist layout `#1' already declared}{}}{}
  \listadd{\tclst@ChecklistLayouts}{#1}% Save the \langle fields \rangle list of the new layout.
  \csdef{tclst@ChecklistLayoutFields@#1}{}% \forcsvlist{\listcsadd{tclst@ChecklistLayoutFields@#1}}{#2}% Save the \langle begin \rangle and \langle end \rangle code of the new layout.
  \csdef{tclst@ChecklistLayoutBegin@#1}{#3}%  \csdef{tclst@ChecklistLayoutEnd@#1}{#4}%
}

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{tclst@ChecklistFormat@#2@##1}{% \csletcs{tclst@ChecklistFormat@#1@##1}{tclst@ChecklistFormat@#2@##1}}{}}% \dolistcsloop{tclst@ChecklistLayoutFields@#2}%}
The `\CheckListDefineFieldFormat{⟨layout⟩}{⟨field⟩}{⟨code⟩}` macro defines the \langle code \rangle to be used for displaying the given \langle field \rangle (or fields) in a checklist of the given \langle layout \rangle. Multiple fields can be displayed by specifying \langle field \rangle in the form \langle field \rangle_1 + \ldots + \langle field \rangle_n.

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

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

\newcommand{\tchklst@deffieldmacro}[3]{% \begingroup Get number of properties (‘+’-separated) in \langle field \rangle into \@tempcnta. \@tempcnta=0\relax \def\do##1{\advance\@tempcnta by 1\relax} \tchklst@dopsvlist{#2} \edef\do{\endgroup \csundef{#1} \noexpand\newcommand\expandonce{\csname #1\endcsname} \[\the\@tempcnta]{\unexpanded{#3}}} \do \tchklst@usefieldmacro

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

\newcommand{\tchklst@usefieldmacro}[3][\@firstofone]{% \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}

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

\newcommand{\tchklst@CheckLayout}[1]{% \xifinlist{#1}{\tchklst@ChecklistLayouts}{}{\PackageError{typed-checklist}}}
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.

\newcommand*{\CheckListSetFilter}[3][*]{% 
  \ifstrequal{#1}{*}{% 
    {\forlistloop{\tchklst@SetFilter{#2}{#3}}{\tchklst@ChecklistTypes}}% 
  }{% 
    {\forcsvlist{\tchklst@SetFilter{#2}{#3}}{#1}}% 
  }% 
}\tchklst@SetFilter{⟨fields⟩}{⟨code⟩}{⟨type⟩} macro defines a filter for a single type.

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

\newcommand*{\CheckListFilterValue}[3][*]{% 
  \CheckListSetFilter[#1]{#2} %
  \ifstrequal{##1}{#3}{\\togglefalse{display}}}% 
\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).

\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}}}
If parsing \texttt{\langle 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\texttt{\langle types \rangle}} macro resets the filters for all checklist types in the comma-separated list \texttt{\langle types \rangle}. If \texttt{\langle types \rangle} is omitted or equals '⋆', then the filters for all checklist types are reset.

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

### 10.8 Checklist and Entry Definition

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

We check whether the provided \texttt{\langle 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 \texttt{\langle type \rangle} \texttt{Goal}.

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

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

\begin{verbatim}
\newcommand{\chklist@entry}[3][]{%
 \begingroup
 First check for a valid status. There is no need to check for a valid type, because the surrounding \texttt{CheckList} environment already does this.
 \endgroup}
\newcommand{\chklist@CheckTypeStatus}{\chklist@cur@type}{#2}%
\end{verbatim}

Parse the options.

\begin{verbatim}
\setkeys{tchklst}{Entry}{#1}%
\def\cmdtchklst@Entry@status{#2}%
\def\cmdtchklst@Entry@description{#3}%
\end{verbatim}

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

\begin{verbatim}
\newtoggle{display}\toggletrue{display}%
\def\do##1{%
 \tchklst@usefieldmacro
 \iftoggle{display}{}{\listbreak}}%
\dolistcsloop{tchklst@ChecklistFilters}{}%
\end{verbatim}

Show the fields of the entry in the order they were given. The whole entry is first collected in a macro (\texttt{\chklist@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.

\begin{verbatim}
\def\chklist@entry{%
 \iftoggle{display}{%\def\do##1{%
 \tchklst@usefieldmacro[appto=]{\chklist@@entry}{#1}}%
 \dolistloop\chklist@cur@fields}{}%
\chklist@@entry}
\end{verbatim}

The \texttt{\chklist@dopsvlist} macro parses a `+`-separated list.

\begin{verbatim}
\DeclareListParser{\chklist@dopsvlist}{+}
\end{verbatim}

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

\begin{verbatim}
\newcommand{\CheckListSigned}{\@dblarg\tchklst@signed}\
\newcommand{\tchklst@signed}[2][]{%
 \ifstrempty{#1}{\nobreak\hfill\null}{
 \leavevmode\unskip\nobreak\hfill\penalty0\hskip0.25em
 \hbox{#2}}}
\end{verbatim}
10.9 Deadlines

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

\CheckListDisplayDeadline The \CheckListDisplayDeadline{\langle status\rangle}\{\langle deadline\rangle\} formats a \langle deadline\rangle dependent on the \langle status\rangle and the current date.
\newcommand\CheckListDisplayDeadline[2]{% Try to parse \langle deadline\rangle as a date.
  \CheckListParseDate{#2}{\tchklst@DisplayDeadline@i}
  % \begin{macrocode}
  \def\tchklst@@args{}%
  \CheckListIfClosed{#5}{\appto\tchklst@@args{{true}}}{{false}}%
  \tchklst@ifafter{#1}{#2}{#3}{\appto\tchklst@@args{{true}}}{{false}}%
  \expandafter\CheckListHighlightDeadline\tchklst@@args
\CheckListHighlightDeadline The \CheckListHighlightDeadline{\langle closed?\rangle}\{\langle passed?\rangle\}{\langle deadline\rangle} macro highlights the given \langle deadline\rangle based on the two Boolean (‘true’ or ‘false’) arguments \langle done?\rangle (whether the respective checklist entry has been completed) and \langle passed?\rangle (whether the deadline has passed). One can \renewcommand this macro to change the deadline highlighting.
\newcommand\CheckListHighlightDeadline[3]{%
The following auxiliary macro just swaps the first two arguments, \textit{text} and \textit{sep} of \texttt{tchklist@splitapply} such that the now first argument \textit{sep} can be expanded more easily.

\begin{verbatim}
\newcommand*{\tchklist@splitapply@i}[2]{\tchklist@splitapply{#2}{#1}}
\end{verbatim}

\texttt{tchklist@ifafter} The \texttt{tchklist@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}!

\begin{verbatim}
\newcommand*{\tchklist@ifafter}[3]{% 
  \ifnum\the\year\two@digits\month\two@digits\day% 
    \numexpr#1\two@digits{#2}\two@digits{#3}\relax 
    \expandafter\@firstoftwo \else \expandafter\@secondoftwo \fi{#6}{#7}}% 
\end{verbatim}

\texttt{CheckListDateCompare} 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 \textit{<}, \textit{=} and \textit{>}. 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⟩}.

\begin{verbatim}
\newcommand{\CheckListDateCompare}[6]{% 
  \bgroup \def\do##1##2##3##4\egroup 
    \tchklist@DateCompare{#1}{#2}{##1}{##2}{##3}{#4}{#5}{#6}}% 
\end{verbatim}

\texttt{tchklist@DateCompare} The \texttt{tchklist@DateCompare}\{⟨date⟩\}{⟨comp⟩}\{⟨y⟩\}{⟨m⟩}\{⟨d⟩\}{⟨iftrue⟩}{⟨iffalse⟩}{⟨iffail⟩} macro is the internal counterpart to \texttt{CheckListDateCompare}. The difference is that the former expects \textit{⟨refdate⟩} to already be parsed into \textit{⟨y⟩}, \textit{⟨m⟩}, and \textit{⟨d⟩}.

\begin{verbatim}
\newcommand*{\tchklist@DateCompare}[8]{% 
  \bgroup \def\do##1##2##3##4\egroup \tchklist@DateCompare{#1}{#2}{#3}{#4}{#5}{#6}}% 
\end{verbatim}

\texttt{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

\begin{verbatim}
\newcommand{\CheckListParseDate}[3]{% 
  \ifnum\the\year\two@digits\month\two@digits\day% 
    \numexpr#1\two@digits{#2}\two@digits{#3}\relax 
    \expandafter\@firstoftwo \else \expandafter\@secondoftwo \fi{#6}{#7}}% 
\end{verbatim}
\texttt{\textbackslash cmd\{(year\}\}{\{(month\}\}{\{(day\}\}{\{(date\}\} (i.e., \texttt{\textbackslash cmd} must take four arguments). Otherwise, the macro expands to \texttt{\textbackslash fail\{(date\}}} (i.e., \texttt{\textbackslash fail} must take one argument).

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

Dates have three components and all must be positive numbers.

256 (3){\tchklst@ifPositive}

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

257 {\expandafter#2\tchklst@inputdate@order}
258 (#3)

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

259 (#1)}

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

\texttt{\textbackslash tchklst@@InputDateFormats}\texttt{\textbackslash tchklst@@OutputDateFormats} The \texttt{\tchklst@@InputDateFormats} and \texttt{\tchklst@@OutputDateFormats} macros collect the list of known input date formats and, respectively, output date formats. Both are etoolbox lists. Initially, both lists are empty.

260 \newcommand{\tchklst@@InputDateFormats{}}
261 \newcommand{\tchklst@@OutputDateFormats{}}

\texttt{\textbackslash tchklst@registerdateinputfmt} The \texttt{\tchklst@registerdateinputfmt\{(name\}\}{\{reorder\}\}{\{sep\}}} macro registers a date input format under the given \texttt{name}. The format uses \texttt{sep} as the separator symbol between dates' components and uses the \texttt{reorder} code to reorder the components from given dates to the ordering “\texttt{year\-month\-day\}”.

262 \newcommand{\tchklst@registerdateinputfmt[3]{%}
263 \listadd{\tchklst@@InputDateFormats(#1}%
264 \csgdef{tchklst@dateorder@#1}##1##2##3{#2}%
265 \csgdef{tchklst@dateformat@sep@#1}{#3}}

The following registers some typical date input formats.

266 \tchklst@registerdateinputfmt{d.m.y}{(#3}{(#2}{(#1)}
267 \tchklst@registerdateinputfmt{m/d/y}{(#2}{(#3}{(#1)}
268 \tchklst@registerdateinputfmt{y-m-d}{(#1}{(#2}{(#3)}

\texttt{\textbackslash tchklst@registerdateoutputfmt} The \texttt{\tchklst@registerdateoutputfmt\{(name\}\}{\{use\}}} macro registers a date output format under the given \texttt{name}. The \texttt{use} code may take four parameters: \texttt{year}, \texttt{month}, \texttt{day}, and \texttt{deadline}. Hence, \texttt{use} can use the original \texttt{deadline} as well as the decomposed form into year, month, and day.

269 \newcommand{\tchklst@registerdateoutputfmt[2]{%}
270 \listadd{\tchklst@@OutputDateFormats(#1}%
271 \csgdef{tchklst@dateoutput@use@#1}##1##2##3##4{#2}}

The following registers some typical date output formats.

272 \tchklst@registerdateoutputfmt{same}{(#4}
273 \tchklst@registerdateoutputfmt{datetime}
The following `\numexpr` looks a bit unfamiliar but it computes the modulo, given that integer division rounds the result.

The remainder of this section defines generic auxiliary macros for deadline parsing.

The `\tchklist@splitapply{\langle\text{text}\rangle}{\langle\sep\rangle}{\langle\n\rangle}{\langle\cmd\rangle}{\langle\fail\rangle}` macro is a generic macro for parsing a list-like `\langle\text{text}\rangle` of fixed length `\langle\n\rangle`, whose components satisfy `\langle\cond\rangle` (a macro with one argument) and are separated by `\langle\sep\rangle`. If \(c_1\) to \(c_n\) are the components of `\langle\text{text}\rangle`, i.e., if `\langle\text{text}\rangle=c_1\langle\sep\rangle\cdots\langle\sep\rangle c_n`, then the macro expands to `\langle\cmd\rangle\{c_1\}\cdots\{c_n\}`. If `\langle\text{text}\rangle` has less than `\langle\n\rangle` or more than `\langle\n\rangle` components, or if at least one of the components does not satisfy `\langle\cond\rangle`, then the macro expands to `\langle\fail\rangle`.

The `\tchklist@splitapply{\langle\text{text}\rangle}{\langle\sep\rangle}{\langle\cond\rangle}{\langle\cmd\rangle}{\langle\fail\rangle}` macro recursively grabs `\langle\prefix\rangle`s from the remaining `\langle\suffix\rangle` and appends them to `\langle\cmd\rangle`. The latter accumulates `\langle\cmd\rangle` and the already parsed components.

Check whether `\langle\cond\rangle` holds for `\langle\prefix\rangle` first.

If \(k > 0\), i.e., then `\langle\suffix\rangle` contains the last \(k\) components of `\langle\text\rangle` plus a trailing `\langle\sep\rangle`.

```latex
\tchklist@splitapply{\langle\text\rangle}{\langle\sep\rangle}{\langle\cond\rangle}{\langle\cmd\rangle}{\langle\fail\rangle}
\tchklist@splitapply{\langle\text\rangle}{\langle\sep\rangle}{\langle\cond\rangle}{\langle\cmd\rangle}{\langle\fail\rangle}
```
If \langle suffix \rangle is empty, then \langle text \rangle contained too few components and, hence, expand to \langle fail \rangle. Otherwise recurse.

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

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.

\begin{verbatim}
\ifstrempty{##4}
{##2{##3}}
\else\tchklst@split@@rec{#3-1}{#5}#1#2\relax}
\end{verbatim}

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

\begin{verbatim}
\tchklst@ifPositive[1]{% 
 \ifcat _\ifnum\z@<0#1_\else A\fi 
 \expandafter\@firstoftwo \else \expandafter\@secondoftwo \fi}
\end{verbatim}

The \texttt{\tchklst@ifPositive} macro expands to \texttt{\iftrue} if \langle text \rangle is a positive number and expands to \texttt{\iffalse} 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.

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

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 \texttt{\import*} or \texttt{\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}{%}
\CheckListAddStatus{Artifact,\footnotesize?}{%}

32
The following provides the default set of checklist layouts.

10.11 Default Checklist Layouts

We use the marginnote package to display deadlines in the list layout.

\RequirePackage{marginnote}

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

\CheckListDeclareLayout{list}{status,label,description,who,deadline+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.

\CheckListDefineFieldFormat{list}{status}%
{\item[\CheckListStatusSymbol{#1}]}

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

\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}}}}}

Show the description, with leading spaces removed.

\CheckListDefineFieldFormat{list}{description}%
{\ignorespaces#1\relax}

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

\CheckListDefineFieldFormat{list}{who}%
\CheckListSigned[#1]{(\emph{#1})}

Show the deadline of the entry in the margin, if the deadline option is given in \{options\}.

The following \unskip prevents \marginnote from breaking an overfull margin text at it’s very beginning, which meant that the margin text would vertically be placed below the actual entry (see also https://tex.stackexchange.com/questions/117695/).

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

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

The table layout formats the checklist as a table, one row per checklist entry. The NC field just inserts the column separator.
The following macros define the package-specific table code.

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

\begin{verbatim}
\newcommand\tchklst@inittab@tabu{\RequirePackage{longtable,tabu}}
\newcommand\tchklst@begintab@tabu{\tabulinesep=0.5ex\relax
\def\tchklst@@endhead{\endhead}\
\let\chklst@@newline\c@oddrow\chklst@@newline}
\newcommand\tchklst@endtab@tabu{\endlongtabu}
\end{verbatim}

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.

\begin{verbatim}
\newcommand\tchklst@inittab@xltabular{\RequirePackage{array,xltabular}}
\newcommand\tchklst@begintab@xltabular{\setlength{\extrarowheight}{0.5ex}\
\let\tchklst@@endhead\relax\def\tchklst@@newline\c@oddrow\chklst@@newline}
\preto\XLT@ii@TX@endtabularx{\toks@\expandafter{\the\toks@\tchklst@@newline}}
\xltabular{\linewidth}{|c|X|l|r|}}
\newcommand\tchklst@endtab@xltabular{\endxltabular}
\end{verbatim}

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

\begin{verbatim}
\newcommand\tchklst@inittab@tabularx{\RequirePackage{array,tabularx}}
\newcommand\tchklst@begintab@tabularx{\let\tchklst@@endhead\relax\
\setlength{\extrarowheight}{0.5ex}\
\let\tchklst@@newline\c@oddrow\chklst@@newline}
\preto\TX@endtabularx{\toks@\expandafter{\the\toks@\tchklst@@newline}}
\tabularx{\linewidth}{|c|X|l|r|}}
\newcommand\tchklst@endtab@tabularx{\endtabularx}
\end{verbatim}

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

\begin{verbatim}
\newcommand\tchklst@inittab@ltablex{\RequirePackage{ltablex}}
\newcommand\tchklst@begintab@ltablex{\let\tchklst@@endhead\relax\
\setlength{\extrarowheight}{0.5ex}\
\let\tchklst@@newline\c@oddrow\chklst@@newline}
\preto\TX@endtabularx{\toks@\expandafter{\the\toks@\tchklst@@newline}}
\tabularx{\linewidth}{|c|X|l|r|}}
\newcommand\tchklst@endtab@ltablex{\endtabularx}
\end{verbatim}
The following fixes a bug in ltablex, 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.}{}
\let\tchklst@begintab@ltablex=\tchklst@begintab@tabularx
\let\tchklst@endtab@ltablex=\tchklst@endtab@tabularx

10.12 Package Options

10.12.1 Package-Only Options

The withAsciilist option enables support for the asciilist package.
\define@boolkey[tchklist]{PackageOptions}{withAsciilist}{true}{

The tablepkg option specifies which table package is used for layouting checklists in the table layout.
\define@choicekey[tchklist]{PackageOptions}{tablepkg}{\val}{
  \letcs{tchklist@@inittab}{tchklist@inittab@\val}
  \letcs{tchklist@@begintab}{tchklist@begintab@\val}
  \letcs{tchklist@@endtab}{tchklist@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.
\define@boolkey[tchklist]{PackageOptions}{onecounter}{true}{

10.12.2 Processing Options

Set option defaults and then load the given options.
\ExecuteOptionsX[tchklist]<PackageOptions,GlobalListOptions>{
  withAsciilist=false,
  tablepkg=tabu,
  onecounter=true,
  layout=list,
  input-dates=d.m.y,
  output-dates=same,
  strict-dates=false,
}
\ProcessOptionsX[tchklist]<PackageOptions,GlobalListOptions>\relax
\tchklist@inittab
10.12.3 Labels

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

\newcommand*\CheckListDefaultLabel[1][]{\ifstrempty{#1}{}{\ifbool{tchklst@onecounter}{{\refstepcounter{tchklst@entryID}}}{{\refstepcounter{tchklst@entryID\tchklst@cur@type}}}\label{#1}}}\tchklst@NewEntryCounter 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⟩.

\newcommand*\tchklst@NewEntryCounter[2][@]{\newcounter{tchklst@entryID#1#2}\setcounter{tchklst@entryID#1#2}{0}\ifstrempty{#2}{\csgdef{thetchklst@entryID#1#2}{\tchklst@cur@type~\protect\textsc{\roman{tchklst@entryID#1#2}}}}{\csgdef{thetchklst@entryID#1#2}{#2~\protect\textsc{\roman{tchklst@entryID#1#2}}}}}\tchklst@ChkListEntry The \tchklst@ChkListEntry{⟨item-macro⟩}{⟨content⟩} macro can be used as a parameter to \AsciiListEndArg of the \texttt{asciilist} package in order to allow for checklist entries in an AsciiList.

\newcommand*\tchklst@ChkListEntry[2][@]{\tchklst@ChkListEntry@i{#1}#2@undefined}The used auxiliary macros serve the purpose of parsing the input and have the following signatures:

- \tchklst@ChkListEntry@i{⟨item-macro⟩}{⟨status+opts⟩}{⟨descr⟩}

10.12.4 asciilist

If the package is loaded with \texttt{asciilist} support...

\iftchklst@withAsciilist First, we load the package.
\RequirePackage{asciilist}
\tchklst@ChkListEntry The \tchklst@ChkListEntry{⟨item-macro⟩}{⟨content⟩} macro can be used as a parameter to \AsciiListEndArg of the \texttt{asciilist} package in order to allow for checklist entries in an AsciiList.

\newcommand*\tchklst@ChkListEntry[2][@]{\tchklst@ChkListEntry@i(#1)#2@undefined}The used auxiliary macros serve the purpose of parsing the input and have the following signatures:

- \tchklst@ChkListEntry@i{⟨item-macro⟩}{⟨status+opts⟩}{⟨descr⟩}
The \tchklst@RegisterAsciiTypeEnv{\langle type \rangle} registers an asciilist environment for the given checklist \langle type \rangle.

\newcommand*\tchklst@RegisterAsciiTypeEnv[1]{\AsciiListRegisterEnv{#1List}\tchklst@aux@OargAfter{\CheckList{#1}}}\endCheckList\AsciiListEndArg{\tchklst@ChkListEntry{\csname #1\endcsname}}}\tchklst@IntroduceTypeHook{\tchklst@RegisterAsciiTypeEnv}\fi

Change History

v0.1
  General: Initial version ............ 1
  Added \CheckListAddEntryOption macro ........................................ 18
v0.2
  General: Better handling of empty
  “who” ................................. 1
  \CheckListExtendLayout: Enabled extensible layouts ....................... 22
v1.2b
  \CheckListDefaultLayout:
  Enabled setting default
  checklist layouts .............. 19
v0.3
  General: Added deadline and label
  support ............................. 1
  General: Support for combining
  checklists with asciilist ...... 1
v1.3
v0.4
  General: Added “dropped” tasks .. 1
  v1.3
v0.4b
  General: Fix package dependencies
  \langle xcolor \rangle .................... 1
  \langle xcolor \rangle .................... 1
v0.5
  General: Added “dropped” artifacts 1
v1.3b
v0.6
  General: Added “dropped” artifacts 1
  General: Removed dependency on
  paralist package ............... 32
v1.3c
v0.7
  General: Indication of closed
  checklist entries ................ 1
v1.3d
v0.8
  General: First documented version 1
v1.4
v1.1
  General: Added definable layouts . 1
  \CheckListAddEntryOption:
  General: Fixed symbol for dropped
  tasks .......................... 31
v1.1b
v1.4
v1.1c
  General: Added milestone checklists 31
v1.2
v1.4
v1.4
v1.5
  General: Add nobreak to ‘who’ field in ‘list’ layout ............... 32
  Improve left alignment of entry text in list layout ............... 32
  Raggedright for labels in case of a narrow list .......... 32
  Raggedright for labels in case of narrow table display .......... 33
v1.5b
  General: Fix for list layout (changed to itemize) ............... 32
v1.5c
  General: Fix vertical placement of deadlines in narrow margins .... 33
v2.0
  \CheckListDisplayDeadline: More flexibility for deadline parsing and deadline printing . 27
  \CheckListHighlightDeadline: Show future deadlines of completed entries also in green 27
  \CheckListSet: Macro added ... 19
  \CheckListSetFilter: Added customizable filtering feature . 24
  \tchklst@entry: Simplified field display through
  \tchklst@usefieldmacro .... 26
  \tchklst@ifafter: Reversed argument list .................... 28
  \tchklst@symbolcombine: Ignore height of first argument ...... 21
  Switched combining order ...... 21
  General: Add package option for using per-type entry counters . 35
  Allow empty list-type checklists 32
  Made “who” column left-aligned in table layout ............... 33
  Package option ‘tablepkg’ added 34

Index

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 ...... 463, 464, 465, 466
\[ ........................... 369, 370
\[ ........................... 349, 381
\{" ........................... 349, 381
\advance ..................... 118
\allowbreak .................. 348, 380
\appto .................. 201, 221, 223
\ArrowBoldRightStrobe . 328, 333
\Artifact .......................... 5
\AsciiListEndArg ............. 471
\AsciiListRegisterEnv ...... 468
\AtBeginDocument ............. 310
\begin ......................... 214
\begingroup ........... 116, 126, 187
\bfseries ..................... 322
\bgroup .................. 157, 240, 245, 339
\bigcirc ..................... 311
\bigtriangleup .................. 313
\CheckList ................. 469
\CheckList (environment) . 3, 173
\CheckListAddEntryOption 11, 27, 30, 31, 32
\CheckListAddStatus 6, 63, 315, 316, 317, 318, 319, 321, 324, 325, 327, 329, 331, 332, 334, 335
\CheckListAddType 6, 38, 311, 312, 313, 314
\CheckListDateCompare ..... 239
\CheckListDeclareLayout . 9, 94, 104, 337, 363, 365
\CheckListDefaultLabel 10, 345, 378, 441
\CheckListDefaultLayout ... 35
\CheckListDefineFieldFormat 9, 113, 341, 343, 353, 355, 357, 361, 364, 374, 375, 376, 382,
null
<table>
<thead>
<tr>
<th>Command</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf</td>
<td>371, 372</td>
</tr>
<tr>
<td>\textcolor</td>
<td>228, 229, 230, 351, 352</td>
</tr>
<tr>
<td>\textit</td>
<td>293, 356</td>
</tr>
<tr>
<td>\textsc</td>
<td>452, 454</td>
</tr>
<tr>
<td>\the</td>
<td>123, 233, 282, 284, 286, 400, 408</td>
</tr>
<tr>
<td>\tiny</td>
<td>316, 322, 333</td>
</tr>
<tr>
<td>\togglefalse</td>
<td>152, 155, 161, 162</td>
</tr>
<tr>
<td>\toggletrue</td>
<td>192</td>
</tr>
<tr>
<td>\toks@</td>
<td>400, 408</td>
</tr>
<tr>
<td>\topsep</td>
<td>339</td>
</tr>
<tr>
<td>\twodigits</td>
<td>233, 234, 247, 249</td>
</tr>
<tr>
<td>\TX@endtabularx</td>
<td>408, 412</td>
</tr>
<tr>
<td>\underbar</td>
<td>351</td>
</tr>
<tr>
<td>\unexpanded</td>
<td>123</td>
</tr>
<tr>
<td>\unskip</td>
<td>210, 359</td>
</tr>
<tr>
<td>\val</td>
<td>422, 424, 425, 426</td>
</tr>
<tr>
<td>\wd</td>
<td>85</td>
</tr>
<tr>
<td>\xifinlist</td>
<td>137</td>
</tr>
<tr>
<td>\XLT@i@TX@endtabularx</td>
<td>399</td>
</tr>
<tr>
<td>\xltabular</td>
<td>401</td>
</tr>
<tr>
<td>\XSolid</td>
<td>316, 317</td>
</tr>
<tr>
<td>\year</td>
<td>233</td>
</tr>
<tr>
<td>\x@</td>
<td>307</td>
</tr>
</tbody>
</table>