The exframe Package Reference Manual

Niklas Beisert

Institut für Theoretische Physik
Eidgenössische Technische Hochschule Zürich
Wolfgang-Pauli-Strasse 27, 8093 Zürich, Switzerland

nbeisert@itp.phys.ethz.ch

v3.5.3: 2026-03-23

https://ctan.org/pkg/exframe
https://github.com/nbeisert/latex-pkg-nb

Abstract

exframe is a LaTeX 2_ε package which provides a general purpose framework to describe and typeset exercises and exam questions along with their solutions. The package features mechanisms to hide or postpone solutions, to assign and handle points, to collect problems on exercise sheets, to store and use metadata and to implement a consistent numbering. It also provides a very flexible interface for configuring and customising the formatting, layout and representation of the exercise content.

Contents

1 Introduction

This package provides a framework to describe and typeset exercises (homework problems, classroom exercises, quizzes, exam questions, exercise questions in books and lecture notes, ...) and their solutions or answers. The aim of this package is to set up a few LaTeX environments into which questions and corresponding answers can be filled conveniently. The main task of the package is to manage the text and data that are provided in the source document, perform some common operations on them, and then output the content appropriately. The package has the following goals, tasks and features:

- The package is designed with generality in mind. It is meant to be usable in many different situations. The primary target is science and education, but it may well be useful in other areas.
- The package defines a basic functional layout for the output and provides many options to reshape the layout and formatting according to the author's needs and wishes.
- The package can handle two layers of exercises: main problems and subproblems. The use of subproblems is optional.
- The display of solutions can be configured: Solutions can be hidden for a hand-out version of exercise sheets. When displayed, they may appear immediately, collectively after the problem, at the end of each sheet or at some manually defined location.
- The package can handle exercise sheets which combine several exercise problems: A LaTeX document can consist of an individual sheet or of a collection of sheets (e.g. spanning a lecture course). In the latter case, the document files can be set up such that single sheets as well as a collection of all sheets can be compiled; the package childdoc may be of assistance.
- The package can handle points to be credited: Points will be displayed according to the layout. Overall points for a problem or a sheet can be added automatically. Points can also be stored and used elsewhere.
- The package provides an interface to specify exercise metadata (author, source, ...): Some basic types of metadata are predefined and more specific metadata categories can be added.
- The package can use alternative counters for equations within solutions (and problems). This is to ensure a consistent numbering independently of whether solutions are output or not.

2 Usage

To use the package exframe add the command

```
\usepackage{exframe}
```

to the preamble of the LaTeX document.

2.1 Exercise Environments

The package provides four environments to describe the main entities of exercise problems. Additional information on the exercises can be provided in the optional arguments to these environments which will be discussed in the following sections. Furthermore, a limited set of commands is provided for control and extra features, see the sections below for details.

`problem` (*env.*) The `problem` environment describes an exercise problem:

```
\begin{problem}[opts]
  problem text and subproblems
\end{problem}
```

As one of the many available options *opts*, one can provide a title for the exercise by specifying `title={`*title*`}`. If no title is given, the problem number will be displayed instead. See section 2.3 and section 2.4 for a description of the available options.

`subproblem` (*env.*) The `subproblem` environment describes a subproblem, part or an individual question of an exercise problem:

```
\begin{subproblem}[opts]
  subproblem text
\end{subproblem}
```

A `subproblem` environment must be contained within a `problem` environment (however, a `problem` block need not contain `subproblem` blocks).

`solution` (*env.*) The `solution` environment describes the solution to a problem or a subproblem:

```
\begin{solution}[opts]
  solution text
\end{solution}
```

A `solution` environment should be at the end of a `subproblem` or `problem` environment (it is not mandatory to provide a `solution`). It can be contained within the corresponding environment or it can follow it (where needed, the option `forproblem` associates a `solution` following a `subproblem` to the encompassing `problem` rather than the preceding `subproblem`). Depending on the choice of solution display, see section 2.2, the output may have a slightly different layout. In terms of logic, it is preferred to define a solution *within* the corresponding environment; this may also have some technical advantages and produce a slightly better result in terms of layout.

`sheet` (*env.*) The `sheet` environment describes an exercise sheet:

```
\begin{sheet}[opts]
  sheet text and problems
\end{sheet}
```

A sheet typically contains one or several problems (it is not mandatory to group problems into a `sheet`). There may or may not be additional auxiliary text introducing the problems. A header will be added to the sheet according to the specified layout.

2.2 Solution and Problem Display

There are several options to control the output of solutions and of problems.

`solutions` Most importantly, the display of solutions can be disabled or enabled altogether:

```
\exercisesetup{solutions[=true|false]}
```

Solutions are hidden by default, and their display needs to be activated explicitly (it suffices to specify the option `solutions` without the value `true`). It is also possible to control the display by an analogous package option `solutions`, see section 2.8 for further information.

`\ifsolutions` `onlysolutions` The display of solutions is reflected by the conditional `\ifsolutions`. As the hiding of solutions is performed automatically, the conditional would typically be used to change some details, e.g. for adjusting titles:

```
\ifsolutions Solutions\else Exercises\fi
```

Alternatively, content to be processed only in solutions mode can be enclosed in an `onlysolutions` block:

```
\begin{onlysolutions}
...
\end{onlysolutions}
```

This structure can be useful to hide auxiliary text or material if all solution content is to be stripped from a source file, e.g. by an automated `sed` filter rule (doubling of backslashes required for `sed` as well as for shell script strings):

```
sed "/\\\\begin{solution}/,/\\\\end{solution}/d;"\
"/\\\\begin{onlysolutions}/,/\\\\end{onlysolutions}/d"
```

`solutionequation` As solutions can contain numbered equations while the display of solutions can be switched on and off, it is important to assign a different counter for equations within solutions in order for the equation numbers to be stable. A separate counter for equations within solutions is enabled by default. It can be disabled by:

```
\exercisestyle{solutionequation=false}
```

This option prepends the letter 'S' to equation numbers within solutions which are counted separately; the display can be configured differently, see section 2.6.

`solutionbelow` `\insertsolutions` The package allows to collect solutions and defer their display to particular locations:

```
\exercisestyle{solutionbelow=pos}
```

The available choices for *pos* are to display solutions where they are defined (`here`), defer them to the end of the current subproblem (`subproblem`), problem (`problem`) or sheet (`sheet`) or display them at a manually chosen location (`manual`). Note that typically solutions are defined at the end of a (sub)problem and therefore the choice `here` is similar to (`sub`)`problem`. The latter form, however, makes sure that a solution does not inherit the margin of the parent environment. The alternate modes `problem*` and `subproblem*` positions the solution *after* the (sub)problem environment such that it does not inherit any layout, but also no definitions made in the parent environment. In `manual` mode, all solutions are collected (with appropriate headers) until they are output by the directive `\insertsolutions`. If no solutions are stored in the buffer (or if the mode is not `manual`), `\insertsolutions` has no effect.

`\writesolutions` Another option to handle solutions is to write them to a file for later use. Writing to a file is initiated by:

```
\writesolutions[filename]
```

The optional argument describes the filename as *filename*`.sol`; no argument defaults to the main tex filename as `\jobname.sol`; the extension `.sol` can be customised by the configuration `extsolutions`. This mode overrides the `solutionbelow` behaviour described above; all subsequent solutions are written to the file. The file is closed by `\closesolutions` and the display of solutions returns to manual mode. It is not necessary to close a file as it will be closed automatically by reading from a file, writing to another file or by the end of the document.

`\readsolutions` Solutions are read from a file by:

\readsolutions[*filename*]

This command outputs a sectional title and reads the file via \input{*filename*.sol}.

solutionbuf
problembuf
subproblembuf

The package offers similar functionality to control the display of problems. In order to have any control over the content of problem and subproblem environments, the latter need to be read into an internal buffer. Reading of solutions and problems to internal buffers is activated or deactivated by:

\exercisesetup{solutionbuf[=true|false]}
\exercisesetup{problembuf[=true|false]}
\exercisesetup{subproblembuf[=true|false]}

By default, solution environments are read to an internal buffer, while the content of [sub]problem environments is processed directly by the TEX engine. Therefore, the below options to control the display of [sub]problem environments require the statement \exercisesetup{[sub]problembuf}.

problemmanual
\insertproblems

The immediate display of problem environments is controlled by:

\exercisestyle{problemmanual[=true|false]}

In the default automatic mode, problems are displayed directly where they are declared. In manual mode, problems are collected to an internal buffer, and only displayed by issuing \insertproblems.

Note that solution environments should be declared within the corresponding problem environment in order to preserve their appropriate association. The solution environment is then processed at the place where the problem environment is displayed, and it may (or may not) be deferred further.

\writeproblems
\readproblems

Problems can be written out to an external file for later usage. The functionality is analogous to solutions and uses the macros:

\writeproblems[*filename*]
\readproblems[*filename*]

The optional argument describes the filename as *filename*.prb; no argument defaults to the main tex filename as \jobname.prb; the extension .prb can be customised by the configuration extproblems.

disable
insertproblemselect
sertsubproblemselect

The display of a particular [sub]problem can be suppressed altogether by an optional argument:

\begin{[sub]problem}[disable]

This option can be exploited to automatically suppress certain classes of problems as follows: A hook function insert[sub]problemselect declared by:

\exerciseconfig{insert[sub]problemselect}[1]{*code*}

can call \set[sub]problemdata{disable} whenever a problem is to be suppressed. In order to decide, the optional argument of the [sub]problem environment is passed on to the hook function as the single argument. Note that the argument needs to be processed manually.

2.3 Metadata

In a collection of exercise problems it makes sense to keep track of metadata for the overall collection as well as for individual problems and potentially display some of them. The framework defines a standard set of metadata fields and offers functionality to add more specialised metadata fields.

`\exercisedata` Global metadata is specified by the command:

`\exercisedata{`*data*`}`

The argument *data* is a comma-separated list of metadata specifications in the form *key*`={`*value*`}`. The standard set of global metadata keys consists of:

- `author`: principal author(s) of the exercise collection; also invokes the LaTeX command `\author`; will be written to pdf documents.
- `title`: title of the exercise collection; also invokes the LaTeX command `\title`; will be written to pdf documents.
- `date`: date of the exercise collection; also invokes the LaTeX command `\date`; will be written to pdf documents.
- `subject`: subject area of the exercise collection; will be written to pdf documents.
- `keyword`: keyword(s) for the exercise collection; will be written to pdf documents.
- `course`: title of the course (class, lecture, module, ...) for the exercise collection.
- `institution`: institution (school, department, institute, university, ...) offering the course or exercise collection.
- `instructor`: instructor(s) for the course or exercise; this field refers to person(s) who organise the corresponding course or exercises whereas `author` refers to the principal creator of the material.
- `period`: period (year, season, date, term identifier, ...) of the corresponding course.
- `material`: type of material (exercises, homework assignments, exam, quizzes, solutions, ...).

`\defexercisedata` Additional custom fields for global metadata can be created with:

`\defexercisedata{`*key*`}`

`\getexercisedata` `\exercisedataempty` Global metadata should typically be specified somewhere at the top of the main document, and it can be inserted wherever needed. There are two commands to read and process metadata. To insert the value of metadata field *key* use:

`\getexercisedata{`*key*`}`

In some situations the output should depend on whether a metadata has been filled (e.g. to fill a default value or to display something else instead). This can be checked with the conditional:

`\exercisedataempty{`*key*`}{`*empty code*`}{`*filled code*`}`

The *empty code* is executed if no value or an empty value has been specified; otherwise the *filled code* is executed.

`sheet` `problem` The package offers a similar mechanism to describe and use metadata for sheets and problems:

```
\begin{sheet}[opts]
\begin{problem}[opts]
```

The argument *opt* is a comma-separated list which can contain metadata specifications in the form *key*={*value*}. The standard set of metadata keys for sheets consists of:

- `due`: indication of the due date for the exercise sheet.
- `handout`: indication of the handout date for the exercise sheet.
- `title`: specifies a title for the sheet; when reading value (see below), returns composed title; untitled sheets will be displayed by their number; title will be written to pdf documents.
- `rawtitle` (for reading only): contains the raw title as specified by `title`.
- `author`: author(s) of the sheet; will be written to pdf documents.
- `editor`: editor(s) of the sheet; this field refers to a person who makes adjustments to the sheet whereas `author` refers to the creator of the sheet.
- `editdate`: indication of the date when the sheet was last edited.

The standard set of metadata keys for problems consists of:

- `title`: specifies a title for the problem; when reading value (see below), returns composed title; untitled problems will be displayed by their number.
- `rawtitle` (for reading only): contains the raw title as specified by `title`.

`\defsheetdata` `\setsheetdata` `\getsheetdata` `\sheetdataempty` `\defproblemdata` `\setproblemdata` `\getproblemdata` `\problemdataempty`

Metadata for sheets can be used in the same way as the global metadata. The following directives are analogous to `\defexercisedata`, `\exercisedata`, `\getexercisedata` and `\exercisedataempty`:

```
\defsheetdata{key}
\setsheetdata{data}
\getsheetdata{key}
\sheetdataempty{key}{empty code}{filled code}
\defproblemdata{key}
\setproblemdata{data}
\getproblemdata{key}
\problemdataempty{key}{empty code}{filled code}
```

`pdfdata` `\writeexercisedata`

The most relevant metadata can be written to the metadata section of pdf files (using pdfLaTeX and the package hyperref whenever loaded). This feature is configured by:

```
\exercisesetup{pdfdata[=auto|manual|sheet|off]}
```

The option `auto` writes the global metadata `title`, `author`, `subject` and `keyword` to the corresponding fields in the pdf file. To make this work, these must be defined before the `\begin{document}` directive. The option `manual` allows to manually write these metadata by the command `\writeexercisedata`. It should be issued after the metadata have been set, but before any content is written to the pdf file. In other words, it can be anywhere in the document preamble directly after `\begin{document}`, or following a couple of content-free definitions at the beginning of the document body (in case the metadata should be set within the document body for some reason). The option `sheet` writes out the metadata at the beginning of the first `sheet` environment (which should follow `\begin{document}` without any content in between). This option is primarily for filling the `author` and `title`

fields with metadata of a sheet rather than a collection of exercises. Note that if no `author` is defined for the sheet, the global metadata `author` is used. The option `off` disables all writing of metadata.

`problem` `subproblem` `solution`

There is an additional mechanism to keep track of metadata for problems, subproblems and solutions which can be displayed in the opening line of these entities. Displayed metadata serve two purposes: they are used to describe the quality of a problem or they are intended for internal documentation purposes. Their output can be controlled individually, e.g. only in development versions of a document. Note that specifying a key more than once will display the content multiple times in the order in which they are encountered. Displayed metadata are specified at the top of the corresponding environment:

```
\begin{problem}[opts]
\begin{subproblem}[opts]
\begin{solution}[opts]
```

The standard set of displayed metadata keys consists of:

- `author`: author(s) of the problem (or subproblem, solution).
- `editor`: editor(s) of the problem; this field refers to a person who has made adjustments to the problem whereas `author` refers to the creator of the problem.
- `source`: source of the problem; in case the problem has been taken from elsewhere (conceptually or literally).
- `difficulty`: indication of the level of difficulty of the problem.
- `keyword`: keyword(s) for the problem;
- `comment`: some comment on the problem.
- `optional` (display enabled by default): whether addressing the problem is mandatory or optional; by default the text will be displayed after the title in italic shape.

By default, only the `optional` items are displayed, all other types of items are hidden; controlling the display for each type of item is described below.

`extdata`

Further displayed metadata keys are defined by the package option `extdata`, see section 2.8:

- `review`: field to review the aspects of the problem (quality, length, appropriateness, difficulty, ...).
- `recycle`: indication of previous instances where this problem was used.
- `timesolve`: indication of the time needed to solve this problem (or subproblem).
- `timepresent`: indication of the time needed to present this problem (or subproblem, solution).

`\showprobleminfo`

The display of the above metadata fields for a problem (or subproblem, solution) is controlled by:

```
\showprobleminfo{keys}
```

Here *keys* is a comma-separated list of keys to be activated (*key* or *key*`=true`) or deactivated (*key*`=false`).

`\defprobleminfo`

Displayable metadata can be defined or adjusted by:

```
\defprobleminfo{key}{code}
```

Here *key* specifies the metadata field and *code* the code to display this type of metadata where the argument `#1` represents the data to be displayed.

`insertprobleminfo` `insertsubprobleminfo` `insertsolutioninfo` Additional information can be injected into the opening line of problems and solutions by the definitions:

```
\exerciseconfig{insertprobleminfo}{code}
\exerciseconfig{insertsubprobleminfo}{code}
\exerciseconfig{insertsolutioninfo}{code}
```

`\addprobleminfo` `\addprobleminfo*` The hook code *code* will be called after processing the environment arguments. Information can be added to the opening line by:

```
\addprobleminfo{info}
\addprobleminfo*{info}
```

The unstarred command adds information at the end of the opening line, the starred version at the beginning (but after the title or identifier).

2.4 Points

`points` Exercise problems or certain parts of them can be credited with points (credits, awards, ...). The package provides an interface to specify and manage such points. Points are declared by the option `points=`*points* for the environments `sheet`, `problem` and `subproblem`. These numbers will be printed to the opening line of problems and subproblems.

Note that the points should normally be integer numbers. Fractional points are permissible as well, but the internal storage by the TEX engine is somewhat limited, so that only fractions with powers of two as denominators (.5, multiples of .25, .125, .0625, ...) are reliable. More general fractional decimal numbers such as multiples of 0.2 will be subject to rounding errors and will not display nicely.

Bonus points can be specified in the format `points=`[*regular*][`+`*bonus*]. By default, such points will be printed as [*regular*][`+`*bonus*] where 0 components are omitted.

`problempointsat` `subproblempointsat` `solutionpointsat` The location where points of problems and subproblems shall be displayed can be adjusted individually by:

```
\exercisestyle{problempointsat=start|start*|margin|end|manual|off}
\exercisestyle{subproblempointsat=start|start*|margin|end|manual|off}
\exercisestyle{solutionpointsat=start|start*|margin|end|manual|off}
```

The default values are `start` and `end` for problems and subproblems, respectively. The option `start` displays points at the very end of the opening line; the option `start*` displays them at the start of it. The option `end` displays points at the end of the problem or subproblem text. The option `margin` displays points in the margin. The option `manual` displays points at a manually chosen location specified by the directive `\showpoints`. Note that `\showpoints` can also be used for the option `end` to display the points prematurely (e.g. if the text ends with a displayed equation, it may make sense to display the points just before the equation). The option `off` disables the display of points.

`\getsheetdata` Points for sheets are only stored by the package; they must displayed manually. Within the corresponding `sheet` environment the points can be accessed by:

```
\getsheetdata{points}
```

`\getsheetpoints` `\getproblempoints` `\getsubproblempoints` `\getsolutionpoints` `\extractpoints` `\switchpoints` The package allows to read the point totals for other sheets and problems:

```
\getsheetpoints{[tag]}
\getproblempoints{[tag]}
\getsubproblempoints{[tag]}
\getsolutionpoints{}
```

Here *tag* is the tag assigned to the corresponding sheet or problem, see section 2.5. An empty argument *tag* refers to the current sheet, (sub)problem or solution. If bonus points are used, the points will be returned in the format [*regular*][+*bonus*]; the components *regular* and *bonus* can be extracted from the returned expression by `\extractpoints` and `\extractpoints*`, respectively. A convenient case switch of the returned value can be performed by:

```
\switchpoints{reg}{bonus}{both}{none}{val}
```

Here *val* is the value returned from the points register, *reg* is displayed for purely regular points, *bonus* is displayed for purely bonus points, *both* is displayed for mixed points, *none* is displayed for no points. In each of the four expressions, `#1` will be replaced by the regular points and `#2` by the bonus points.

`\awardpoints` Grading instructions with points to be awarded can be specified in the solution text by:

```
\awardpoints[details]{points}
\awardpoints*[details]{points}
```

Here *details* is an optional text with further details, e.g. to explain under which conditions these points are to be awarded. The starred form is used to specify optional points or alternative paths with alternative grading instructions. These points will be marked and not be used for the computation of a total.

`warntext` The package attempts to add up the points of subproblems to the problem total and likewise the points of problems to the sheet total. The package also performs some sanity checks on the provided numbers: If points are specified for both subproblems and problems or for both problems and sheets, they will be compared. Also the points within solutions (excluding optional or alternative points) are added up and compared to the corresponding problem or subproblem. Furthermore the package checks whether points are defined for all subproblems within a problem or all problems within a sheet. Mismatches are reported as package warnings. As point mismatches can be rather severe, there is an option to write such warnings directly into the output document (to be removed before distribution):

```
\exercisesetup{warntext[=true|false]}
```

`fracpoints` The package offers pretty display of fractional points with denominators 2, 4 and 8 by writing the decimal part as a fraction, e.g. $1.75 \rightarrow 1\frac{3}{4}$. This feature is enabled by:

```
\exercisestyle{fracpoints}
```

2.5 Labels and Tags

`label` LaTeX provides labels to make references to remote parts of the text. Labels can be set as usual by `\label{`*label*`}` within the `problem`, `subproblems` and `sheet` environments. Alternatively, they can be specified as the environment option:

```
label={label}
```

`tag` `\sheettag` `\problemtag` `\subproblemtag` The package provides an additional mechanism to tag sheets and problems. Each `sheet`, `problem` and `subproblem` can be assigned a unique tag *tag* by the environment option:

`tag={`*tag*`}`

This tag is used for reading point totals as described in section 2.4. Furthermore, the macro `\sheettag`, `\problemtag` or `\subproblemtag` is set to the tag *tag* within the current environment. If no tag is specified it defaults to the number of the current sheet or (sub)problem; note that this number can change by reordering sheets and problems and therefore it should not be used to identify the entity from other parts of the document.

A useful application for tags is to encapsulate labels within individual sheets and problems which are part of a collection of exercises. Labels which are composed as `\sheettag-`*label* or `\problemtag-`*label* can be considered local and will not clash with labels defined within a different environment. Within the same sheet or problem, local labels can be accessed by the same construction. They can also be accessed from remote parts of the document by fully expanding `\sheettag` or `\problemtag` for the desired target environment.

`autolabelsheet` `autolabelproblem` If unique tags are specified, the package can automatically create labels for sheets (`sheet:`*tag*) and problems (`prob:`*tag*) by:

```
\exercisesetup{autolabelsheet[=true|false]}
\exercisesetup{autolabelproblem[=true|false]}
```

`\getsheetlist` `\getproblemlist` `\getsubproblemlist` Tags can also be used to process a list of sheets, problems and subproblems:

```
\getsheetlist{}
\getproblemlist{[sheet-tag]|*}
\getsubproblemlist{[problem-tag]}
```

These commands return a list of sheets, problems and subproblems tags, where each item is encapsulated in braces. The commands `\get[sub]problemlist` return the (sub)problems within the specified sheet or problem. If no argument is given, the current sheet or problem is assumed. `\getproblemlist*` returns the list of all problems.

`\exerciseloop` `\exerciseloopstr` `\exerciseloopret` `exerciseloop` The package defines two commands to walk through such a list:

```
\exerciseloop{items}{cmd}
\exerciseloopstr[ret]{items}{str}
```

Here, *items* is a list of items in braces, and `\exerciseloop` evaluates *cmd* for each item of the list where '`#1`' is replaced by the item. The command `\exerciseloopstr` concatenates the evaluations of *str* and returns the resulting string in the macro *ret* (which defaults to `\exerciseloopret`). Both commands maintain a counter of items `exerciseloop` which can be accessed within *cmd*/*str* or after `\exerciseloop[str]` to obtain the total number of items in the list.

2.6 Layout

The package provides a large number of parameters to adjust the display of exercises to a desired layout.

`\exerciseconfig` Configuration settings are declared and modified by the command:

```
\exerciseconfig{key}[narg]{value}
```

Here *key* is a key and *value* is its assigned value. Configuration options can also be macros with arguments in which case *narg* is the number of arguments and *value* is the macro definition using arguments `#`*n*. The command `\exerciseconfig` therefore is analogous

to \(re)newcommand except that the definitions are encapsulated by the package and any previous definition is overwritten without checking.

exerciseconfigappend
xerciseconfigprepend

In some cases it may be useful to be able to append or prepend to a (parameterless) definition by:

\exerciseconfigappend{*key*}{*value*}
\exerciseconfigprepend{*key*}{*value*}

\getexerciseconfig
\exerciseconfigempty

Configuration definitions can be read by:

\getexerciseconfig{*key*}[*arguments*]

The number of arguments after {*key*} must match the optional argument *nargs* of the definition. Furthermore, it can be checked whether a configuration definition is empty:

\exerciseconfigempty{*key*}{*empty code*}{*filled code*}

The *empty code* is executed if no value or an empty value has been specified. Otherwise the *filled code* is executed.

The package defines numerous layout configuration options. They are listed along with their original definition and a brief description in the configuration section of the source code documentation. Some particular aspects are described in more detail in section 3.1. The configuration includes options to:

- adjust the language for the principal entities of this package like "sheet(s)", "problem(s)", "solution(s)", "points(s)";
- adjust the fonts styles of various parts of the text;
- adjust the spacing above, below, between various elements;
- define code to process data and insert text at various locations;
- compose text to be used in various situations;
- adjust the appearance of counters;
- adjust some other behaviour of the package.

We will highlight a few examples in section 3.2.

2.7 Exercise Styles

The package provides a mechanism to define exercise styles which customise the display of exercises in some coordinated fashion.

\exercisestyle

Style(s) are activated by the command:

\exercisestyle{*styles*}

Here *styles* is a comma-separated list of styles, where each style is given by a pair *style*[={*argument*}]. The package defines a couple of standard styles:

- solutionbelow=*pos* (can take values here, subproblem, subproblem*, problem, problem*, sheet and manual; initially set to subproblem) – positions the solutions below the indicated environments; see section 2.2 for details.

- `problempointsat=`*pos* (can take values `start`, `start*`, `margin`, `end` and `manual`; initially set to `start`) – displays points in problems at the indicated location; see section 2.4 for details.
- `subproblempointsat=`*pos* (can take values `start`, `start*`, `margin`, `end` and `manual`; initially set to `end`) – displays points in subproblems at the indicated location; see section 2.4 for details.
- `solutionpointsat=`*pos* (can take values `start`, `start*`, `margin`, `end` and `manual`; initially set to `end`) – displays points in solutions at the indicated location; see section 2.4 for details.
- `problemby={`*counter*`}` – number problems with the prefix *counter*, i.e. reset the problem counter whenever *counter* increases and use a composite label *counter*`.`*problem* to identify problems.
- `equationby={`*counter*`}` – number the dedicated equation counters for sheets, problems and solutions with the prefix *counter*.
- `problembysheet` – number problems by sheet.
- `equationbysheet` – number dedicated equations for sheets, problems and solutions by sheet; note that the main equation counter is unaffected by this setting, it therefore makes sense to also activate the style `sheetequation` or use `\counterwithin{equation}{sheet}`.
- `pagebysheet` – number pages by sheet and denote pages by *sheet*`.`*page*; this style is useful to generate stable page numbers for a collection of sheets.
- `sheetequation`[`=true`|`false`] (no value implies `true`, initially set to `false`) – use a dedicated equation counter within sheets.
- `problemequation`[`=true`|`false`] (no value implies `true`, initially set to `false`) – use a dedicated equation counter within problems.
- `solutionequation`[`=true`|`false`] (no value implies `true`, initially set to `true`) – use a dedicated equation counter within solutions.
- `fracpoints`[`=true`|`false`] (no value implies `true`, initially set to `false`) – display fractional points for denominators 2, 4, 8; see section 2.4 for details.
- `twoside`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable two-sided layout; in two-sided layout, sheets will start on odd pages and empty pages are added at the end of sheets to produce an even number of pages.
- `subproblemdelimitem` – shift subproblem number delimiter ')' from subproblem label to subproblem item composition; needed if `\ref` to a subproblem `\label` should produce a bare identifier without a trailing ')'; assumes (and restores) numbering `\alph` and delimiter ')'.

`extstyle` Further exercise styles are defined by the package option `extstyle`, see section 2.8:

- `plainheader` – define a plain sheet header to display some essential exercise and sheet data: `course`, `institution`, `instructor`, `period` (optional), sheet `title`, see section 2.3; the line below the header, font styles and spaces can be adjusted, see the definition in the styles section of the source code documentation.
- `contents` – display sheets and problems in the table of contents (as sections and subsections).
- `solutionsf` – display solutions in sans serif font family.
- `solutiondimproblem` – dim the problem text whenever solutions are displayed.
- `solutionsep` – separate the solutions from the remaining text by horizontal lines.

\defexercisestyle Custom styles can be defined by:

\defexercisestyle{*style*}{*init*}
\defexercisestylearg[*default*]{*style*}{*init*}

This feature can be used to predefine certain aspects of the exercises layout. For example, different default page layouts could be declared in this way. The first version declares a style which is initialised by the code *item* upon activation by \exercisestyle{*style*[=true]}. Note that \exercisestyle{*style*=false} does nothing. The second version declares a style which is activated by \exercisestyle{*style*[={*arg*}]} and which calls *item* with the argument #1 referring to *arg* (or *default* if no argument is given).

2.8 Package Options

\exercisesetup Features and options of general nature can be selected by the commands:

\usepackage[*opts*]{exframe}
or \PassOptionsToPackage{*opts*}{exframe}
or \exercisesetup{*opts*}

\PassOptionsToPackage must be used before \usepackage; \exercisesetup must be used afterwards. *opts* is a comma-separated list of options.

The following options are available only when loading the package, i.e. they will not work within \exercisesetup:

- extdata[=true|false] (no value implies true, initially set to false) – define some more advanced metadata entries.
- extstyle[=true|false] (no value implies true, initially set to false) – define some more advanced styles.
- metastr[=true|false] (no value implies true, initially set to false) – load metastr package and use to handle pdf metadata and basic internationalisation, see the metastr interaction section of the source code documentation for details.
- problemenv=*name* – redefine environment name problem. This and the following alike options may be useful in quickly adjusting existing sources to the exframe framework if the original framework works similarly and no special features are used. Otherwise, it is highly advisable to leave the names of environments and counters defined by the package untouched.
- subproblemenv=*name* – redefine environment name subproblem.
- solutionenv=*name* – redefine environment name solution.
- sheetenv=*name* – redefine environment name sheet.
- problemcounter=*name* – redefine counter name problem.
- subproblemcounter=*name* – redefine counter name subproblem.
- solutioncounter=*name* – redefine counter name solution.
- sheetcounter=*name* – redefine counter name sheet.

The following options can be specified by all three methods described above:

- solutions[=true|false] (no value implies true, initially set to false) – Enable/disable display of solutions. Sets the conditional \ifsolutions accordingly.

- `pdfdata`[`=auto`|`manual`|`sheet`|`off`] (no value implies `auto`, initially set to `auto`) – control writing most relevant metadata to pdf files; has no effect without package hyperref.
- `lineno`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable writing of line numbers as comments into solution files.
- `twoside`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable two-sided layout; see section 2.7 for details.
- `solutionhref`[`=true`|`true`] (no value implies `true`, initially set to `false`) – enable/disable use of hyper-references from solutions to the corresponding problems; has no effect without package hyperref.
- `warntext`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable writing of relevant warning messages (points mismatch, point sums require update) into the document output for easier detection.
- `autolabelsheet`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable automatically assigning labels (`sheet:\sheettag`; can be adjusted) to sheets according to their tag `\sheettag`.
- `autolabelproblem`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable automatically assigning labels (`prob:\problemtag`; can be adjusted) to problems according to their tag `\problemtag`.
- `solutionbuf`[`=true`|`false`] (no value implies `true`, initially set to `true`) – enable/disable buffering for `solution` environments in order to control their display; disabling buffering can be helpful in debugging faulty `solution` environments; it might also resolve some tokenisation issues in special circumstances; note that the display of solutions cannot be suppressed with `\exercisesetup{solutions=false}` when buffering if disabled.
- `problembuf`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable buffering for `problem` environments in order to control their display.
- `subproblembuf`[`=true`|`false`] (no value implies `true`, initially set to `false`) – enable/disable buffering for `subproblem` environments in order to control their display.
- `emptytestchar=`*char* (initially set to '`&`') – character to use for testing whether an argument `#1` is empty via `\if&#1&...\fi`; if '`&`' causes trouble within tables, could try '`@`' instead.

3 Customisations

This section attempts to shed some light on the rich array of options for layout customisations. It is not at all complete, but it provides an overview and illustrates selected aspects by means of guidance and examples.

3.1 Guidance

The following provides some guidance on customisation options which might otherwise be hard to follow. Suggestions for topics to be expanded are welcome.

Package Philosophy. The philosophy of the present package is to define a low-level framework to describe exercises with solutions to be used in various situations. The aim is to provide the means to describe the content (problems, solutions, sheets) in a simple fashion and separate it from the various layout definitions and choices which will define the appearance of the content. The package itself provides a functional basic layout rather than

an elaborate or particularly appealing one. However, it provides means to adjust the basic layout in many ways and to predefine custom layout schemes. The package is meant to collaborate with other packages which provide solutions for particular applications in order to produce an appealing layout for the given content.

Framework for Layout Customisation. The framework for defining the layout consists of an internal and a public component which allow for minor adjustments and major changes both with reasonable efforts.

The internal component implements some very coarse choices for the presentation of the content. It is hard-coded in the package (using internal macros starting with `exf@`) and its operation can be adjusted mainly through the available package options, see section 2.8, and through configuration switches for individual blocks.

The public framework uses a hierarchy of configuration macros defined by `\exerciseconfig`, see section 2.6. The internal component interfaces with the public framework through a set of higher-level interface macros. These include macros which produce the desired text in a well-prescribed situation as well as macros which perform some LaTeX operations such as adjusting presentation styles or modifying some variables. In many cases the structural higher-level macros have been set up to branch out to further lower-level macros prescribing presentation details.

The hierarchy has the following structure: High-level macros compose certain abstract elements into readable text, e.g. an intro/outro for some block of text made from space, decorations and abstract text elements. Medium-level macros compose elementary objects into abstract text elements, e.g. a header for some block of text composed from a label and a title with some space and delimiters in between. Low-level macros define elementary objects, such as delimiter characters, spaces, widths and words (translations).

The public framework has been set up such that elements at high, medium and low levels can be adjusted independently to arrive at a specific effect. For example, the composition of problem titles can be adjusted abstractly, but also the representation of a specific delimiter can be tweaked from '.' to ':'; either change can be obtained by adjusting a single definition. This flexibility comes along with an unfortunate complexity to allow for adjustments to be implemented individually but also universally. Nevertheless, the implementation of the public configuration macros for a particular setup can be rather simple because the particular setup will neither use all of the available coarse layout choices nor all of the customisation options. For example, a higher-level macro for a problem title may directly produce the desired text without branching into further lower-level definitions. Also, only those higher-level macros which actually come to use in a particular setup will need to be implemented.

The following paragraphs provide guidance on aspects of the public framework. There is no complete manual for the framework, but a skilled LaTeX tinkerer can find further information and pieces of code ready for tweaking in the implementation notes the configuration and styles sections of the source code documentation.

Labels for Problems and Sheets. The package provides many methods with various names to customise labels for problem and solution blocks. These names follow certain patterns (but are sometimes stuck with odd assignments due to backward compatibility). The following guide describes which methods come to use in which situations.

The labels can be grouped into two categories, items and titles, and the corresponding methods are typically called `...item...` and `...title...`. Items are enumerative identifiers for the [sub]problems such as '1.', 'b)' or '3iv)'. Titles serve as headers for [sub]problem or solution blocks which usually take the form "Problem 1", "2. Fermat's Last" or simply "c)" for a subproblem. Titles may use the corresponding item, and display it together with the

type, name or using a different punctuation depending on the situation.

composetitleproblem composenamedproblem composebareproblem

The composition of problem titles through `composetitleproblem` depends mainly on whether an individual name is provided (`title` option to `problem` environment). The method passes on to `composenamedproblem` if a name is provided (second argument non-empty), otherwise to `composebareproblem`. It might also produce a plain name if no item is provided (first argument empty).

osenamedproblemdelim mposenamedproblemsep posebareproblemdelim omposebareproblemsep poseitemproblemdelim omposeitemproblemsep

Titles and items are separated by some space (`...sep`) and/or delimiting characters (`...delim`). The item within a named title is separated from the following name by `composenamedproblemdelim` and `composenamedproblemsep`. In an unnamed title, the type is separated from the following item by `composebareproblemsep`, and the item is followed by `composebareproblemdelim`. A plain item not used within a title is delimited by `composeitemproblemdelim`. The presentation of all titles and items is followed by `composeitemproblemsep`.

composetocproblem

Problem titles can also be made to appear in the table of contents, their display is governed by `composetocproblem` which distinguishes between named and unnamed problems. The method uses the above delimiter `composenamedproblemdelim` but no other customisations because detailed layout will be less welcome in the table of contents.

composetitlesheet composetocsheet composemetasheet

Sheet titles generated by `composetitlesheet` are similar to problem titles but simpler. They may or may not have a name, but there are no pre-defined delimiters. The two similar methods `composetocsheet` and `composemetasheet` govern the display of sheet titles in the table of contents and in pdf metadata, respectively.

omposeitemsubproblem mposetitlesubproblem eitemsubproblemdelim oseitemsubproblemsep countersubproblem subproblemdelimitem

Subproblem items and titles are simple as well because subproblems have no names. They are generated by `composeitemsubproblem` and `composetitlesubproblem`, respectively. The subproblem delimiter and separator are determined by `composeitemsubproblemdelim` and `composeitemsubproblemsep`, respectively. Note that the delimiter ')' within 'b)' may be declared as either `composeitemsubproblemdelim` or as part of the subproblem counter defined in `countersubproblem`. This behaviour is governed by the style `subproblemdelimitem`.

Labels for Solutions. The display of appropriate solution items and titles is very complex as it depends on four factors: (i) The first factor is whether solutions are displayed as an itemised list or as a plain list. (ii) The second factor is whether solutions are displayed individually, grouped by problem or assembled as a list for multiple problems. (iii) The third factor is whether the solution corresponds to a problem or subproblem. (iv) A fourth factor is whether a solution is the first one within a problem. In addition, the list of solutions may have a title (v) and sections (vi). Let us discuss the relevant combinations.

omposetitlesolutions lutionsproblemsingle olutionsproblemmulti

The solution list title ("Solutions") (v) generated by `composetitlesolutions` only applies to lists of (potentially) multiple problems (ii). The solution section title (vi) generated by `composetitlesolutionsproblem...` indicates the corresponding problem. It comes in two variants `...single` and `...multi` which distinguish a single problem ("Solution") from a list of (potentially) multiple problems ("Problem 2") (ii).

skipsolutionitem skipsolutionitemsub composeitemsolution mposeitemsolutionsub oseitemsolutionfirst itemsolutionfirstsub

An itemised list of solutions (i) is invoked by a non-zero width `skipsolutionitem`[`sub`] (iii) for horizontal alignment of the solution text. In this case, solution items are displayed to the left of the line of horizontal alignment. These are generated by the configuration `composeitemsolution`[`sub`] (iii) irrespectively of the display mode (ii). By default, the current problem item is displayed as a section title (vi), therefore the solution items will only display the corresponding subproblem item leaving blank the solution items for problems. However, in an adjusted scheme without section titles (vi), the very first solution within a problem (iv) might additionally indicate the corresponding problem item `composeitemsolutionfirst`[`sub`] (iii).

etitlesolutionsingle oseitemsolutiondelim

An individual solution (ii) formally part of a plain list (i) starts with the title "Solution:"

which is generated by the method `composetitlesolutionsingle`. The delimiter ':' is defined by `composeitemsolutiondelim`.

`setitlesolutionmulti` `titlesolutionproblem` `lesolutionsubproblem` The title for a grouped solution (ii) within a plain list (i) is generated by the method `composetitlesolutionmulti` which forwards to `composetitlesolution`[`sub`]`problem` (iii). Again, by default, these methods only display the subproblem item because the problem item is provided as a section title (vi) instead.

`composetocsolution` `composetocsolutions` Items for the table of contents corresponding to solution lists and problem solutions are generated by `composetocsolution`[`s`].

3.2 Examples

This section contains some sample customisations of the default (plain) layout which can be realised with moderate effort using the various tools provided by the package. Additional applications are shown in the sample files `exframe-samp.tex` and `exframe-ser.tex`. Suggestions for further applications are welcome.

`insertsheettitle` **Sheet Title.** An important setting is:

`\exerciseconfig{insertsheettitle}{`*code*`}`

The code *code* is meant to print the title or header of an exercise sheet. The minimalistic default code `\centerline{\getsheetdata{title}}` merely prints the sheet title "Sheet #" at the centre of a line. Commonly, one would replace this by a more elaborate header (potentially with some more information, appealing layout, logos, ...). In order to design a header template, it makes sense to retrieve data via `\getexercisedata` and `\getsheetdata` described in section 2.3. Likewise `\exercisedataempty` and `\sheetdataempty` can be used to display default values or alternative data if some particular data is not provided. An example is given by the `plainheader` extended style option defined in the styles section of the source code documentation.

`composetitleproblem` `composenamedproblem` `composebareproblem` **Problem Title.** Another noteworthy example is `composetitleproblem` to compose the title for a problem. It takes two parameters, the number and the title. The (somewhat simplified) default declarations are:

```
\exerciseconfig{composetitleproblem}[2]{\exerciseifempty{#2}
  {\getexerciseconfig{composebareproblem}{#1}}
  {\getexerciseconfig{composenamedproblem}{#1}{#2}}}
\exerciseconfig{composebareproblem}[1]{\getexerciseconfig{termproblem}
  #1\getexerciseconfig{composebareproblemdelim}}
\exerciseconfig{composenamedproblem}[2]
  {#1\getexerciseconfig{composenamedproblemdelim} #2}
```

This checks whether the title is empty. If no title is given use the configuration `composebareproblem` to compose "Problem #", where the term "Problem" is made abstract by the configuration `termproblem` (e.g. to support internationalisation). If a title is given use the configuration `composenamedproblem` to compose "#. *title*".

`\exerciseifempty` `\exerciseifnotempty` **Conditionals.** Handy conditional commands to check whether an expression *expr* is empty are:

`\exerciseifempty{`*expr*`}{`*empty code*`}{`*filled code*`}`
`\exerciseifnotempty{`*expr*`}{`*filled code*`}`

Their main purpose is to test whether some provided expression *expt* is empty. They expand to the common TEX constructs `\if&#1&#2\else#3\fi` and `\if&#1&\else#2\fi` which work assuming that *expr* is not too exotic (e.g. it should not start with the character '`&`' and other special TEX characters or macros are potentially dangerous; also using this within tables can be troublesome; the character '`&`' can be reconfigured by the package option `emptytestchar`).

nsertsolutionsbefore
insertsolutionsafter
posetitlesolution...

Decorate Solutions with Boxes. The following adjustment changes the presentation of solutions to be surrounded by boxes with background colour (thanks to Till Bargheer for the example). This yields an alternative presentation that visually distinguishes problem from solution content.

The adjustment uses the package tcolorbox to supply such boxes. The coloured box environment `tcolorbox` is invoked by the customisations `insertsolutionsbefore` and `...after` which are called just before and after a block of solutions is:

```
\RequirePackage{tcolorbox}
\tcbuselibrary{breakable}
\exerciseconfig{insertsolutionsbefore}{%
  \begin{tcolorbox}[breakable,boxrule=0.25pt]}
\exerciseconfig{insertsolutionsafter}{\end{tcolorbox}}
\exerciseconfig{composetitlesolutionsingle}[2]{}
```

With the highlighting of solutions by boxes, the default introductory statement "Solution:" (when `solutionbelow` is `here` or `subproblem`) of a solution block becomes obsolete. This is achieved by removing the content of `composetitlesolutionsingle` (or similar composition definitions if `solutionbelow` is set otherwise).

solutionpointsat
insertpointsmargin

Note that displaying points in the margin would have to be adjusted because the plain LATEX command `\marginpar` does not work inside a `tcolorbox` (due to nested floats). If this combination is desired, one should adjust the placement of margin paragraphs to the package marginnote:

```
\exercisestyle{solutionpointsat=margin}
\RequirePackage{marginnote}
\exerciseconfig{insertpointsmargin}[1]{\marginnote{\footnotesize#1}}
```

4 Information

4.1 Copyright

4.2 Files and Installation

The package consists of the files:

`README.txt`	readme file
`exframe.ins`	installation file
`exframe.dtx`	source file
`exframe.sty`	package file
`exframe.tex`	reference manual file
`exframe.pdf`	reference manual
`exframe-src.tex`	source code documentation file
`exframe-src.pdf`	source code documentation
`exframe-samp.tex`	sample file
`exframe-ser.tex`	multipart sample main file
`exframe-ser-0`n`.tex`	multipart sample sheet n=`1`, `2`, `3`
`exframe-ser-aa.tex`	multipart sample unused problems
`exframe-ser-p`x`.tex`	multipart sample problem x=`e`, `f`
`exframe-ser.sh`	multipart sample compile script
`exframe-ser.mak`	multipart sample makefile

The distribution consists of the files `README.txt`, `exframe.ins` and `exframe.dtx`.

- Run LaTeX on `exframe.ins` to create the package `exframe.sty` and further documentation files.
- Copy the file `exframe.sty` to an appropriate directory of your LaTeX distribution, e.g. *texmf-root*`/tex/latex/exframe`.
- Alternatively, you may copy `exframe.sty` to your project directory.
- Run (pdf)LaTeX on `exframe.dtx` to compile the documentation `exframe.pdf` (this file).

4.3 Related Packages

The package makes use of other packages available at CTAN:

- This package relies on some functionality of the package verbatim to read verbatim code from the LaTeX source without expansion of macros. Compatibility with the verbatim package has been tested with v1.5q (2014-10-28).
- This package uses the package xkeyval to process the options for the package, environments and macros. Compatibility with the xkeyval package has been tested with v2.7a (2014-12-03).
- This package can use the package hyperref to include hyperlinks between problems and solutions. Compatibility with the hyperref package has been tested with v6.88e (2018-11-30).
- This package can use the package amstext (which is automatically loaded by amsmath) to display text within equations. Compatibility with the amstext package has been tested with v2.01 (2000-06-29).
- This package uses the command `\currfilename` provided by the package currfile (if available and loaded) to indicate the LaTeX source file in the generated metapost file. Compatibility with the currfile package has been tested with v0.7c (2015-04-23).

- This package can use the package metastr to write pdf metadata and to handle basic translations. Compatibility with the metastr package has been tested with v1.0 (2020-02-06).

There are several other LaTeX packages which offer a similar functionality varying largely in scope and sophistication:

- The package exsheets and its successor xsim provide a LaTeX 3 style for typesetting exercises with solutions. They offer options to hide or delay solutions, print only specific problems, deal with points, specify metadata, handle exercise collections, as well as some more specific options. They allow to adjust the layout and choose among predefined ones.
- The package exercise provides a style for typesetting exercises with solutions. It offers many options to hide or delay solutions, print only specific problems, specify some metadata as well as some more specific options. It allows to customise the layout.
- The package exercises provides a style for typesetting exercises with solutions. It offers options to hide solutions and deal with points. It allows basic customisation of the layout.
- The package exam provides a document class for typesetting exams conveniently. It offers many options to hide solutions, deal with points and deal with other exam-specific tasks. It allows to adjust the layout and choose among predefined ones.
- The package probsoln provides a style for typesetting exercises with solutions which are stored in a collection. It offers options to hide solutions and to assemble problems from an external collection.
- The packages uebungsblatt, uassign, mathexam, exsol, homework, jhwhw provide basic functionality for somewhat more particular situations.

See CTAN categories exercise and exam for further up-to-date packages.

The philosophy of the present package is to define a low-level framework to describe exercises with solutions to be used in various situations. The aim is to provide the means to describe the content (problems, solutions, sheets) in a simple fashion and separate it from the various layout definitions and choices which will define the appearance of the content. The interface was designed to reduce potential conflict with other packages and definitions. The package itself provides a functional basic layout rather than an elaborate or particularly appealing one. However, it provides means to adjust the basic layout in many ways and to predefine custom layout schemes. The package offers most of the functionality of the above packages, but (presently) misses out on some more advanced features. See section 3 on customisation assistance and examples and section 4.4 for feature suggestions.

4.4 Feature Suggestions

The following is a list of features which may be useful for future versions of this package:

- Define structures for multiple-choice questions.
- Add more guidance to section 3.1 on customisation settings.
- Add more examples of customisation settings to achieve specific goals to section 3.2. Please send suggestions.
- Option to hide problem text while maintaining access to embedded solutions (for a version containing only solutions): this is difficult to implement because the problem environment cannot simply be discarded, but would have to be scanned very carefully for the embedded solution; instead process problems to some document and save solutions to file, then read solutions from different document.

4.5 Revision History

v3.5.3: 2026-03-23

- split off source code documentation into separate document `exframe-src`
- package maintenance

v3.5.2: 2026-01-22

- distinguish between problem, subproblem and solutions for display of points in margin

v3.5.1: 2025-03-25

- maintenance and manual update

v3.5: 2024-10-18

- more convenient delimiter configuration `compose...delim` for various items
- composition of problem titles in `composetitleproblem` split up further into the configurations `composebareproblem` and `composenamedproblem`; to enable delimiter '.' for problems without a name, use

  ```
  \exerciseconfig{composebareproblemdelim}{.}
  ```

- style `subproblemdelimitem` added to shift delimiter ')' from subproblem reference labels to subproblem item composition.
- `composetitlesolutionsproblemmulti` improved (thanks to Daniel Wegmann for pointing out insufficient behaviour)
- option `forproblem` for `solution` to release association to previous subproblem
- option to `disable` particular subproblems and solutions including following `solution` block
- handling of first solutions within a problem block

v3.4: 2020-02-24

- lists of sheets, problems and subproblems; list iterators
- tags for subproblems, tag customisation
- interaction with package metastr
- minor fixes

v3.31: 2020-01-11

- `onlysolutions` environment for solution mode content
- sample multipart setup streamlined

v3.3: 2019-06-15

- control display of `problem` environments via package option `problembuf`: manual display, write to file, disable individual problems
- `solutionbelow` mode `here*` superseded by package option `solutionbuf`
- display total points within solution: `solutionpointsat` (thanks to Till Bargheer for suggestion)
- read points for current sheet, (sub)problem and solution (thanks to Johannes Hahn for suggestion)
- case switch for bonus points (thanks to Johannes Hahn for suggestion)
- option to `disable` particular problems, control by hook function (thanks to Manuel Benz for suggestion)
- provided interface `\showfracpoints` and `\exerciseconfig{frac}` for fractional points display
- filename extensions configurable

v3.2: 2019-05-01

- bonus points can be specified as `points=`[*regular*][`+`*bonus*]
- `solutionbelow` mode `here*` added for direct processing of the solution environment
- multipart sample added

v3.11: 2019-04-15

- fix interaction with package calc (thanks to Johannes Hahn for bug report)
- fix style `fracpoints` in combination with some [`sub`]`problempointsat` choices (thanks to Johannes Hahn for bug report)
- fix spacing for [`sub`]`problempointsat=margin` (thanks to Johannes Hahn for bug report)

v3.1: 2019-01-21

- alternate placement modes for solutions
- fixed expansion of problem title
- reset font size for problem text

v3.0: 2019-01-16

- renamed to `exframe.sty`
- first version published on CTAN
- overhaul and streamline interface
- solution processing remodelled
- changed metadata handling
- changed and generalised points handling
- generalised sectioning layout

- changed layout specification model
- insert hyperlinks using hyperref
- manual, example and installation package added

v2.0 – v2.6: 2014-10-03 – 2018-11-05

- changed metadata interface
- broadened scope
- added more layout options
- added more metadata
- added sheet and problem tags
- add and remember points

v1.1 – v1.6: 2014-08-07 – 2014-09-14

- renamed to `nbprob.sty`
- added metadata
- added points
- added layout configuration
- removed specific macros

v1.0 – v1.02: 2011-09-23 – 2013-03-17

- first version as `problems.cls`
- dedicated layout and macros for author's exercise sheets