

Package: exams (via r-universe)

August 13, 2024

Version 2.4-2

Date 2024-08-04

Title Automatic Generation of Exams in R

Description Automatic generation of exams based on exercises in Markdown or LaTeX format, possibly including R code for dynamic generation of exercise elements. Exercise types include single-choice and multiple-choice questions, arithmetic problems, string questions, and combinations thereof (cloze). Output formats include standalone files (PDF, HTML, Docx, ODT, ...), Moodle XML, QTI 1.2, QTI 2.1, Blackboard, Canvas, OpenOlat, ILIAS, TestVision, Particify, ARSnova, Kahoot!, Grasple, and TCEexam. In addition to fully customizable PDF exams, a standardized PDF format (NOPS) is provided that can be printed, scanned, and automatically evaluated.

Depends R (>= 3.4.0)

Imports graphics, grDevices, stats, tools, utils, base64enc, knitr, rmarkdown

Suggests magick, openxlsx, parallel, png, qpdf, RCurl, RJSONIO, tinytex, tth, xml2

SystemRequirements pandoc (>= 2.0)

License GPL-2 | GPL-3

URL <https://www.R-exams.org/>

BugReports <https://www.R-exams.org/contact/>

Repository <https://zeileis.r-universe.dev>

RemoteUrl <https://github.com/r-forge/exams>

RemoteRef HEAD

RemoteSha 184988d103e30da57ecbfefc840d107576c1cc99

Contents

exams	3
exams2arsnova	6
exams2blackboard	8
exams2canvas	12
exams2evaexam	14
exams2grasple	16
exams2html	18
exams2ilias	21
exams2inspera	24
exams2kahoot	26
exams2lops	27
exams2moodle	29
exams2nops	34
exams2openolat	38
exams2pandoc	41
exams2particify	43
exams2pdf	45
exams2qti12	48
exams2qti21	53
exams2tcexam	59
exams2testvision	60
exams_eval	64
exams_skeleton	68
expar	70
fmt	71
include_supplement	73
include_tikz	74
make_exercise_transform_pandoc	75
match_exams_call	77
matrix_to_schoice	78
mchoice2string	80
moodle2exams	81
nops_eval	83
nops_fix	87
nops_language	89
nops_scan	90
num_to_schoice	93
read_exercise	94
stresstest_exercise	97
testvision2exams	99
tex2image	101
xexams	103
xweave	106

Description

Old (version 1) interface for Sweave-based automatic generation of exams including multiple choice questions and arithmetic problems. Now it is recommended to use the (version 2) interface [exams2pdf](#).

Usage

```
exams(file, n = 1, nsamp = NULL, dir = NULL, template = "plain",
      inputs = NULL, header = list(Date = Sys.Date()), name = NULL,
      quiet = TRUE, edir = NULL, tdir = NULL, control = NULL)
```

Arguments

file	character. A specification of a (list of) exercise files, for details see below.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character. The output directory, this has to be set if n is greater than 1 (or template is longer than 1).
template	character. A specification of a LaTeX template. The package currently provides "exam", "solution", "plain". For details see below.
inputs	character. Names of files that are needed as inputs during LaTeX compilation (e.g., style files, headers). Either the full path must be given or the file needs to be in edir.
header	list. A list of further options to be passed to the LaTeX files.
name	character. A name prefix for resulting exercises, by default chosen based on template.
quiet	logical. Should output be suppressed when calling Sweave and texi2dvi .
edir	character specifying the path of the directory in which the files in file are stored (see also below).
tdir	character specifying a temporary directory, by default this is chosen via tempfile . Note that this is cleaned up and potentially temporary files are deleted.
control	A list of control arguments for the appearance of multiple choice results (see 'Details').

Details

exams is the old (version 1) interface for Sweave-based generation of PDF exams. It is only provided for backward compatibility and is superseded by the far more flexible function `exams2pdf`.

exams generates exams from lists of Sweave source files by: (1) running Sweave on each exercise, (2) including the resulting LaTeX files in a template, (3) running `texi2dvi` on the template, and (4) storing the resulting PDF file in an output dir (or displaying it interactively).

Each exercise in an exam is essentially a standalone Sweave source file that exams knows (almost) nothing about, it just calls Sweave (n times). The only exception is some meta-information which is passed by means of four commands back to exams. The commands are `\extype` (which may be `mchoice` or `num`), `\exsolution` (e.g., 3.124 for a numeric solution and 10010 for a multiple choice solution), `\exstring` (containing a human-readable string with the solution), and `\extol` (a tolerance for numeric solutions).

The specification in file should be either of form `"foo"` or equivalently `"foo.Rnw"`, where the file `"foo.Rnw"` should either be in the local directory, the `edir` directory or in the `exercises` directory of the package. file can either be a simple vector or a list of vectors. In the latter case, exercises are chosen randomly within each list element. For example, the specification `file = list(c("a", "b"), "xyz")` will result in an exam with two exercises: the first exercise is chosen randomly between `"a"` and `"b"` while `"xyz"` is always included as the second exercise.

The template is a (vector of) specification(s) of LaTeX templates. It can be `"foo"` or equivalently `"foo.tex"` where `"foo.tex"` should either be in the local directory (or provided with the full path) or in the `tex` directory of the package. It should specify where in the template the exercises are included, using the markup `\exinput{exercises}`. Additionally, it may contain `\exinput{questionnaire}` and `\exinput{header}`. template can also be a vector, then for each of the n runs several output files (one for each template) are created.

The name prefix for each file is by default the base name of the corresponding template but can also be changed via `name`.

exams creates the PDF files and stores them in an output directory together with the solution meta information as `'metainfo.rda'` (see also below). If only a single PDF is created (currently the default), `dir` may be `NULL` and it is only displayed on the screen.

The argument `control` is specified by a named list with elements `mchoice.print` and `mchoice.symbol`. The element `mchoice.print` is used for specifying the characters used for printing. It is again a named list where element `True` gives the (five) characters used for printing when the answer is correct and `False` if the answer is wrong. The symbol used for the questionnaire output in the final PDF file is defined by `mchoice.symbol` which is vector with elements `True` and `False`.

Value

An object of class `"exams_metainfo"` is returned invisibly. It is a list of length n, containing a list of meta informations for each exercise:

<code>mchoice</code>	logical. Is the exercise a multiple choice exercise?
<code>length</code>	integer. Length of solution.
<code>solution</code>	either a logical vector (for multiple choice) or numeric vector (for arithmetic problems).
<code>string</code>	character. A human-readable version of the solution.

References

Gruen B, Zeileis A (2009). Automatic Generation of Exams in R. *Journal of Statistical Software*, **29**(10), 1–14. doi:10.18637/jss.v029.i10.

See Also

[exams2pdf](#), [Sweave](#), [texi2dvi](#), [mchoice2string](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rnw",
  c("tstat.Rnw", "ttest.Rnw", "confint.Rnw"),
  c("regression.Rnw", "anova.Rnw"),
  "scatterplot.Rnw",
  "relfreq.Rnw"
)

if(interactive()) {
  ## compile a single random exam (displayed on screen)
  sol <- exams(myexam)
  sol
}

## generate multiple exams (stored in output directory)
odir <- tempfile()
sol <- exams(myexam, n = 2, dir = odir, template = c("exam", "solution"))
sol

## inspect solution for a particular exam
print(sol, 3)

if(interactive()) {
  ## modify control argument for printing
  mymchoice.control <- list(mchoice.print = list(True = LETTERS[1:5], False = "_"))
  sol <- exams("boxplots.Rnw", template = "solution",
    control = mymchoice.control)
  sol
}
```

Description

Interface for generating interactive sessions in the JSON format of the audience response system ARSnova. As ARSnova has been superseded by Particify, it is recommended to use [exams2particify](#) instead. exams2arsnova will be removed in future versions of the package.

Usage

```
exams2arsnova(file, n = 1L, dir = ".",
  name = "R/exams", sname = NULL, qname = NULL,
  quiet = TRUE, resolution = 100, width = 4, height = 4, svg = FALSE,
  encoding = "UTF-8", envir = NULL, engine = NULL,
  url = "https://arsnova.eu/api", sessionkey = NULL, jsessionid = NULL,
  active = TRUE, votingdisabled = FALSE, showstatistic = FALSE, showanswer = FALSE,
  abstention = TRUE, variant = "lecture", ssl.verifypeer = TRUE,
  fix_choice = TRUE, ...)
```

```
make_exams_write_arsnova(url = "https://arsnova.eu/api", sessionkey = NULL,
  jsessionid = NULL, name = "R/exams", sname = NULL, qname = NULL,
  active = TRUE, votingdisabled = FALSE, showstatistic = FALSE, showanswer = FALSE,
  abstention = TRUE, variant = "lecture", ssl.verifypeer = TRUE, fix_choice = TRUE)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is either display on the screen or the current working directory.
name	character. A name prefix for resulting exercises and RDS file.
sname	character. A vector of length 2 with the session name (maximum of 50 characters) and its abbreviation (maximum of 8 characters). Ignored if the sessionkey of an existing session is supplied and otherwise copied by default from name.
qname	character. A vector of names for each question/exercise in file. By default, the name is used.
quiet	logical. Should output be suppressed when calling xweave and texi2dvi .
resolution, width, height	numeric, passed to xweave .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to xweave (which passes it to knit).

engine	argument passed to xweave indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
url, sessionkey, jsessionid	character specifying (1) the base URL of the ARSnova API, (2) the 8-digit ARSnova session key, (3) the JSESSIONID cookie of an active ARSnova session. If all are provided all questions are imported directly into the existing ARSnova session. Otherwise, a JSON import file is generated.
active	logical. Should the question be active (i.e., released for students) or locked?
votingdisabled	logical. Should voting be disabled?
showstatistic	logical. Should statistics be shown?
showanswer	logical. Should answers be shown?
abstention	logical. Are abstentions allowed?
variant	character. Should the question be a "lecture" or a "preparation" questions?
ssl.verifypeer	logical. Should SSL certificates be validated when connecting via https?
fix_choice	logical. Should math markup be removed in single and multiple choice lists? (This may be needed for older ARSnova versions where math markup is rendered in the question itself but not the choice list.)
...	arguments passed on to xexams .

Details

exams2arsnova generates exams in the JSON format for ARSnova using [xexams](#). It proceeds by (1) calling [xweave](#) on each exercise, (2) reading the resulting Markdown or LaTeX text, (3) transforming the text to Markdown, and (4) embedding the Markdown text into the JSON format for ARSnova (and optionally imports it into a running ARSnova session).

Since 2020 the development of ARSnova has been shifted to a new tool called Particify and hence the arsnova.eu server is not hosted anymore. For an export function to Particify see [exams2particify](#).

For steps (1) and (2) the standard drivers in [xexams](#) are used. For step (3) a suitable transformation function is set up on the fly using [make_exercise_transform_pandoc](#). For step (4) a simple writer function is set up on the fly that embeds the transformed Markdown code into a hard-coded JSON template using [toJSON](#) and either writes a single JSON file for each exam or imports these directly into an ARSnova session.

When url, sessionkey, and jsessionid are all supplied, [curlPerform](#) is used to import tall questions directly into the existing ARSnova session. Otherwise, a file is written to the disk and then needs to be imported manually into an ARSnova server. This file is either a JSON file for a whole new session (if sessionkey is NULL, the default) or a CSV file with the questions only.

Value

A list of exams as generated by [xexams](#) is returned invisibly.

See Also

[exams2particify](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## Not run:
## exams2arsnova can either create text files with JSON data
exams2arsnova("tstat2")

## or directly post this to an active ARSnova session (for which the
## server URL, the 8-digit session key, and the JSESSIONID cookie are needed)
exams2arsnova("tstat2", url = "https://arsnova.eu/api",
  sessionkey = "49061284", jsessionid = "A5BEFDA4141816BB425F2204A602E4B3")

## End(Not run)
```

exams2blackboard

Generation of Exams in Blackboard Format

Description

Automatic generation of exams in Blackboard format (which is partially based on QTI 1.2).

Usage

```
exams2blackboard(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL,
  tdir = NULL, sdir = NULL, verbose = FALSE, rds = FALSE,
  resolution = 100, width = 4, height = 4, encoding = "UTF-8",
  envir = NULL, engine = NULL,
  num = NULL, mchoice = NULL,
  schoice = mchoice, string = NULL, cloze = NULL,
  template = "blackboard",
  pdescription = "This is an item from an item pool.",
  tdescription = "This is today's test.",
  pinstruction = "Please answer the following question.",
  tinstruction = "Give an answer to each question.",
  maxattempts = 1, zip = TRUE, points = NULL,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  base64 = FALSE, converter = NULL, seed = NULL, mathjax = NULL,
  fix_pre = TRUE, ...)

make_itembody_blackboard(rtiming = FALSE, shuffle = FALSE,
  rshuffle = shuffle, minnumber = NULL, maxnumber = NULL,
  defaultval = NULL, minvalue = NULL, maxvalue = NULL,
  cutvalue = NULL, enumerate = TRUE, digits = NULL,
  tolerance = is.null(digits), maxchars = 12,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  qti12 = FALSE, mathjax = FALSE)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character. The default is the current working directory.
name	character. A name prefix for resulting exercises and ZIP file.
quiet	logical. Should output be suppressed when calling <code>xweave</code> ?
edir	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
tdir	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
sdir	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
verbose	logical. Should information on progress of exam generation be reported?
rds	logical indicating whether the return list should also be saved as an RDS data file.
resolution, width, height	numeric. Options for rendering PNG graphics passed to <code>xweave</code> .
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
engine	argument passed to <code>xweave</code> indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
num	function or named list applied to numerical (i.e., type num) questions. If num is a function, num will be used for generating the item body of the question, see function <code>make_itembody_blackboard()</code> . If num is a named list, these arguments will be passed to function <code>make_itembody_blackboard()</code> .
mchoice, schoice, string, cloze	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type mchoice, schoice, string, and cloze), respectively. See argument num for more details.
template	character. The IMS QTI 1.2 or 2.1 template that should be used. Currently, the package provides "blackboard.xml".
pdescription	character. Description (of length 1) of the item pool (i.e., the set of copies).
tdescription	character. Description (of length 1) of the overall assessment (i.e., exam).
pinstruction	character. Instruction (of length 1) for the item pool (i.e., set of copies).
tinstruction	character. Instruction (of length 1) for the overall assessment (i.e., exam).
maxattempts	integer. The maximum attempts for one question, may also be set to Inf.

zip	logical. Should the resulting XML file (plus supplements) be zipped?
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an <code>expoints</code> tag in the Rnw/Rmd exercise file. The vector of points supplied is expanded to the number of exercises in the exam.
eval	named list, specifies the settings for the evaluation policy, see function <code>exams_eval</code> .
base64	logical. Should supplementary files be embedded using Base 64 coding? Argument <code>base64</code> may also be a character vector of file suffixes that should be encoded, e.g. <code>base64 = c("png", "rda")</code> will only encode PNG images and binary <code>.rda</code> files. If set to NULL only image files will be encoded.
converter, ...	arguments passed on to <code>make_exercise_transform_html</code> . The default for <code>converter</code> is set to <code>"ttm"</code> unless there are Rmd exercises in file where <code>"pandoc"</code> is used.
seed	integer matrix or logical. Either NULL (default), logical, or a matrix of random seeds for each possible exercise to be set prior to calling <code>driver@sweave</code> . If NULL no random seeds are set. If a matrix, the number of rows must be <code>n</code> and the number of columns must correspond to <code>unlist(file)</code> . If TRUE a suitable matrix of seeds is sampled.
mathjax	logical. Should the JavaScript from https://www.MathJax.org/ be included for rendering mathematical formulas? By default <code>mathjax = FALSE</code> unless <code>converter = "pandoc-mathjax"</code> is used. However, also for the default converters (producing MathML output) <code>mathjax = TRUE</code> can be used, enabling rendering of mathematical equations in browsers without native MathML support (like Chrome/Chromium). Note that this only works in Classic Blackboard but not in Blackboard Ultra.
fix_pre	logical. Should the HTML <code><pre></code> environment for rendering verbatim output be replaced by <code><code></code> tags? This is necessary in classical Blackboard which does not render <code><pre></code> correctly.
rtiming, shuffle, rshuffle, minnumber, maxnumber, defaultval, minvalue, maxvalue	arguments used for IMS QTI 1.2 item construction, for details see the XML specification (see IMS Global Learning Consortium, Inc. 2012), especially Section 4.
cutvalue	numeric. The cutvalue at which the exam is passed.
enumerate	logical. Insert potential solutions in enumerated list?
digits	integer. How many digits should be used for num exercises?
tolerance	logical. Should tolerance intervals be used for checking if the supplied num answer/number is correct? The default is to use tolerance intervals if <code>digits = NULL</code> .
maxchars	numeric. Lower bound for the number of characters in fill-in-blank fields. The actual number of characters is selected as the maximum number of characters of this value and the actual solution.
qti12	logical. For reverse compability to plain QTI 1.2 XML format.

Details

Blackboard employs an XML format that essentially uses the Question & Test Interoperability (QTI) standard, version 1.2, see IMS Global Learning Consortium, Inc. (2012). However, as this deviates from the plain QTI 1.2 standard in several places, the `exams2qti12` cannot be used directly. Instead, `exams2blackboard` is a new interface that is likely to be improved in future versions.

`exams2blackboard` produces a .zip file that may be uploaded into Blackboard. This includes the final XML file of the exam/assessment as well as possible supplement folders that include images, data sets etc. used for the exam. After uploading the test into Blackboard, the material will appear under ‘Course Tools’: the test will be available in ‘Tests’, and each pool within the test will also appear in ‘Pools’.

`exams2blackboard` proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting LaTeX code, (3) transforming the LaTeX code to HTML, and (4) embedding the HTML code in a XML file using Blackboard’s QTI standards for assessments and question items. For steps (1) and (2) the standard drivers in `xexams` are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in `exams2html`. For step (4), the function will cycle through all questions and exams to generate the final XML file in the Blackboard QTI standard. Therefore, each question will be included in the XML as one section. The replicates of each question will be written as question items of the section.

The function uses the XML template for Blackboard’s QTI standards for assessments and items to generate the exam (per default, this is the XML file `blackboard.xml` provided in the `xml` folder of this package). The assessment template must provide one section including one item. `exams2blackboard` will then use the single item template to generate all items, as well as the assessment and section specifications set within the template.

The default template will generate exams/assessments that sample one replicate of a question/item for each section. The usual procedure in exam/assessment generation would be to simply copy & paste the XML template of the package and adapt it to the needs of the user. Note that all specifiers that have a leading `##` in the XML template will be replaced by suitable code in `exams2blackboard` and should always be provided in the template. I.e., the user may add additional tags to the XML template or modify certain specifications, like the number of replicates/items that should be sampled for each section etc.

Per default, the individual question/item bodies are generated by function `make_itembody_blackboard`, i.e., `make_itembody_blackboard` checks the type of the question and will produce suitable XML code. Note that for each question type, either the arguments of `make_itembody_blackboard` may be set within `num`, `mchoice`, `schoice` and `string` in `exams2blackboard`, by providing a named list of specifications that should be used, or for each questiontype, a function that produces the item body XML code may be provided to `num`, `mchoice`, `schoice` and `string`. E.g., `mchoice = list(shuffle = TRUE)` will force only multiple choice questions to have a shuffled answerlist.

Note that in Blackboard `cloze` items are not officially supported, and hence this type of item is not supported in the current version of `exams2blackboard` either. It is currently investigated if a workaround may be implemented to allow for `cloze` items.

Value

`exams2blackboard` returns a list of exams as generated by `xexams`.

`make_itembody_blackboard` returns a function that generates the XML code for the `itembody` tag in Blackboard’s version of the IMS QTI 1.2 format.

References

- Blackboard, Inc. (2016). *Blackboard Help: Question types*. https://help.blackboard.com/Learn/Instructor/Ultra/Tests_Pools_Surveys/Question_Types
- IMS Global Learning Consortium, Inc. (2012). *IMS Question & Test Interoperability: ASI XML Binding Specification Final Specification Version 1.2*. https://www.imslobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html
- Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

[exams2qti12](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
  "ttest.Rmd",
  "anova.Rmd",
  "scatterplot.Rmd",
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with Blackboard exam in temporary directory
exams2blackboard(myexam, n = 3, dir = mydir)
dir(mydir)
```

Description

Automatic generation of exams in QTI 1.2 with some tweaks (still under development) for the learning management system Canvas.

Usage

```
exams2canvas(file, n = 1L, dir = ".", name = "canvasquiz",
             maxattempts = 1, duration = NULL, points = NULL,
             converter = "pandoc-mathjax", template = "canvas_qti12.xml",
             quiztype = "assignment", ...)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is the current working directory.
name	character. A name prefix for resulting exercises and ZIP file.
maxattempts	integer. The maximum attempts for one question (must be smaller than 100000).
duration	integer. Set the duration of the exam in minutes.
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an <code>"\expoints{}</code> " tag in the <code>.Rnw</code> file. The vector of points supplied is expanded to the number of exercises in the exam.
converter	character passed on to <code>make_exercise_transform_html</code> , indicating the type of converter from LaTeX/Markdown to HTML.
template	character. The IMS QTI 1.2 template that should be used.
quiztype	character. The type of quiz that should be created in Canvas. Defaults to "assignment" with alternative values being "practice_quiz", "graded_survey", or "survey".
...	arguments passed on to <code>exams2qti12</code> . The arguments <code>base64</code> and <code>eval</code> cannot be modified but are hard-coded internally (for details see below).

Details

`exams2canvas` is a convenience interface to `exams2qti12` for generating QTI 1.2 with some small tweaks for Canvas (<https://www.instructure.com/canvas>). The supported exercise types at the moment are num, schoice, mchoice, and string. There is limited support for cloze exercises with multiple dropdown menus, i.e., schoice elements placed in the text with ANSWERi tags.

The Canvas-specific tweaks include:

- Canvas needs a converter that produces HTML with mathematical notation that can be rendered by MathJax. Hence "pandoc-mathjax" is the default but "pandoc-mathml" (or "ttm" for exams with only R/LaTeX `.Rnw` exercises) works as well, albeit possibly with problems when imported from a quiz to the item bank.
- Supplementary files (images, data, ...) must be embedded without Base 64 encoding. Thus, `base64 = FALSE` is hard-coded internally in `exams2canvas`.
- Multiple-choice exercises (referred to as multiple answer questions in Canvas) are always evaluated (aka scored) with partial credits in Canvas. Therefore, `eval = list(partial = TRUE, negative = FALSE)` is hard-coded internally in `exams2canvas`. Negative points are not supported in Canvas.

- The QTI XML file requires a few special tags which are enforced through `flavor = "canvas"` in `exams2qti12`.

Technical note: For multiple-choice questions the QTI XML file produced by `exams2canvas` appears to encode an “all-or-nothing” scheme without partial credits (i.e., `partial = FALSE`). However, this is necessary for Canvas to recognize the answer alternatives correctly. Despite this, Canvas always applies partial-credit evaluation (as explained above).

Value

`exams2canvas` returns a list of exams as generated by `xexams`.

See Also

[exams2qti12](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- c(
  "boxplots.Rmd",
  "tstat.Rmd",
  "ttest.Rmd",
  "regression.Rmd",
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with Canvas test in temporary directory
exams2canvas(myexam, n = 3, dir = mydir)
dir(mydir)
```

exams2evaexam

Generation of Exam Exercises in EvaExam Format

Description

Interface for generating text files for import in the exam system EvaExam.

Usage

```
exams2evaexam(file, n = 1L, dir = ".", name = "evaexam",
  quiet = TRUE, resolution = 100, width = 4, height = 4,
  points = NULL, fix_choice = TRUE, sep = ";",
  converter = "pandoc-mathjax", base64 = TRUE, ...)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is either display on the screen or the current working directory.
name	character. A name prefix for resulting file.
quiet	logical. Should output be suppressed when calling <code>xweave</code> and <code>texi2dvi</code> .
resolution, width, height, points	numeric, passed to <code>xweave</code> .
fix_choice	logical. Should math markup be removed in single and multiple choice lists?
sep	character. Separator for columns passed on to <code>write.table</code> .
converter, base64, ...	arguments passed on to <code>make_exercise_transform_html</code> , controlling the HTML formatting of the question texts.

Details

`exams2evaexam` generates exams in semicolon- or comma-separated values (CSV) format that can be imported into the exam system EvaExam (<https://evasys.de/evaexam/>) using `xexams`. In particular, single-choice and multiple-choice exercises are fully supported while num and string question are converted to open-ended text questions.

CAUTION: Various formatting features do not work properly yet, in particular images will be dropped upon import into EvaExam, and mathematical equations are not rendered correctly.

The function proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting Markdown/LaTeX text, (3) transforming the text to HTML, and (4) embedding the HTML text into the CSV format for EvaExam. For steps (1) and (2) the standard drivers in `xexams` are used. For step (3) a suitable transformation function is set up on the fly using `make_exercise_transform_pandoc`. For step (4) a custom writer function is set up on the fly.

Value

A list of exams as generated by `xexams` is returned invisibly.

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)
```

```
## output directory
dir.create(mydir <- tempfile())

## create a CSV file evaexam-1.csv for import in EvaExam
exams2evaexam(c("swisscapital.Rmd", "capitals.Rmd", "deriv2.Rmd",
  "boxplots.Rmd", "ttest.Rmd", "function.Rmd"), dir = mydir)
```

exams2grasple

*Generation of Exercises in Grasple Format***Description**

Automatic generation of exercises in JSON format for the e-learning platform Grasple.

Usage

```
exams2grasple(file, n = 1L, dir = ".", name = NULL, quiet = TRUE,
  resolution = 100, width = 4, height = 4, svg = FALSE, encoding = "UTF-8",
  envir = NULL, engine = NULL,
  converter = "pandoc-mathjax", zip = TRUE, use_solutionlist = TRUE,
  license_name = NULL, license_description = NULL, license_value = NULL,
  license_link = NULL, ...)
```

```
make_exams_write_grasple(name = NULL, license_name = NULL,
  license_description = NULL, license_value = NULL, license_link = NULL,
  zip = TRUE, use_solutionlist = TRUE)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is the current working directory.
name	character. A name prefix for resulting ZIP file.
quiet	logical. Should output be suppressed when calling xweave and texi2dvi .
resolution, width, height	numeric, passed to xweave .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to xweave (which passes it to knit).
engine	argument passed to xweave indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
converter	character passed on to make_exercise_transform_html , indicating the type of converter from LaTeX/Markdown to the specific requirements in Grasple. To accomplish this the converter must be set to "pandoc-mathjax".

<code>zip</code>	logical. Should the resulting JSON file(s) be zipped?
<code>use_solutionlist</code>	logical. By default it is assumed that for schoice items the solutionlist contains separate feedback for each option. If set to FALSE the general feedback is placed in the solutionlist for each option separately.
<code>license_name, license_description, license_value, license_link</code>	character, arguments for specifying the copyright license for the exercise(s). Defaults to "Non-explicit license".
<code>...</code>	arguments passed on to <code>xexams</code> .

Details

`exams2grasple` generates exercises in the JSON format for Grasple using `xexams`. It proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting Markdown or LaTeX text, (3) transforming the text to Markdown, and (4) embedding the Markdown text into the JSON format for Grasple.

For steps (1) and (2) the standard drivers in `xexams` are used. For step (3) a suitable transformation function is set up on the fly using `make_exercise_transform_html`. For step (4) a simple writer function is set up on the fly that embeds the transformed Markdown code into a hard-coded JSON template using `toJSON` and writes a JSON file for each exercise and by default bundles the collection in a ZIP file.

Note that Grasple only officially supports `schoice` and `num` items, and hence other item types are not supported in the current version of `exams2grasple` either. If the function is used for other types the execution of the function is stopped and a warning is issued.

For `num` exercises the content as specified under the `solution` environment in the Rmd/Rnw files is presented as feedback in Grasple. By contrast, for `schoice` items Grasple requires separate feedback for each option. This can be accomplished in two ways. First (the default), by putting unique feedback for each option in the `items` of the `solutionlist` in the Rmd/Rnw file. Second by putting a general feedback in the `solution` environment and setting `use_solutionlist` to `FALSE`, which creates a `solutionlist` with this general feedback for each option.

If the meta-information of an exercise contains an `exsection` element, its content will be transferred to the `name` element of the JSON-file. This `name` element is searchable in Grasple. In the ShareStats project the `exsection` element is used for classifying items in terms of its taxonomy of topics in statistics education.

Within the HTML of Grasple exercises, LaTeX elements must be embedded within `\\(\\)` blocks. After conversion using "pandoc-mathjax" the function employs further tweaks to meet Grasple's requirements. Also, in case of displaying multiline equations Grasple only allows for using the `align` environment. Environments `eqnarray` and `eqnarray*` are automatically converted into `align`. It is uncertain how pandoc deals with other multiline environments.

Value

A list of exercises as generated by `xexams` is returned invisibly.

References

Grasple - Open Education (2022). *Format for Open Interactive Math Exercises*. <https://github.com/grasple/open-format>

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define a list of exercises
myexam <- list(
  "fruit.Rmd",
  "tstat.Rmd",
  "regression.Rmd",
  "swisscapital.Rmd",
  "tstat2.Rmd",
  "dist3.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## Not run:
## generate .zip with GraspLe exercises and use general feedback
exams2graspLe(myexam, dir = mydir, use_solutionlist = FALSE)
dir(mydir)

## End(Not run)
```

exams2html

Generation of Exams in HTML Format

Description

Automatic generation of exams in HTML format.

Usage

```
exams2html(file, n = 1L, nsamp = NULL, dir = ".", template = "plain.html",
  name = NULL, quiet = TRUE, edir = NULL, tdir = NULL, sdir = NULL, verbose = FALSE,
  rds = FALSE, question = "<h4>Question</h4>", solution = "<h4>Solution</h4>",
  mathjax = NULL, resolution = 100, width = 4, height = 4, svg = FALSE,
  encoding = "UTF-8", envir = NULL, engine = NULL, converter = NULL, seed = NULL,
  exshuffle = NULL, ...)

make_exercise_transform_html(converter = c("ttm", "tth", "pandoc", "tex2image"),
  base64 = TRUE, options = NULL, ...)

make_exams_write_html(template = "plain", name = NULL,
  question = "<h4>Question</h4>", solution = "<h4>Solution</h4>",
  mathjax = FALSE)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character specifying the output directory (default: current working directory). If only a single HTML file is produced and no dir is explicitly specified, the file is displayed in the browser rather than saved in dir.
template	character. A specification of a HTML template.
name	character. A name prefix for resulting exercises.
quiet	logical. Should output be suppressed when calling <code>xweave</code> ?
edir	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
tdir	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
sdir	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
verbose	logical. Should information on progress of exam generation be reported?
rds	logical indicating whether the return list should also be saved as an RDS data file.
question	character or logical. Should the question be included in the HTML output? If question is a character it will be used as a header for resulting questions. Argument question may also be a vector that controls the output for the templates.
solution	character or logical, see argument question.
mathjax	logical. Should the JavaScript from https://www.MathJax.org/ be included for rendering mathematical formulas? By default mathjax = FALSE unless converter = "pandoc-mathjax".
resolution, width, height	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
engine	argument passed to <code>xweave</code> indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
base64	logical. Should supplementary files be embedded using Base 64 coding? Argument base64 may also be a character vector of file suffixes that should be encoded, e.g. base64 = c("png", "rda") will only encode PNG images and binary .rda files. If set to NULL only image files will be encoded.

converter, ...	arguments passed on to <code>make_exercise_transform_html</code> . The default for <code>converter</code> is set to "ttm" unless there are Rmd exercises in <code>file</code> where "pandoc" is used.
seed	integer matrix or logical. Either NULL (default), logical, or a matrix of random seeds for each possible exercise to be set prior to calling <code>driver@sweave</code> . If NULL no random seeds are set. If a matrix, the number of rows must be <code>n</code> and the number of columns must correspond to <code>unlist(file)</code> . If TRUE a suitable matrix of seeds is sampled.
exshuffle	logical or integer. If the <code>exshuffle</code> argument is non-NULL it is used to overrule the <code>exshuffle</code> tag from the <code>file</code> (e.g., <code>exshuffle = FALSE</code> can be used to keep all available answers without permutation).
options	list of options to be passed on to <code>pandoc_convert</code> .

Details

`exams2html` generates exams in a very simple HTML format using `xexams`. It proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting LaTeX code, (3) transforming the LaTeX code to HTML, and (4) embedding the HTML code in a template (a simple and plain template is used by default).

For steps (1) and (2) the standard drivers in `xexams` are used.

For step (3) a suitable transformation function is set up on the fly using `make_exercise_transform_html`. This transforms the LaTeX code in `question/questionlist` and `solution/solutionlist` by leveraging one of four functions: `ttm` produces HTML with MathML for mathematical formulas, `tth` produces plain HTML that aims to emulate mathematical formulas, `pandoc_convert` employs `pandoc` offering different options for handling formulas, and `tex2image` runs LaTeX and turns the result into a single image. In all cases, images can either be stored in supplementary files or embedded directly in Base 64 coding.

For step (4) a simple writer function is set up on the fly that embeds the transformed HTML code into a template and writes a single HTML file for each exam.

Value

`exams2html` returns a list of exams as generated by `xexams`.

`make_exercise_transform_html` returns a function that is suitable for being supplied as `driver$transform` to `xexams`.

`make_exams_write_html` returns a function that is suitable for being supplied as `driver$write` to `xexams`.

References

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

`xexams`, `ttm`, `tth`, `pandoc_convert`, `tex2image`, `browseURL`

Examples

```
## load package and enforce par(ask = FALSE)
options(device.ask.default = FALSE)

if(interactive() && requireNamespace("png")) {
## compile a single random exam (displayed in the browser)
exams2html(list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  "scatterplot.Rmd",
  "relfreq.Rmd"
))

## various versions of displaying mathematical formulae

## via MathML (displayed correctly in MathML-aware browsers, e.g. Firefox)
exams2html("tstat")

## via MathML + MathJax (should work in all major browsers,
## note the display options you get when right-clicking on the formulas
## in the browser)
exams2html("tstat", mathjax = TRUE)

## via plain HTML (works in all browsers but with inferior formatting)
exams2html("tstat", converter = "tth")

## via HTML with embedded picture (works in all browsers but
## is slow and requires LaTeX and ImageMagick)
## Not run:
exams2html("tstat", converter = "tex2image")

## End(Not run)
}
```

exams2ilias

Generation of Exams in ILIAS Format

Description

Automatic generation of exams in QTI 1.2 with some tweaks (still under development) for the learning management system ILIAS.

Usage

```
exams2ilias(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL, tdir = NULL, sdir = NULL,
  verbose = FALSE, resolution = 100, width = 4, height = 4, svg = FALSE,
  encoding = "UTF-8", num = list(fix_num = FALSE, minvalue = NA),
```

```

mchoice = list(maxchars = c(3, NA, 3), minvalue = NA),
schoice = mchoice, string = NULL, cloze = NULL,
template = "ilias",
duration = NULL, stitle = "Exercise", ititle = "Question",
adescription = "Please solve the following exercises.",
sdescription = "Please answer the following question.",
maxattempts = 1, cutvalue = 0, solutionswitch = TRUE, zip = TRUE,
points = NULL, eval = list(partial = TRUE, negative = FALSE),
converter = "pandoc-mathjax", xmlcollapse = TRUE,
metasolution = FALSE, ...)

```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from file.
<code>nsamp</code>	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises and ZIP file.
<code>quiet</code>	logical. Should output be suppressed when calling <code>xweave</code> ?
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>resolution, width, height</code>	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
<code>svg</code>	logical. Should graphics be rendered in SVG or PNG (default)?
<code>encoding</code>	character, ignored. The encoding is always assumed to be UTF-8.
<code>num</code>	function or named list applied to numerical (i.e., type num) questions. If num is a function, num will be used for generating the item body of the question, see function <code>make_itembody_qti12()</code> (or <code>make_itembody_qti21()</code>). If num is a named list, these arguments will be passed to function <code>make_itembody_qti12()</code> (or <code>make_itembody_qti21()</code> using <code>exams2qti21()</code>).
<code>mchoice, schoice, string, cloze</code>	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type mchoice, schoice, string, and cloze), respectively. See argument num for more details.
<code>template</code>	character. The IMS QTI 1.2 or 2.1 template that should be used. Currently, the package provides "qti12.xml" and "qti21.xml".

<code>duration</code>	integer. Set the duration of the exam in minutes.
<code>stitle</code>	character. A title that should be used for the sections. May be a vector of length 1 to use the same title for each section, or a vector containing different section titles.
<code>ititle</code>	character. A title that should be used for the assessment items. May be a vector of length 1 to use the same title for each item, or a vector containing different item titles. Note that the maximum of different item titles is the number of sections/questions that are used for the exam.
<code>adescription</code>	character. Description (of length 1) for the overall assessment (i.e., exam).
<code>sdescription</code>	character. Vector of descriptions for each section, omitted if empty (or NULL or FALSE).
<code>maxattempts</code>	integer. The maximum attempts for one question. This may also be a vector so that the maximum number of attempts varies across questions. A value of Inf or 0 signals that the attempts per question are not limited.
<code>cutvalue</code>	numeric. The cutvalue at which the exam is passed.
<code>solutionswitch</code>	logical. Should the question/item solutionswitch be enabled? In OLAT this means that the correct solution is shown after an incorrect solution was entered by an examinee (i.e., this is typically only useful if <code>maxattempts = 1</code>).
<code>zip</code>	logical. Should the resulting XML file (plus supplements) be zipped?
<code>points</code>	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an <code>"\expoints{}</code> " tag in the <code>.Rnw</code> file. The vector of points supplied is expanded to the number of exercises in the exam.
<code>eval</code>	named list, specifies the settings for the evaluation policy, see function <code>exams_eval</code> .
<code>converter</code>	character. Argument passed on to <code>make_exercise_transform_html</code> . The default for <code>converter</code> is set to <code>"tm"</code> unless there are Rmd exercises in file where <code>"pandoc"</code> is used.
<code>xmlcollapse</code>	logical or character. Should line breaks be collapsed in the XML code. If TRUE everything is collapsed with spaces (<code>" "</code>) but other collapse characters could be supplied.
<code>metasolution</code>	logical. Should the solution be added to the XML output as <code>qtimetadata</code> tag? This currently only works for ILIAS essay questions.
<code>...</code>	further arguments passed on to <code>make_exercise_transform_html</code> .

Details

`exams2ilias` is a convenience interface to `exams2qti12` for generating QTI 1.2 with some tweaks for ILIAS (<https://www.ilias.de/>). The support for ILIAS so far is somewhat rudimentary. Not all question types are currently supported. What has been tested most extensively, are string questions with open answers and solutions; these are rendered into ILIAS essay questions within a question pool. Also `mchoice` and `schoice` questions render in ILIAS, but have received less amount of testing. Numeric and cloze questions do not work yet.

Value

exams2ilias returns a list of exams as generated by [xexams](#).

If zip is TRUE (default), an ILIAS Questionpool object is generated.

See Also

[exams2qti12](#),

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  ## multiple-choice with graphics
  c("boxplots.Rmd", "scatterplot.Rmd"),

  ## multiple-choice with output or tables
  c("ttest.Rmd", "anova.Rmd", "relfreq.Rmd"),

  ## single-choice
  c("deriv2.Rmd", "swisscapital.Rmd"),

  ## string (closed)
  "function.Rmd",

  ## string (open-ended)
  "essayreg.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with ILIAS Questionpool in temporary directory
## using a few customization options
exams2ilias(myexam, n = 3, dir = mydir,
  maxattempts = 3,
  num = list(digits = 1),
  mchoice = list(shuffle = TRUE, enumerate = FALSE),
  solutionswitch = FALSE
)
dir(mydir)
```

Description

Automatic generation of exams in QTI 2.1 with some tweaks optimized for Inspera.

Usage

```
exams2inspera(file, n = 1L, dir = ".", name = "inspera", template = "inspera",
  converter = "pandoc-mathml", base64 = FALSE, ...)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is the current working directory.
name	character. A name prefix for resulting exercises and ZIP file.
template	character. The QTI 2.1 template that should be used, defaulting to "inspera.xml" provided within the package.
converter	character passed on to make_exercise_transform_html , indicating the type of converter from LaTeX/Markdown to HTML. Defaults to HTML with MathML.
base64	logical. Should supplementary files be embedded using Base 64 coding? Currently only FALSE is supported by Inspera.
...	arguments passed on to exams2qti21 , providing many further arguments for customization.

Details

exams2inspera is a convenience interface to [exams2qti21](#) for generating QTI 2.1 output with some small tweaks for Inspera. Most importantly, no full "assessment" is exported but only a collection of individual exercises. Consequently, all options from exams2qti21 that pertain to the configuration of the assessment (e.g., maxattempts, cutvalue, navigation, etc.) have no effect here. Moreover, MathJax is used for type-setting mathematical content by default.

Value

exams2inspera returns a list of exams as generated by [xexams](#).

See Also

[exams2qti21](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
```

```

c("tstat.Rmd", "ttest.Rmd", "confint2.Rmd"),
c("regression.Rmd", "anova.Rmd"),
c("scatterplot.Rmd", "boxhist.Rmd"),
"relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with exercises for Inspira in temporary directory
exams2inspera(myexam, n = 3, dir = mydir)
dir(mydir)

```

exams2kahoot

Generation of Excel Sheets with Quiz Questions for Kahoot!

Description

Interface for generating Excel sheets with plain-text quiz questions for the Kahoot! game-based learning platform.

Usage

```

exams2kahoot(file, n = 1L, dir = ".", name = "kahootquiz",
  quiet = TRUE, time = NULL, ...)

```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is the current working directory.
name	character. A name prefix for resulting Excel sheets.
quiet	logical. Should output be suppressed when calling xweave?
time	numeric. Time limit (in seconds) allowed for answering the question in Kahoot!, must be one of 5, 10, 20, 30, 60, 90, 120, 240. If other numeric inputs are made, they are forced to the nearest admissible specification. Default: 60 seconds.
...	arguments passed on to xexams .

Details

exams2kahoot generates Excel sheets, requiring [write.xlsx](#) from the **openxlsx** package, that can be imported to the Kahoot! game-based learning platform. Only single-choice (schoice) and multiple-choice (mchoice) questions are allowed with plain questions and answers (using conversion via [pandoc_convert](#)). Questions and answers must not exceed 120 and 75 characters, respectively.

Value

A list of exams as generated by [xexams](#) is returned invisibly.

Examples

```
## Not run:
## create an .xlsx file for Kahoot! (requiring openxlsx) based
## on three simple text-based single-choice and multiple-choice questions
exams2kahoot(c("capitals", "swisscapital", "switzerland"))

## End(Not run)
```

exams2lops

Generation of Exams in LOPS Exam Server Format (WU Wien)

Description

Automatic generation of exams in LOPS exam server format (WU Wien).

Usage

```
exams2lops(file, n = 1L, nsamp = NULL, dir = ".", name = NULL,
  quiet = TRUE, edir = NULL, tdir = NULL, sdir = NULL, verbose = FALSE,
  solution = TRUE, doctype = NULL, head = NULL, resolution = 100, width = 4,
  height = 4, svg = FALSE, encoding = "UTF-8", envir = NULL, engine = NULL,
  converter = "tex2image", base64 = FALSE,
  auto_scramble = TRUE, ...)
```

```
make_exams_write_lops(name = NULL, auto_scramble = TRUE, ...)
```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from <code>file</code> .
<code>nsamp</code>	integer. The number(s) of exercise files sampled from each list element of <code>file</code> . Sampling without replacement is used if possible. (Only if some element of <code>nsamp</code> is larger than the length of the corresponding element in <code>file</code> , sampling with replacement is used.)
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises.
<code>quiet</code>	logical. Should output be suppressed when calling xweave ?
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in <code>file</code> are stored (see also xexams).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via tempfile . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.

<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via tempfile .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>solution</code>	logical. Should the solution be included in the HTML output?
<code>doctype</code>	character vector with a DOCTYPE tag for the HTML page. By default HTML4 is employed.
<code>head</code>	character vector for the head tag. By default a simple header is employed, setting the font to Arial.
<code>resolution, width, height</code>	numeric. Options for rendering PNG (or SVG) graphics passed to xweave .
<code>svg</code>	logical. Should graphics be rendered in SVG or PNG (default)?
<code>encoding</code>	character, ignored. The encoding is always assumed to be UTF-8.
<code>envir</code>	argument passed to xweave (which passes it to knit).
<code>engine</code>	argument passed to xweave indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
<code>converter</code>	character. Workhorse function for transforming LaTeX code to HTML.
<code>base64</code>	logical. Should supplementary files be embedded using Base 64 coding? Argument <code>base64</code> may also be a character vector of file suffixes that should be encoded, e.g. <code>base64 = c("png", "rda")</code> will only encode PNG images and binary <code>.rda</code> files. If set to NULL only image files will be encoded.
<code>auto_scramble</code>	logical. Should answers be scrambled automaticall?
<code>...</code>	arguments passed on to <code>make_exercise_transform_html</code> .

Details

`exams2lops` will produce a `.zip` file that may be uploaded. It proceeds by (1) calling [xweave](#) on each exercise, (2) reading the resulting LaTeX code, (3) transforming the LaTeX code to HTML, and (4) embedding the HTML code in a XML file using the LOPS exam server XML format (WU Wien).

For steps (1) and (2) the standard drivers in `xexams` are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in [exams2html](#).

For step (4) a simple writer function is set up on the fly that embeds the transformed HTML code into the final XML files for each question and the exam.

Note that in `make_exams_write_lops` only multiple and single choice questions are supported at the moment, since the LOPS exam server XML format (WU Wien) is used to generate printed versions for large scale multiple choice exams. In addition, only images of the question/questionlist/solution/solutionlist should be generated, since the server has only minimum support for e.g. MathML markup used to produce mathematical formulas.

Value

`exams2lops` returns a list of exams as generated by [xexams](#).

`make_exams_write_lops` returns a function that generates the XML code for the question in LOPS exam server format (WU Wien).

References

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

[xexams](#), [ttm](#), [tth](#), [tex2image](#), [make_exercise_transform_html](#),

Examples

```
## Not run:
## output directory
dir.create(mydir <- tempfile())

## generate the exam
exams2lops(c("scatterplot.Rmd", "boxplots.Rmd"), dir = mydir)
dir(mydir)

## End(Not run)
```

exams2moodle

Generation of Exams in Moodle XML Format

Description

Automatic generation of exams in Moodle XML format.

Usage

```
exams2moodle(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL,
  tdir = NULL, sdir = NULL, verbose = FALSE, rds = FALSE,
  resolution = 100, width = 4, height = 4, svg = FALSE, encoding = "UTF-8",
  iname = TRUE, stitle = NULL,
  testid = FALSE, zip = FALSE, num = NULL, mchoice = NULL,
  schoice = mchoice, string = NULL, cloze = NULL,
  points = NULL, rule = NULL, pluginfile = TRUE, forcedownload = FALSE,
  converter = "pandoc-mathjax", envir = NULL, engine = NULL,
  table = "table_shade", css = NULL, ...)

make_question_moodle(name = NULL, solution = TRUE,
  shuffle = FALSE, penalty = 0, answernumbering = "abc",
  usecase = FALSE, cloze_mchoice_display = NULL, cloze_schoice_display = NULL,
  truefalse = c("True", "False"), enumerate = FALSE, abstention = NULL,
  eval = list(partial = TRUE, negative = FALSE, rule = "false2"),
  essay = NULL, numwidth = NULL, stringwidth = NULL,
  css = NULL)
```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from <code>file</code> .
<code>nsamp</code>	integer. The number(s) of exercise files sampled from each list element of <code>file</code> . Sampling without replacement is used if possible. (Only if some element of <code>nsamp</code> is larger than the length of the corresponding element in <code>file</code> , sampling with replacement is used.)
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises and ZIP file.
<code>quiet</code>	logical. Should output be suppressed when calling <code>xweave</code> ?
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in <code>file</code> are stored (see also <code>xexams</code>).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>rds</code>	logical indicating whether the return list should also be saved as an RDS data file.
<code>resolution, width, height</code>	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
<code>svg</code>	logical. Should graphics be rendered in SVG or PNG (default)?
<code>encoding</code>	character, ignored. The encoding is always assumed to be UTF-8.
<code>envir</code>	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
<code>engine</code>	argument passed to <code>xweave</code> indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
<code>table</code>	character or logical. Control the style for tables in an exercise via a custom class: "table_shade" (equivalent to <code>table = TRUE</code>), "table_rule", and "table_grid" being provided. See also the details below.
<code>css</code>	character. A character string (or vector) containing the path(s) to CSS style file(s). Alternatively, a string (or vector) with a <code><style></code> tag to be included in every exercise. This is used internally for the table styles above, see also the details below.
<code>iname</code>	logical. Should the exam name be included in the path in the <code><category></code> tag in the final XML file? This option may be useful when questions should be added to certain already existing question banks, i.e. <code>iname = TRUE</code> will include the exam name by <code>\$course\$/ExamName/</code> .
<code>stitle</code>	character. For the questions specified in argument <code>file</code> , additional section titles may be set. The section titles will then be added to the <code><category></code> tag in the final XML file (see also argument <code>iname</code>), i.e. the section name for each question will be written to <code>\$course\$/ExamName/SectionName</code> . Note that section

names may also be provided in the `\exsection{}` tag in the exercise file of the question. However, section names that are specified in `stitle` will overwrite `\exsection{}` tags. `stitle` may also include NA, e.g. `stitle = c("Exercise 1", NA, "Exercise 3")`.

<code>testid</code>	logical. Should an unique test id be added to the exam name.
<code>zip</code>	logical. Should the resulting XML file be zipped?
<code>num</code>	function or named list applied to numerical (i.e., <code>num</code>) questions. If <code>num</code> is a function, this will be used to set up the question XML code. If <code>num</code> is a list, such a function is generated on the fly via <code>make_question_moodle</code> using the arguments in the list. For example, <code>num = list(solution = FALSE)</code> can be used to suppress embedding the solution text in the XML.
<code>mchoice, schoice, string, cloze</code>	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type <code>mchoice</code> , <code>schoice</code> , <code>string</code> , and <code>cloze</code>), respectively. For more guidance see argument <code>num</code> and the details below.
<code>points</code>	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an <code>"\expoints{ }"</code> tag in the exercise file. The vector of points supplied is expanded to the number of exercises in the exam.
<code>rule</code>	character specifying which rule to use for negative partial credits. See function exams_eval . Note that Moodle is somewhat restrictive about the number of multiple-choice alternatives when using partial credits (see below for details).
<code>pluginfile</code>	logical. Should supplements be included in the Moodle XML file via Moodle's Pluginfile mechanism? This is the default but may not work with older versions of Moodle (<2.5). If set to <code>FALSE</code> supplements like graphics and data are included as data URIs.
<code>forcedownload</code>	logical. Should all supplementary links be forced to download when clicked (as opposed to opening in the browser)? Only supported if <code>pluginfile = TRUE</code> . If <code>forcedownload = FALSE</code> the behavior typically depends on the browser, user settings, and file type.
<code>solution</code>	logical. Should the question solution, if available, be added in the question XML?
<code>shuffle</code>	For <code>mchoice</code> and <code>schoice</code> exercises, if set to <code>TRUE</code> will force Moodle to additionally shuffle the provided answer list.
<code>penalty</code>	numeric. Specifies the penalty tag for a question.
<code>answernumbering</code>	character. Specifies how choice questions should be numbered, allowed values are: <code>"abc"</code> (default), <code>"ABCD"</code> , <code>"123"</code> or <code>"none"</code> .
<code>usecase</code>	logical. Should string questions be case sensitive or not.
<code>cloze_mchoice_display, cloze_schoice_display</code>	character. In <code>cloze</code> questions, the user may set the visual appearance of choice questions. If <code>NULL</code> (default), <code>"MULTIRESPONSE"</code> (column of checkboxes) is used for <code>mchoice</code> questions and <code>"MULTICHOICE"</code> (drop-down menu) for <code>schoice</code> questions unless math markup is present in the question list. The latter is not rendered in drop-down menus and hence <code>"MULTICHOICE_V"</code> (radio buttons, vertical

	column) are used. Other options include a horizontal row of either checkboxes ("MULTIRESPONSE_H") or radio buttons ("MULTICHOICE_H"), respectively. Shuffled versions of all display types are also available (since Moodle 3.0) by appending an "S", e.g., "MULTICHOICE_S" or "MULTICHOICE_VS" etc.
truefalse	character of length 2. For single choice answers in cloze questions, the user may specify the possible options shown.
enumerate	logical. In cloze questions, if set to TRUE, the answerlist and solutionlist will be enumerated.
abstention	character or logical. Should an explicit abstention option be added in single/multiple choice exercises? The character text specified is used for an extra button in Moodle which (when selected) always leads to zero points.
eval	named list, specifies the settings for the evaluation policy, see function exams_eval .
essay	logical. Should string questions be rendered into Moodle shortanswer or essay questions? The default is to use shortanswer unless either essay=TRUE or the exercise's meta-information is set to essay.
numwidth, stringwidth	logical, numeric, or character. Should the width of all num or string sub-items, respectively, in a cloze be fixed to the same width? This can be accomplished by adding a wrong solution with a suitable length to all sub-items in Moodle XML. The default (NULL or equivalently FALSE) is not to do so but let Moodle decide the width of each cell based on the respective correct solution. Alternatively, the arguments can be set to TRUE (then the maximum width from the correct solutions is used), an integer (indicating the maximum width) or a character (like "1111111", to be used as the wrong solution). Both arguments can also be set through exextra tags in each of the exercises' meta-information.
converter, ...	arguments passed on to make_exercise_transform_html. The default for converter is "pandoc-mathjax" which assumes that the quiz is imported in a Moodle site with MathJax plugin activated (which is the default setting in Moodle). For using MathML instead of MathJax the converter can be set to NULL or "pandoc-mathml" etc. For details see Zeileis (2019).

Details

exams2moodle produces an XML file that may be uploaded into Moodle. It proceeds by (1) calling [xweave](#) on each exercise, (2) reading the resulting Markdown or LaTeX text, (3) transforming the text to HTML, and (4) embedding the HTML code in a Moodle XML file for exams/quizzes.

For steps (1) and (2) the standard drivers in [xexams](#) are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in [exams2html](#).

For step (4), the function will cycle through all questions and exams to generate the final Moodle XML file. The structure of the resulting XML file is such that one category will be set for the exam/quiz using the exam/quiz name (or this category may be suppressed (i.e., not included in the XML) by setting `iname = FALSE`), followed by one category/section for each question, while the replicates of each question will be included in the corresponding category/section. Note that category/section names may also be provided in the `exsection` tag in the exercise files, or within argument `stitle` in `exams2moodle`. This may be useful when questions should automatically be added to already existing Moodle question banks. (See also the argument descriptions above.)

The XML code for each question type (numeric, multiple-choice, etc.) is set up by separate functions that can be specified through the separate arguments `num`, `mchoice`, `schoice`, `string`, and `cloze` in `exams2moodle`. While it is possible to pass a suitable function to these arguments, it is more common to set suitable functions up on the fly using `make_question_moodle`. In this case, the arguments `num`, `mchoice`, `schoice`, `string` and `cloze` can be lists of arguments to pass on to `make_question_moodle`. For example, to suppress numbering the multiple-choice answer items with `a/b/c/...` one has to specify `mchoice = list(answernumbering = "none")` (which, by default, also gets passed on to `schoice`).

When using partial credits for multiple-choice exercises, only certain numbers of alternatives are supported in Moodle. This is because the Moodle XML format just supports certain percentages which can be added or subtracted from the score of an item. Therefore, it may not be possible to use partial credits for certain combinations of true and false answer alternatives when the overall number of alternatives is greater than 10.

When specifying cloze exercises, two approaches are possible: Either a `answerlist` with all questions is provided within the question or, alternatively, the answer fields can be placed anywhere in the question text. For the latter, the strings `##ANSWER1##`, `##ANSWER2##`, etc., have to be used, see the exercises `"boxhist2"` and `"fourfold2"` for illustration and Appendix C in Zeileis et al. (2014) for further details.

To fix the width of numeric answer fields withing cloze exercises (in order not to convey any clues about the length of the correct solution), the `exextra[numwidth]` metainformation command can be used in the exercise file. For example, it can be set to `\exextra[numwidth,logical]{TRUE}`, `\exextra[numwidth,numeric]{5}`, or `\exextra[numwidth,character]{100.0}`.

In order to generate open-ended text questions in Moodle one can use `string` questions and then additionally set `exstringtype` to `essay` and/or `file`. See the `"essayreg"` question for a worked example. On top of the basic `exstringtype` one can make further Moodle-specific customizations via some `exextra` options, namely:

- `essay`: logical. Enables the essay function.
- `format`: character. Type of text field (one of: `plain`, `editor`, `editorfilepicker` `monospaced` `noinline`)
- `required`: logical. Whether an answer is required.
- `attachments`: numeric. How many attachments can be uploaded.
- `attachmentsrequired`: numeric. The number of required attachments.

To control the style used for rendering the HTML in Moodle exercises, it is possible to include some custom CSS (cascading style sheets) code via the argument `css`. In particular, the `exams2moodle` function leverages this for table formatting. It includes its own CSS for this purpose if one of the classes `"table_shade"` (rows highlighted with different shades of gray), `"table_rule"` (with horizontal lines), or `"table_grid"` (with both horizontal and vertical lines) is used.

Value

`exams2moodle` returns a list of exams as generated by `xexams`.

`make_question_moodle` returns a function that generates the XML code for the question in Moodle's XML standard.

References

- Dougiamas M, et al. (2022). *Moodle, Version 4.0*. <https://moodle.org/>.
- MoodleDocs (2022). *Moodle XML Format*. https://docs.moodle.org/en/Moodle_XML
- Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.
- Zeileis A (2019). *Mathematical Notation in Online R/exams*. <https://www.R-exams.org/tutorials/math/>

See Also

[xexams](#), [ttm](#), [tth](#), [tex2image](#), [make_exercise_transform_html](#),

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  c("scatterplot.Rmd", "boxhist.Rmd"),
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate moodle quiz in temporary directory
## using a few customization options
exams2moodle(myexam, n = 3, dir = mydir,
  num = list(solution = FALSE),
  mchoice = list(shuffle = TRUE)
)
dir(mydir)
```

Description

Generation of exams in PDF format that can be printed, scanned, and evaluated automatically.

Usage

```
exams2nops(file, n = 1L, nsamp = NULL, dir = ".", name = NULL,
  language = "en", title = "Exam", course = "",
  institution = "R University", logo = "Rlogo.png", date = Sys.Date(),
  replacement = FALSE, intro = NULL, blank = NULL, duplex = TRUE, pages = NULL,
  usepackage = NULL, header = NULL, encoding = "UTF-8", startid = 1L,
  points = NULL, showpoints = FALSE, samepage = FALSE, newpage = FALSE,
  twocolumn = FALSE, reglength = 7L, seed = NULL, ...)
```

```
make_nops_template(n, replacement = FALSE, intro = NULL, blank = NULL,
  duplex = TRUE, pages = NULL, file = NULL, nchoice = 5, encoding = "UTF-8",
  samepage = FALSE, newpage = FALSE, twocolumn = FALSE, reglength = 7L)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file (in exams2nops) and the number of exercises per exam (in make_nops_template), respectively.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character. The default is either display on the screen (if n = 1L) or the current working directory.
name	character. A name prefix for resulting exams and RDS file.
language	character. Path to a DCF file with a language specification. See below for the list of supported languages.
title	character. Title of the exam, e.g., "Introduction to Statistics".
course	character. Optional course number, e.g., "101".
institution	character. Name of the institution at which the exam is conducted.
logo	character. Path to a logo image (in a file format supported by pdfLaTeX). If set to NULL, the logo is omitted.
date	character or "Date" object specifying the date of the exam.
replacement	logical. Should a replacement exam sheet be included?
intro	character. Either a single string with the path to a .tex file or a vector with with LaTeX code for optional introduction text on the first page of the exam.
blank	integer. Number of blank pages to be added at the end. (Default is chosen to be half of the number of exercises.) If pages is specified, blank can also be a vector of length two with blank pages before and after the extra pages, respectively.
duplex	logical. Should blank pages be added after the title page (for duplex printing)?
pages	character. Path(s) to additional PDF pages to be included at the end of the exam (e.g., formulary or distribution tables).
usepackage	character. Names of additional LaTeX packages to be included.

header	character vector or list. Either a character vector with LaTeX code to include in the header or a named list with further options to be passed to the LaTeX files.
encoding	character, ignored. The encoding is always assumed to be UTF-8.
startid	integer. Starting ID for the exam numbers (defaults to 1).
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within the <code>expoints</code> tags of the exercise files (if any). The vector of points supplied should either have length 1 or the number of exercises in the exam.
showpoints	logical. Should the PDF show the number of points associated with each exercise (if specified in the <code>Rnw/Rmd</code> exercise or in <code>points</code>)?
samepage	logical. Should the itemized question lists be forced to be on the same page?
newpage	logical. Should each exercise start on a new page? (Technically, a page break is added after each exercise.)
twocolumn	logical. Should a two-column layout be used?
reglength	integer. Number of digits in the registration ID. The default is 7 and it can be increased up to 10. In case of <code>reglength < 7</code> , internally <code>reglength = 7</code> is enforced (and thus necessary in the registration CSV file) but the initial ID digits are fixed to 0 in the exam sheet and corresponding boxes ticked already.
seed	integer matrix or logical. Either NULL (default), logical, or a matrix of random seeds for each possible exercise to be set prior to calling <code>driver@sweave</code> . If NULL no random seeds are set. If a matrix, the number of rows must be <code>n</code> and the number of columns must correspond to <code>unlist(file)</code> . If TRUE a suitable matrix of seeds is sampled.
...	arguments passed on to <code>exams2pdf</code> .
nchoice	character. The number of choice alternatives per exercise.

Details

`exams2nops` is a convenience interface for `exams2pdf` with a dynamically generated title page which can be printed, scanned with `nops_scan` and evaluated automatically by `nops_eval`. It is originally intended for single- and multiple choice (`schoice/mchoice`) questions only but has also some limited support for open-ended (string) questions.

The exam sheet consists of various sections where information is either printed or filled in by the students. The section with personal data is just for human readers, it is not read automatically. The registration number has to be filled in in digits and also marked with corresponding crosses where only the latter is read automatically. The exam ID/type/scrambling are printed directly into the PDF and read automatically after scanning. Note that the font in the PDF must not be modified for the reading step to work reliably. (A sans-serif font is used and hence the `sfnorm` LaTeX package is also used - if it is installed.) The questions can have up to five alternatives which have to be answered by the students. The crosses are read automatically where both empty and completely filled boxes are regarded as not crossed.

Tutorial for NOPS workflow: <https://www.R-exams.org/tutorials/exams2nops/>.

Limitations: (a) Only up to five answer alternatives per question are supported. (b) Currently, only up to 45 questions are supported. If you have more questions, consider splitting the entire

exam up into two NOPS exams. (c) Only up to 3 open-ended questions can be included. (d) Each question must have the same number of answer alternatives and the same number of points across random replications. For example, the n replications drawn for the first exercise all need, say, five alternatives and two points. Then, the second exercise may have, say, four alternatives and five points and so on. But this may not be mixed within the same exercise number.

The examples below show how PDF exams can be generated along with an RDS file with (serialized) R data containing all meta-information about the exam. The PDFs can be printed out for conducting the exam and the exam sheet from the first page then needs to be scanned into PDF or PNG images. Then the information from these scanned images can be read by `nops_scan`, extracting information about the exam, the participants, and the corresponding answers (as described above). The ZIP file produced by `nops_scan` along with the RDS of the exam meta-information and a CSV file with participant information can then be used by `nops_eval` to automatically evaluate the whole exam and producing HTML reports for each participant. See `nops_eval` for a worked example.

Currently, up to three open-ended string questions can also be included. These do not generate boxes on the first exam sheet but instead a second exam sheet is produced for these open-ended questions. It is assumed that a human reader reads these open-ended questions and then assigns points by marking boxes on this separate sheet. Subsequently, this sheet can also be read by `nops_scan`.

The language elements can be specified through a relatively simple text file and the package already ships with: English ("en"), Croatian ("hr"), Danish ("da"), Dutch ("nl"), Finnish ("fi"), French ("fr"), Galician ("gl"), German ("de"), Hungarian ("hu"), Italian ("it"), Japanese ("ja"), Korean ("ko"), Norwegian (Bokmål, "no"), Portuguese ("pt-PT" or "pt-BR"; also "pt" is synonymous with "pt-PT"), Romanian ("ro"), Russian ("ru"), Serbian ("sr"), Slovak ("sk"), Slovenian ("sl"), Spanish ("es"), Swiss German ("gsw"), Turkish ("tr"), Vietnamese ("vi"). Note that the language names correspond to the ISO 639 codes (https://www.loc.gov/standards/iso639-2/php/code_list.php) or IETF language tags (https://en.wikipedia.org/wiki/IETF_language_tag) if no ISO 639 codes exists (as for Brazilian Portuguese). For more details about the underlying text file in DCF format, see https://www.R-exams.org/tutorials/nops_language/

Value

A list of exams as generated by `xexams` is returned invisibly.

Examples

```
## load package and enforce par(ask = FALSE)
## additionally, for simplicity, enforce using the basic
## tools::texi2dvi() LaTeX interface instead of the more
## flexible/robust tinytex::latexmk()
library("exams")
oopt <- options(device.ask.default = FALSE, exams_tex = "tools")

## define an exam (= list of exercises)
myexam <- list(
  "tstat2.Rmd",
  "ttest.Rmd",
  "relfreq.Rmd",
  "anova.Rmd",
  c("boxplots.Rmd", "scatterplot.Rmd"),
```

```

"cholesky.Rmd"
)

if(interactive()) {
## compile a single random exam (displayed on screen)
exams2nops(myexam, duplex = FALSE, language = "de")
}

## create multiple exams on the disk (in a
## temporary directory)
dir.create(mydir <- tempfile())

## generate NOPS exam in temporary directory
set.seed(403)
ex1 <- exams2nops(myexam, n = 2, dir = mydir)
dir(mydir)

## use a few customization options: different
## university/logo and language/title
## with a replacement sheet but for non-duplex printing
set.seed(403)
ex2 <- exams2nops(myexam, n = 2, dir = mydir,
  institution = "Universit\\\\"at Innsbruck",
  name = "uibk", logo = "uibk-logo-bw.png",
  title = "Klausur", language = "de",
  replacement = TRUE, duplex = FALSE)
dir(mydir)

options(exams_tex = oopt$exams_tex)

```

exams2openolat

Generation of Exams for OpenOlat

Description

Automatic generation of exams in QTI 2.1 (or 1.2) with some tweaks optimized for OpenOlat.

Usage

```

exams2openolat(file, n = 1L, dir = ".", name = "olattest",
  qti = "2.1", config = TRUE, converter = "pandoc-mathjax", table = TRUE,
  maxattempts = 1, cutvalue = NULL, ...)

openolat_config(cancel = FALSE, suspend = FALSE, scoreprogress = FALSE,
  questionprogress = FALSE, maxscoreitem = TRUE, menu = TRUE,
  titles = TRUE, notes = FALSE, hidelms = TRUE, hidefeedbacks = FALSE,
  blockaftersuccess = FALSE, attempts = 1, anonym = FALSE,
  manualcorrect = FALSE)

```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from <code>file</code> .
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises and ZIP file.
<code>qti</code>	character indicating whether QTI "1.2" or "2.1" (default) should be generated.
<code>config</code>	logical or named list with arguments to be passed to function <code>openolat_config</code> , for adding an OpenOlat-specific configuration file <code>QTI21PackageConfig.xml</code> to the exam. If set to <code>config = FALSE</code> , no configuration file is added (which was the default behavior up to R/exams 2.3.6). Setting <code>config = TRUE</code> is equivalent to <code>config = list()</code> or <code>config = openolat_config()</code> (which is the current default, starting from R/exams 2.4.0). Custom configurations can be specified using the syntax <code>config = list(...)</code> , see the details on the possible arguments below.
<code>converter</code>	character passed on to <code>make_exercise_transform_html</code> , indicating the type of converter from LaTeX/Markdown to HTML. Defaults to HTML with MathJax (and OpenOlat-specific fixups).
<code>table</code>	logical or character. Should a dedicated table class be used in the HTML for OpenOlat? For details see below.
<code>maxattempts</code>	integer. The maximum attempts for one question within an exam. Set to <code>Inf</code> or <code>0</code> for unlimited attempts. For a finite number of attempts this must be smaller than <code>100000</code> . See also <code>attempts</code> below for allowing more than one attempt for the exam as a whole.
<code>cutvalue</code>	numeric. The number of points at which the exam is passed. If set to <code>NULL</code> (default) or equivalently <code>NA</code> , no <code>cutvalue</code> is set and the result of the exam is just the number of points.
<code>...</code>	arguments passed on to <code>exams2qti21</code> (or <code>exams2qti12</code> , respectively). See the corresponding manual pages for further important arguments such as <code>solutionswitch</code> , <code>casesensitive</code> , <code>cloze_schoice_display</code> navigation, <code>shufflesections</code> , <code>eval</code> , <code>selection</code> , among many others.
<code>cancel</code>	logical. Are participants allowed to cancel an exam after starting it? (Default: <code>FALSE</code> .)
<code>suspend</code>	logical. Are participants allowed to suspend an exam after starting it (in order to continue it later)? (Default: <code>FALSE</code> .)
<code>scoreprogress</code>	logical. Should the progress of the score/points achieved so far be displayed during the exam? (Default: <code>FALSE</code> .)
<code>questionprogress</code>	logical. Should the number of questions solved so far be displayed during the exam? (Default: <code>FALSE</code> .)
<code>maxscoreitem</code>	logical. Should the maximum score of a question/item be displayed? (Default: <code>TRUE</code> .)
<code>menu</code>	logical. Should the menu be displayed during the exam? (Default: <code>TRUE</code> .)
<code>titles</code>	logical. Should question titles be displayed during the exam? (Default: <code>TRUE</code> .)

notes	logical. Should participants be enabled to take notes in OpenOlat during the exam? (Default: FALSE.)
hidelms	logical. Should access to the OpenOlat learning management system be hidden during the exam? (Default: TRUE.)
hidefeedbacks	logical. Should feedbacks be hidden? (Default: FALSE.)
blockaftersuccess	logical. Should the exam be blocked after successful completion? (Default: FALSE.)
attempts	integer. How many attempts are allowed for the exam as a whole? Note that this should not be confused with the <code>maxattempts</code> for a question within the exam (see above).
anonym	logical. Should anonymous users be allowed to take the exam? (Default: FALSE.)
manualcorrect	logical. Should the points and pass/fail status from the exam be evaluated manually? (Default: FALSE.)

Details

`exams2openolat` is a convenience interface to `exams2qti12` and `exams2qti21` for generating either QTI 1.2 or 2.1 (default) output with some small tweaks for OpenOlat. Specifically, the MathJax and table output from pandoc is post-processed as expected by OpenOlat. See the corresponding manual page for many more arguments that are supported by `exams2openolat`.

Dedicated table classes currently include: Full-width tables with borders ("`b_grid`", "`b_border`") or without ("`b_full`", "`b_borderless`"). Or regular tables without background color ("`b_middle`") or with white grid and color background ("`b_gray`", "`b_red`", "`b_green`", "`b_blue`", "`b_yellow`"). Setting `table = TRUE` corresponds to `table = "b_gray"`.

In addition to controlling the behavior of the exam and its questions via the standard QTI 2.1 options (see `exams2qti21`), it is possible to specify certain OpenOlat-specific configurations via an additional XML file to be added to the ZIP output. This is why these options are not part of `exams2qti21` but are provided in the separate wrapper function `openolat_config`. As the resulting XML configuration file is not part of the official OpenOlat interface, it may be subject to more changes in the future.

Value

`exams2openolat` returns a list of exams as generated by `xexams`.

See Also

[exams2qti12](#), [exams2qti21](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
```

```

myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  c("scatterplot.Rmd", "boxhist.Rmd"),
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with OpenOlat test in temporary directory
exams2openolat(myexam, n = 3, dir = mydir)
dir(mydir)

```

exams2pandoc

Generation of Exams via Pandoc

Description

Automatic generation of exams via pandoc, by default in docx format.

Usage

```

exams2pandoc(file, n = 1L, nsamp = NULL, dir = ".",
  name = "pandoc", type = "docx", template = "plain.tex",
  question = "Question", solution = "Solution",
  header = list(Date = Sys.Date()), inputs = NULL, options = NULL,
  quiet = TRUE, resolution = 100, width = 4, height = 4, svg = FALSE,
  encoding = "UTF-8", envir = NULL, engine = NULL,
  edir = NULL, tdir = NULL, sdir = NULL, verbose = FALSE,
  points = NULL, exshuffle = NULL, ...)

```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character specifying the output directory (default: current working directory). If only a single HTML file is produced and no dir is explicitly specified, the file is displayed in the browser rather than saved in dir.
name	character. A name prefix for resulting exercises.
type	character. The file type to convert to using pandoc. The default is "docx" (but other choices are also supported, e.g., "odt", "html", "markdown" etc.).

template	character. A specification of a template in either LaTeX, HTML, or Markdown format. The default is to use the "plain.tex" file provided but an alternative "plain.html" is also available.
question	character or logical. Should the question be included in the output? If question is a character it will be used as a header for resulting questions.
solution	character or logical, see argument question.
header	list. A list of named character strings (or functions generating such) to be substituted in the template.
inputs	character. Names of files that are needed as inputs for the template (e.g., images, headers). Either the full path must be given or the file needs to be in edir.
options	character. A string of options to be passed on to <code>pandoc_convert</code> .
quiet	logical. Should output be suppressed when calling <code>xweave</code> ?
resolution, width, height	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
engine	argument passed to <code>xweave</code> indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
edir	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
tdir	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
sdir	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
verbose	logical. Should information on progress of exam generation be reported?
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within the <code>expoints</code> tags of the exercise files (if any). The vector of points supplied should either have length 1 or the number of exercises in the exam.
exshuffle	logical or integer. If the <code>exshuffle</code> argument is non-NULL it is used to overrule the <code>exshuffle</code> tag from the file (e.g., <code>exshuffle = FALSE</code> can be used to keep all available answers without permutation).
...	currently not used.

Details

`exams2pandoc` can generate exams in various output formats (by default `docx`) using `xexams` and `pandoc_convert`. It proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting LaTeX or Markdown code, (3) transforming the code to the markup of some exam template (either LaTeX, HTML, or Markdown), (4) embedding the code in a template and converting it to the desired output format using `pandoc`.

For steps (1) and (2) the standard drivers in `xexams` are used.

For step (3) a suitable transformation function is set up on the fly using `make_exercise_transform_pandoc`. Depending on which format the template uses (LaTeX or HTML or Markdown) the transformation may or may not be trivial.

For step (4) all exercises are inserted into the template (and also replacing certain additional tags from header) and then `pandoc_convert` is used to convert to the desired output format (one file for each exam). In principle, all output types of pandoc are supported, but most of them have not been tested. (The main motivation for `exams2pandoc` was the generation of "docx" or "odt" files.)

Value

`exams2pandoc` returns a list of exams as generated by `xexams`.

See Also

`xexams`, `pandoc_convert`

Examples

```
## load package and enforce par(ask = FALSE)
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  c("scatterplot.Rmd", "boxhist.Rmd"),
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## compile two docx and odt versions each
set.seed(1090)
exams2pandoc(myexam, n = 2, dir = mydir, type = "docx")
set.seed(1090)
exams2pandoc(myexam, n = 2, dir = mydir, type = "odt")
```

Description

Interface for generating comma-separated files for import in the audience response system Particify.

Usage

```
exams2particify(file, n = 1L, dir = ".", name = "particify",
  quiet = TRUE, resolution = 100, width = 4, height = 4, svg = FALSE,
  abstention = FALSE, fix_choice = FALSE, exshuffle = NULL, ...)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
dir	character. The default is either display on the screen or the current working directory.
name	character. A name prefix for resulting exercises and RDS file.
quiet	logical. Should output be suppressed when calling <code>xweave</code> and <code>texi2dvi</code> .
resolution, width, height	numeric, passed to <code>xweave</code> .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
abstention	logical. Are abstentions allowed for choice questions?
fix_choice	logical. Should math markup be removed in single and multiple choice lists? (This may be needed for older Particify versions where math markup is rendered in the question itself but not the choice list.)
exshuffle	logical or integer. If the <code>exshuffle</code> argument is non-NULL it is used to overrule the <code>exshuffle</code> tag from the file (e.g., <code>exshuffle = FALSE</code> can be used to keep all available answers without permutation).
...	arguments passed on to <code>xexams</code> .

Details

`exams2particify` generates exams in comma-separated values (CSV) format that can be imported into the audience response system Particify (<https://particify.de/>) using `xexams`. In particular, single-choice and multiple-choice exercises are fully supported while num and string question are converted to open-ended text questions.

To import the generated CSV file, click on "Create question series" in Particify and then select Settings > Import/Export > Import series.

Internally, the `exams2particify` function proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting Markdown/LaTeX text, (3) transforming the text to Markdown, and (4) embedding the Markdown text into the CSV format for Particify. For steps (1) and (2) the standard drivers in `xexams` are used. For step (3) a suitable transformation function is set up on the fly using `make_exercise_transform_pandoc`. For step (4) a custom writer function is set up on the fly.

Value

A list of exams as generated by `xexams` is returned invisibly.

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## output directory
dir.create(mydir <- tempfile())

## create a CSV file particify-1.csv for import in Particify
exams2particify(c("swisscapital.Rmd", "capitals.Rmd", "deriv2.Rmd",
  "boxplots.Rmd", "ttest.Rmd", "function.Rmd"), dir = mydir)
```

exams2pdf

Generation of Exams in PDF Format

Description

Automatic generation of exams in PDF format.

Usage

```
exams2pdf(file, n = 1L, nsamp = NULL, dir = ".", template = "plain",
  inputs = NULL, header = list(Date = Sys.Date()), name = NULL,
  control = NULL, encoding = "UTF-8", quiet = TRUE, transform = NULL,
  edir = NULL, tdir = NULL, sdir = NULL, texdir = NULL, texengine = "pdflatex",
  verbose = FALSE, rds = FALSE, points = NULL, seed = NULL,
  attachfile = FALSE, exshuffle = NULL, ...)

make_exams_write_pdf(template = "plain", inputs = NULL,
  header = list(Date = Sys.Date()), name = NULL, encoding = "UTF-8",
  quiet = TRUE, control = NULL, texdir = NULL, texengine = "pdflatex")
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character specifying the output directory (default: current working directory). If only a single PDF file is produced and no dir is explicitly specified, the file is displayed on the screen rather than saved in dir.
template	character. A specification of a LaTeX template. The package currently provides "exam", "solution", "plain", among others. The default is to use the "plain.tex" file unless there are Rmd exercises in file for which "plain8.tex" is used. For further details see below.

inputs	character. Names of files that are needed as inputs during LaTeX compilation (e.g., style files, headers). Either the full path must be given or the file needs to be in edir.
header	list. A list of further options to be passed to the LaTeX files.
name	character. A name prefix for resulting exercises, of the same length as template. By default (if name is NULL) the base name of template is used.
control	A list of control arguments for the appearance of multiple choice results (see details).
encoding	character, ignored. The encoding is always assumed to be UTF-8.
quiet	logical. Should output be suppressed when calling <code>xweave</code> and <code>texi2dvi</code> .
transform	function. An optional transform driver passed to <code>xexams</code> (by default no transformation is used).
edir	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
tdir	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
sdir	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
texdir	character specifying a directory for running <code>texi2dvi</code> in. By default this is chosen via <code>tempfile</code> (and deleted again) but, if specified by the user, the temporary LaTeX files from the last iteration are preserved and not deleted. This is intended especially for debugging purposes.
texengine	character. Passed to <code>latexmk</code> if <code>tinytex</code> is available.
verbose	logical. Should information on progress of exam generation be reported?
rds	logical indicating whether the return list should also be saved as an RDS data file.
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within the <code>expoints</code> tags of the exercise files (if any). The vector of points supplied should either have length 1 or the number of exercises in the exam.
seed	integer matrix or logical. Either NULL (default), logical, or a matrix of random seeds for each possible exercise to be set prior to calling <code>driver@sweave</code> . If NULL no random seeds are set. If a matrix, the number of rows must be <code>n</code> and the number of columns must correspond to <code>unlist(file)</code> . If TRUE a suitable matrix of seeds is sampled.
attachfile	logical. Should the LaTeX commands <code>url</code> and <code>href</code> be replaced by <code>attachfile</code> commands when used for supplementary files? This enables embedding these supplementary files directly into the PDF when <code>template</code> loads the <code>attachfile</code> LaTeX package.
exshuffle	logical or integer. If the <code>exshuffle</code> argument is non-NULL it is used to overrule the <code>exshuffle</code> tag from the file (e.g., <code>exshuffle = FALSE</code> can be used to keep all available answers without permutation).
...	further arguments passed on to <code>xweave</code> .

Details

exams2pdf is a more flexible re-implementation of the old (version 1) `exams` function (Gruen and Zeileis 2009), using the new extensible `xexams` framework (Zeileis et al. 2014). A detailed introduction is provided in `vignette("exams", package = "exams")`, also pointing out relative advantages of the new interface.

exams2pdf proceeds by using `make_exams_write_pdf` to set up a custom `driver$write` function on the fly before calling `xexams`. This custom driver combines each exams with the desired template (and inputs etc.) and then calls `texi2dvi` on the resulting LaTeX file to produce PDF output. For a single exam ($n = 1$) the resulting PDF is displayed on screen (unless `dir` is explicitly specified) while for $n > 1$ the PDF files are stored in the output directory `dir`.

The argument `control` is specified by a named list, currently with elements `mchoice.symbol` and `cloze.collapse`. `mchoice.symbol` has to be a character vector with elements `True` and `False`, specifying the symbol used for the questionnaire output in the final PDF file. `cloze.collapse` specifies the character used for collapsing `mchoice/schoice` alternatives within a cloze exercise. By default, these are separated by `" / "` but with `cloze.collapse = "\\ \\ "` each alternative would be in a new line. Finally, `cloze.collapse = "enumerate"` can also be used which employs a nested `enumerate` environment. In the latter case, the questionnaire uses `exclozechoice` rather than `exmchoice` (see `exam.tex` or `solution.tex` for an illustration).

Value

`exams2pdf` returns a list of exams as generated by `xexams`.

`make_exams_write_pdf` returns a function that is suitable for being supplied as `driver$write` to `xexams`.

References

Gruen B, Zeileis A (2009). Automatic Generation of Exams in R. *Journal of Statistical Software*, **29**(10), 1–14. doi:[10.18637/jss.v029.i10](https://doi.org/10.18637/jss.v029.i10).

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:[10.18637/jss.v058.i01](https://doi.org/10.18637/jss.v058.i01).

See Also

[xexams](#), [exams](#), [texi2dvi](#)

Examples

```
## load package and enforce par(ask = FALSE)
##
## additionally, for simplicity, enforce using the basic
## tools::texi2dvi() LaTeX interface instead of the more
## flexible/robust tinytex::latexmk()
library("exams")
oopt <- options(device.ask.default = FALSE, exams_tex = "tools")

if(interactive()) {
```

```
## compile a single random exam (displayed on screen)
exams2pdf(list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  "scatterplot.Rmd",
  "relfreq.Rmd"
))
}

options(exams_tex = oopt$exams_tex)
```

exams2qti12

Generation of Exams in QTI 1.2 Format

Description

Automatic generation of exams in QTI 1.2 format.

Usage

```
exams2qti12(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL,
  tdir = NULL, sdir = NULL, verbose = FALSE, rds = FALSE,
  resolution = 100, width = 4, height = 4, svg = FALSE, encoding = "UTF-8",
  num = NULL, mchoice = NULL,
  schoice = mchoice, string = NULL, cloze = NULL,
  template = "qti12", duration = NULL,
  stitle = "Exercise", ititle = "Question",
  adescription = "Please solve the following exercises.",
  sdescription = "Please answer the following question.",
  maxattempts = 1, cutvalue = 0, solutionswitch = TRUE,
  zip = TRUE, points = NULL,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  converter = NULL, envir = NULL, engine = NULL, xmlcollapse = FALSE,
  flavor = c("plain", "openolat", "canvas", "ilias"), ...)

make_itembody_qti12(rtiming = FALSE, shuffle = FALSE,
  rshuffle = shuffle, minnumber = NULL, maxnumber = NULL,
  defaultval = NULL, minvalue = NULL, maxvalue = NULL,
  cutvalue = NULL, enumerate = FALSE, digits = NULL,
  tolerance = is.null(digits), maxchars = 12,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  fix_num = TRUE, flavor = "plain")
```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from <code>file</code> .
<code>nsamp</code>	integer. The number(s) of exercise files sampled from each list element of <code>file</code> . Sampling without replacement is used if possible. (Only if some element of <code>nsamp</code> is larger than the length of the corresponding element in <code>file</code> , sampling with replacement is used.)
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises and ZIP file.
<code>quiet</code>	logical. Should output be suppressed when calling <code>xweave</code> ?
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in <code>file</code> are stored (see also <code>xexams</code>).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>rds</code>	logical indicating whether the return list should also be saved as an RDS data file.
<code>resolution, width, height</code>	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
<code>svg</code>	logical. Should graphics be rendered in SVG or PNG (default)?
<code>encoding</code>	character, ignored. The encoding is always assumed to be UTF-8.
<code>num</code>	function or named list applied to numerical (i.e., type <code>num</code>) questions. If <code>num</code> is a function, <code>num</code> will be used for generating the item body of the question, see function <code>make_itembody_qti12()</code> . If <code>num</code> is a named list, these arguments will be passed to function <code>make_itembody_qti12()</code> .
<code>mchoice, schoice, string, cloze</code>	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type <code>mchoice</code> , <code>schoice</code> , <code>string</code> , and <code>cloze</code>), respectively. See argument <code>num</code> for more details.
<code>template</code>	character. The QTI 1.2 template that should be used. Currently, the package provides <code>"qti12.xml"</code> .
<code>duration</code>	integer. Set the duration of the exam in minutes.
<code>stitle</code>	character. A title that should be used for the sections. May be a vector of length 1 to use the same title for each section, or a vector containing different section titles.
<code>ititle</code>	character or NULL. A title that should be used for the assessment items. May be a vector of length 1 to use the same title for each item, or a vector containing different item titles. Note that the maximum of different item titles is the number of sections/questions that are used for the exam. Note, if <code>ititle = NULL</code> , simple numbering for the exercises will be used, however, if the exercise contains meta info <code>extitle</code> , this title will be used.

adescription	character. Description (of length 1) for the overall assessment (i.e., exam).
sdescription	character. Vector of descriptions for each section, omitted if empty (or NULL or FALSE).
maxattempts	integer. The maximum attempts for one question. This may also be a vector so that the maximum number of attempts varies across questions. A value of Inf or 0 signals that the attempts per question are not limited.
cutvalue	numeric. The cutvalue at which the exam is passed.
solutionswitch	logical. Should the question/item solutionswitch be enabled? In OLAT this means that the correct solution is shown after an incorrect solution was entered by an examinee (i.e., this is typically only useful if maxattempts = 1).
zip	logical. Should the resulting XML file (plus supplements) be zipped?
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an "\expoints{}" tag in the .Rnw file. The vector of points supplied is expanded to the number of exercises in the exam.
eval	named list, specifies the settings for the evaluation policy, see function exams_eval .
converter	character. Argument passed on to <code>make_exercise_transform_html</code> . The default for <code>converter</code> is set to "ttm" unless there are Rmd exercises in file where "pandoc" is used.
envir	argument passed to xweave (which passes it to knit).
engine	argument passed to xweave indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
xmlcollapse	logical or character. Should line breaks be collapsed in the XML code. If TRUE everything is collapsed with spaces (" ") but other collapse characters could be supplied.
flavor	character. Which QTI 1.2 "flavor" should be used? Some learning management systems require that the QTI XML tags are used in a rather specific and idiosyncratic way. Typically, users should not set this argument directly but use the convenience interfaces exams2openolat , exams2canvas , or exams2ilias instead (which use the <code>flavor</code> argument internally along with further customizations).
rtiming, shuffle, rshuffle, minnumber, maxnumber, defaultval, minvalue, maxvalue	arguments used for QTI 1.2 item construction, for details see the XML specification (see IMS Global Learning Consortium, Inc. 2002), especially Section 4.
enumerate	logical. Insert potential solutions in enumerated list?
digits	integer. How many digits should be used for num exercises?
tolerance	logical. Should tolerance intervals be used for checking if the supplied num answer/number is correct? The default is to use tolerance intervals if digits = NULL.
maxchars	numeric. Lower bound for the number of characters in fill-in-blank fields. The actual number of characters is selected as the maximum number of characters of this value and the actual solution.

<code>fix_num</code>	logical. This is a special flag to enable/force the display of the correct solutions for numeric exercises/answers as well as to obtain results when archiving tests. Note that this is a workaround, which works e.g. within OLAT.
<code>...</code>	further arguments passed on to <code>make_exercise_transform_html</code> .

Details

The Question & Test Interoperability (QTI) is an international XML standard for specifying e-learning tests established by the IMS Global Learning Consortium, Inc. (2002, 2012). The standard evolved over various versions with the first release culminating in the QTI 1.2 standard and the most commonly used stable version of the second release being QTI 2.1. While both versions share many similarities, they also differ in many details. Hence, separate functions `exams2qti12` and `exams2qti21` are provided. Moreover, due to the flexibility of the QTI standard, different learning management systems employ the standard in slightly different ways. Therefore, dedicated interfaces `exams2canvas` and `exams2ilias` (both based on QTI 1.2), and `exams2openolat` (based on either QTI 2.1 or 1.2) are provided for the learning management systems Canvas, ILIAS, and OLAT/OpenOlat, respectively. In addition, the interfaces `exams2blackboard`, `exams2testvision`, and the separate package `exams2sakai` started as forks of the `exams2qti12` and `exams2qti21` functions, respectively.

`exams2qti12` produces a .zip file that may be uploaded into different learning management systems (see above). This includes the final XML file of the exam/assessment as well as possible supplement folders that include images, data sets etc. used for the exam. It proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting Markdown and/or LaTeX code, (3) transforming the Markdown/LaTeX code to HTML, and (4) embedding the HTML code in a XML file using the QTI 1.2 standards for assessments and question items.

For steps (1) and (2) the standard drivers in `xexams` are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in `exams2html`. For step (4), the function will cycle through all questions and exams to generate the final XML file in QTI 1.2 format. Therefore, each question will be included in the XML as one "section". The replicates of each question will be written as question items of the section. The default XML template generates exams/assessments that sample one replicate of a question/item for each section. Typically, the learning management systems sample the items from the different sections independently so that one participant might receive the first random replication from the first exercise but the third random replication from the second exercise, and so on.

Templates other than the default `qti12.xml` provided in the `xml` folder of this package could in principle also be specified. However, adapting these is a bit technical. The assessment template must provide one section including one item. `exams2qti12` will then use the single item template to generate all items, as well as the assessment and section specifications set within the template. Note that all specifiers that have a leading `##` in the XML template will be replaced by suitable code in `exams2qti12` and should always be provided in the template. I.e., the user may add additional tags to the XML template or modify certain specifications, like the number of replicates/items that should be sampled for each section etc.

By default, the individual question/item bodies are generated by function `make_itembody_qti12`, which checks the type of the question and will produce suitable XML code. Note that for each question type, either the arguments of `make_itembody_qti12` may be set within `num`, `mchoice`, `schoice`, `string` and `cloze` in `exams2qti12`, by providing a named list of specifications that should be used, or for each questiontype, a function that produces the item body XML code may be

provided to `num`, `mchoice`, `schoice`, `string` and `cloze`. E.g., `mchoice = list(shuffle = TRUE)` will force only multiple choice questions to have a shuffled answerlist.

Note that in very old OLAT/OpenOlat versions with QTI 1.2, `num` exercises are not officially supported but in fact work correctly. The only drawback is that in certain settings the correct solution is not shown at the end of the assessment (although it is used for all internal computations). Therefore, two workarounds are implemented. Either `fix_num` can be set to `TRUE` (default), then a fix is added by double-checking the result, or `digits` can be set to a fixed value (e.g., `digits = 2`). In the latter case, the `num` exercise is represented by a `string`. Then the answer must be provided exactly to the decimal places specified (e.g., if the exact solution is 16.4562, then the correct answer in the test will be "16.46", i.e., a character string of 5 characters).

Value

`exams2qti12` returns a list of exams as generated by `xexams`.

`make_itembody_qti12` returns a function that generates the XML code for the `itembody` tag in QTI 1.2 format.

References

IMS Global Learning Consortium, Inc. (2002). *IMS Question & Test Interoperability: ASI XML Binding Specification Final Specification Version 1.2*. https://www.imsglobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html

IMS Global Learning Consortium, Inc. (2012). *IMS Question & Test Interoperability (QTI) XSD Binding Version 2.1 Final*. https://www.imsglobal.org/question/qtiv2p1/imsqti_bindv2p1.html

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

`xexams`, `ttm`, `tth`, `tex2image`, `make_exercise_transform_html`,

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  c("scatterplot.Rmd", "boxhist.Rmd"),
  "relfreq.Rmd"
)

## output directory
```

```

dir.create(mydir <- tempfile())

## generate .zip with QTI 1.2 exam in temporary directory
## using a few customization options
exams2qti12(myexam, n = 3, dir = mydir,
  maxattempts = 3,
  num = list(digits = 1),
  mchoice = list(shuffle = TRUE, enumerate = FALSE)
)
dir(mydir)

```

exams2qti21

Generation of Exams in QTI 2.1 Format

Description

Automatic generation of exams in QTI 2.1 format.

Usage

```

exams2qti21(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL,
  tdir = NULL, sdir = NULL, verbose = FALSE, rds = FALSE,
  resolution = 100, width = 4, height = 4, svg = FALSE, encoding = "UTF-8",
  num = NULL, mchoice = NULL,
  schoice = mchoice, string = NULL, cloze = NULL,
  template = "qti21", duration = NULL,
  stitle = NULL, ititle = NULL,
  adescription = "Please solve the following exercises.", sdescription = "",
  maxattempts = 1, cutvalue = NULL, solutionswitch = TRUE,
  casesensitive = TRUE, cloze_schoice_display = "auto",
  navigation = "nonlinear", allowskipping = TRUE, allowreview = FALSE,
  allowcomment = FALSE, shufflesections = FALSE, zip = TRUE, points = NULL,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  converter = NULL, envir = NULL, engine = NULL, base64 = TRUE, mode = "hex",
  include = NULL, selection = c("pool", "exam"), flavor = "plain", ...)

make_itembody_qti21(shuffle = FALSE, defaultval = NULL,
  minvalue = NULL, maxvalue = NULL, enumerate = FALSE,
  digits = NULL, tolerance = is.null(digits), maxchars = 12,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE),
  solutionswitch = TRUE, casesensitive = TRUE,
  cloze_schoice_display = c("auto", "buttons", "dropdown"),
  copypaste = TRUE)

```

Arguments

<code>file</code>	character. A specification of a (list of) exercise files.
<code>n</code>	integer. The number of copies to be compiled from <code>file</code> .
<code>nsamp</code>	integer. The number(s) of exercise files sampled from each list element of <code>file</code> . Sampling without replacement is used if possible. (Only if some element of <code>nsamp</code> is larger than the length of the corresponding element in <code>file</code> , sampling with replacement is used.)
<code>dir</code>	character. The default is the current working directory.
<code>name</code>	character. A name prefix for resulting exercises and ZIP file (must not contain spaces or periods, otherwise replaced by underscores).
<code>quiet</code>	logical. Should output be suppressed when calling <code>xweave</code> ?
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in <code>file</code> are stored (see also <code>xexams</code>).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>rds</code>	logical indicating whether the return list should also be saved as an RDS data file.
<code>resolution, width, height</code>	numeric. Options for rendering PNG (or SVG) graphics passed to <code>xweave</code> .
<code>svg</code>	logical. Should graphics be rendered in SVG or PNG (default)?
<code>encoding</code>	character, ignored. The encoding is always assumed to be UTF-8.
<code>num</code>	function or named list applied to numerical (i.e., type <code>num</code>) questions. If <code>num</code> is a function, <code>num</code> will be used for generating the item body of the question, see function <code>make_itembody_qti21()</code> . If <code>num</code> is a named list, these arguments will be passed to function <code>make_itembody_qti21()</code> .
<code>mchoice, schoice, string, cloze</code>	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type <code>mchoice</code> , <code>schoice</code> , <code>string</code> , and <code>cloze</code>), respectively. See argument <code>num</code> for more details.
<code>template</code>	character. The QTI 2.1 template that should be used. Currently, the package provides <code>"qti21.xml"</code> .
<code>duration</code>	integer. Set the duration of the exam in minutes.
<code>stitle</code>	character. A title that should be used for the sections. May be a vector of length 1 to use the same title for each section, or a vector containing different section titles.
<code>ititle</code>	character or NULL. A title that should be used for the assessment items. May be a vector of length 1 to use the same title for each item, or a vector containing different item titles. Note that the maximum of different item titles is the number

	of sections/questions that are used for the exam. Note, if <code>ititle = NULL</code> , simple numbering for the exercises will be used, however, if the exercise contains meta info <code>extitle</code> , this title will be used.
<code>adescription</code>	character. Description (of length 1) for the overall assessment (i.e., exam).
<code>sdescription</code>	character. Vector of descriptions for each section, omitted if empty (or <code>NULL</code> or <code>FALSE</code>).
<code>maxattempts</code>	integer. The maximum attempts for one question. This may also be a vector so that the maximum number of attempts varies across questions. A value of <code>Inf</code> or <code>0</code> signals that the attempts per question are not limited.
<code>cutvalue</code>	numeric. The number of points at which the exam is passed. If set to <code>NULL</code> (default) or equivalently <code>NA</code> , no <code>cutvalue</code> is set and the result of the exam is just the number of points.
<code>solutionswitch</code>	logical or character. Should the question/item <code>solutionswitch</code> be enabled? In OLAT this means that the correct solution is shown after an incorrect solution was entered by an examinee (i.e., this is typically only useful if <code>maxattempts = 1</code>). Furthermore, if e.g. <code>solutionswitch = c("correct", "incorrect", "summary")</code> , the correct solution can be either displayed when the answer is "correct" and/or "incorrect" and/or after the test is completed in the "summary" of the results.
<code>casesensitive</code>	logical. Should the evaluation of string exercises be case sensitive?
<code>cloze_schoice_display</code>	character. For <code>schoice</code> answers in <code>cloze</code> exercises, select the display of the possible answers. By default (<code>cloze_schoice_display = "auto"</code>), radio "buttons" are used if the answer list appears in its own paragraph and a "dropdown" menu is used if the answer list appears inline (and has no mathematical markup). Both options can also be enforced explicitly, independently from the answer list appearing in a separate paragraph or inline. Note that in dropdown menus the answer list will typically be rendered as plain text which means that mathematical notation may become unintelligible.
<code>copypaste</code>	logical. For string/text responses, controls whether text may be copied into the text entry field, or must actually be entered.
<code>navigation</code>	character. Mode of navigation, can either be "nonlinear" (default) or "linear". The former means that test participants can switch back and forth between questions while the latter implies that the questions need to be answered sequentially.
<code>allowskipping</code>	logical. Can a question/section be skipped (default) or must it be answered?
<code>allowreview</code>	logical. Can questions be viewed again at the end of a test/exam or not (default)?
<code>allowcomment</code>	logical. Are comments allowed?
<code>shufflesections</code>	logical. Should the order of the exercises be shuffled? For <code>selection = "pool"</code> setting <code>shufflesections = TRUE</code> this corresponds to shuffling the sections that contain the pools of exercises. For <code>selection = "exam"</code> it corresponds to shuffling the exercises within each exam section.
<code>zip</code>	logical. Should the resulting XML file (plus supplements) be zipped?

points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an <code>"\expoints{}</code> " tag in the <code>.Rnw</code> file. The vector of points supplied is expanded to the number of exercises in the exam.
eval	named list, specifies the settings for the evaluation policy, see function <code>exams_eval</code> .
converter	character. Argument passed on to <code>make_exercise_transform_html</code> . The default for <code>converter</code> is set to <code>"ttm"</code> unless there are Rmd exercises in file where <code>"pandoc"</code> is used.
envir	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
engine	argument passed to <code>xweave</code> indicating whether <code>"Sweave"</code> (default) or <code>"knitr"</code> should be used for rendering Rnw exercises.
base64	logical. Should supplementary files be embedded using Base 64 coding? Argument <code>base64</code> may also be a character vector of file suffixes that should be encoded, e.g. <code>base64 = c("png", "rda")</code> will only encode PNG images and binary <code>.rda</code> files. If set to <code>NULL</code> only image files will be encoded.
mode	character. See function <code>tth</code> .
include	character, paths of extra files that should be included within the final <code>.zip</code> -file.
selection	character. If <code>selection = "pool"</code> , the function creates one section for each exercise from which one replication will be selected in the exam. If <code>selection = "exam"</code> each section contains all questions and one section will be selected for the exam. The <code>"exam"</code> variant has the advantage that questions that build on each other can be used in the exam.
flavor	character. Which QTI 2.1 "flavor" should be used? Some learning management systems require that the QTI XML tags are used in a rather specific and idiosyncratic way. Typically, users should not set this argument directly but use the convenience interfaces such as <code>exams2openolat</code> instead (which use the <code>flavor</code> argument internally along with further customizations).
shuffle, default	arguments used for item construction, for details see the XML specification (see IMS Global Learning Consortium, Inc. 2012), especially Section 4.
tval, minvalue, maxvalue	
enumerate	logical. Insert potential solutions in enumerated list?
digits	integer. How many digits should be used for num exercises?
tolerance	logical. Should tolerance intervals be used for checking if the supplied num answer/number is correct? The default is to use tolerance intervals if <code>digits = NULL</code> .
maxchars	numeric. Lower bound for the number of characters in fill-in-blank fields. The actual number of characters is selected as the maximum number of characters of this value and the actual solution.
...	further arguments passed on to <code>make_exercise_transform_html</code> .

Details

The Question & Test Interoperability (QTI) is an international XML standard for specifying e-learning tests established by the IMS Global Learning Consortium, Inc. (2002, 2012). The standard

evolved over various versions with the first release culminating in the QTI 1.2 standard and the most commonly used stable version of the second release being QTI 2.1. While both versions share many similarities, they also differ in many details. Hence, separate functions `exams2qti12` and `exams2qti21` are provided. Moreover, due to the flexibility of the QTI standard, different learning management systems employ the standard in slightly different ways. Therefore, dedicated interfaces `exams2canvas` and `exams2ilias` (both based on QTI 1.2), and `exams2openolat` (based on either QTI 2.1 or 1.2) are provided for the learning management systems Canvas, ILIAS, and OLAT/OpenOlat, respectively. In addition, the interfaces `exams2blackboard`, `exams2testvision`, and the separate package `exams2sakai` started as forks of the `exams2qti12` and `exams2qti21` functions, respectively.

`exams2qti21` produces a .zip file that may be uploaded into different learning management systems (see above). This includes the final XML file of the exam/assessment, its exercises as well as possible supplement folders that include images, data sets etc. It proceeds by (1) calling `xweave` for each exercise, (2) reading the resulting Markdown and/or LaTeX code using `read_exercise`, (3) transforming the Markdown/LaTeX code to HTML, and (4) embedding the HTML code in XML files using the QTI 2.1 standard for assessments and question items.

For steps (1) and (2) the standard drivers in `xexams` are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in `exams2html`. For step (4), the function will cycle through all questions and exams to generate the final XML files in QTI 2.1 format: separate XML files for each random replication of each exercise (labeled items in sections in QTI), one XML file tying the overall exam/assessment together based on the individual exercises and one overall `'imsmanifest.xml'`.

For arranging the individual exercise replications (called "items") in so-called "sections" two different types of selection are available: First, a "pool" of replications/items can be put into a section and then one item will be selected randomly for each participant from each section. As learning management systems typically sample the sections independently, this means that one participant might receive the first random replication from the first exercise but the third random replication from the second exercise, and so on. Alternatively, `selection = "exam"` specifies that each section contains a complete set of exercises and one section will be selected randomly for each participant. The "exam" variant is less commonly used but has the advantage that questions that build on each other can be used in the exam. Also, duplicated exercise types can be avoided when using `nsamp` to sample a certain number of exercises without replacement.

Templates other than the default `qti21.xml` provided in the `xml` folder of this package could in principle also be specified. However, adapting these is a bit technical. The assessment template must provide one section including one item. `exams2qti21` will then use the single item template to generate all items, as well as the assessment and section specifications set within the template. Note that all specifiers that have a leading `##` in the XML template will be replaced by suitable code in `exams2qti21` and should always be provided in the template. Thus, the user may add additional tags to the XML template or modify certain specifications, like the number of replicates/items that should be sampled for each section etc.

By default, the individual question/item bodies are generated by function `make_itembody_qti21` which checks the type of the question and will produce suitable XML code. Note that for each question type, either the arguments of `make_itembody_qti21` may be set within `num`, `mchoice`, `schoice`, `string` and `cloze` in `exams2qti21`, by providing a named list of specifications that should be used, or for each question type, a function that produces the item body XML code may be provided to `num`, `mchoice`, `schoice`, `string` and `cloze`. E.g., `mchoice = list(shuffle = TRUE)` will force only multiple choice questions to have a shuffled answerlist.

Value

exams2qti21 returns a list of exams as generated by [xexams](#).

make_itembody_qti21 returns a function that generates the XML code for the itembody tag in QTI 2.1 format.

References

IMS Global Learning Consortium, Inc. (2002). *IMS Question & Test Interoperability: ASI XML Binding Specification Final Specification Version 1.2*. https://www.imslobal.org/question/qtiv1p2/imsqti_asi_bindv1p2.html

IMS Global Learning Consortium, Inc. (2012). *IMS Question & Test Interoperability (QTI) XSD Binding Version 2.1 Final*. https://www.imslobal.org/question/qtiv2p1/imsqti_bindv2p1.html

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. [doi:10.18637/jss.v058.i01](https://doi.org/10.18637/jss.v058.i01).

See Also

[xexams](#), [ttm](#), [tth](#), [tex2image](#), [make_exercise_transform_html](#),

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  c("scatterplot.Rmd", "boxhist.Rmd"),
  "relfreq.Rmd"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with QTI 2.1 exam in temporary directory
## using a few customization options
exams2qti21(myexam, n = 3, dir = mydir,
  maxattempts = 3,
  num = list(digits = 1),
  mchoice = list(shuffle = TRUE, enumerate = TRUE)
)
dir(mydir)
```

Description

Interface for generating exams in TCEXAM format.

Usage

```
exams2tcexam(file, n = 1L, nsamp = NULL, dir = ".",
             name = NULL, quiet = TRUE, edir = NULL, tdir = NULL, sdir = NULL, verbose = FALSE,
             resolution = 100, width = 4, height = 4, svg = FALSE, encoding = "UTF-8",
             points = NULL, modulename = name, subjectname = name, subjectdescription = NULL,
             timer = 0, fullscreen = FALSE, inlineanswers = FALSE, autonext = FALSE,
             shuffle = FALSE, lang = "en", date = Sys.time(), zip = FALSE, converter = NULL,
             ...)
```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp, quiet, edir, tdir, sdir, verbose	arguments passed to xexams .
dir	character specifying the output directory path. The default is the current working directory.
name	character. A name prefix for resulting XML file.
resolution, width, height, svg	arguments passed to xweave .
encoding	character, ignored. The encoding is always assumed to be UTF-8.
points	numeric. Number of points for the questions.
modulename	character. Module name.
subjectname	character. Subject name.
subjectdescription	character. Subject description.
timer	numeric. Number of seconds for each question.
fullscreen	logical. Should the question be shown in full-screen mode?
inlineanswers	logical. Should the question list be presented inline?
autonext	logical. Automatically advance to the next item?
shuffle	logical. Should the question list of schoice/mchoice answers be shuffled (or kept fixed)?
lang	character. Two-letter indicator of the language.
date	character or "Date" object specifying the date of the exam.

zip logical. Should the resulting XML file be zipped?

converter, ... arguments passed on to `make_exercise_transform_html`. The default for `converter` is set to "ttm" unless there are Rmd exercises in file where "pandoc" is used.

Details

`exams2tcexam` generates XML exams that can be imported into the TCEXAM software of Asuni (2012). Currently, the subset of HTML(-like) commands that is supported in TCEXAM is rather limited, e.g., tables and figures cannot be directly included.

Value

A list of exams as generated by `xexams` is returned invisibly.

References

Asuni (2012). *TCEXAM: Computer-Based Assessment Software*. <https://tcexam.org/>.

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## Not run:
## exams2tcexam creates a single XML file
exams2tcexam("tstat2", n = 2)

## End(Not run)
```

exams2testvision

Generation of Exams in TestVision Format

Description

Automatic generation of exams in TestVision format (still under development) for the online testing system TestVision Online.

Usage

```
exams2testvision(file, n = 1L, nsamp = NULL, dir = ".",
  name = NULL, quiet = TRUE, edir = NULL,
  tdir = NULL, sdir = NULL, verbose = FALSE,
  resolution = 100, width = 4, height = 4, svg = FALSE,
  encoding = "UTF-8", envir = NULL, engine = NULL,
  num = NULL, mchoice = NULL,
  schoice = mchoice, string = NULL, cloze = NULL,
```

```

template = "testvision",
stitle = "Exercise", ititle = "Question",
adescription = "Please solve the following exercises.",
sdescription = "Please answer the following question.",
maxattempts = 1, solutionswitch = TRUE, zip = TRUE, points = NULL,
eval = list(partial = TRUE, rule = "false2", negative = FALSE), converter = "pandoc",
base64 = FALSE, mode = "hex", ...)

make_itembody_testvision(shuffle = FALSE, defaultval = NULL,
  minvalue = NULL, maxvalue = NULL,
  enumerate = FALSE, digits = NULL,
  tolerance = is.null(digits), maxchars = 12,
  eval = list(partial = TRUE, rule = "false2", negative = FALSE), solutionswitch = TRUE)

```

Arguments

file	character. A specification of a (list of) exercise files.
n	integer. The number of copies to be compiled from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
dir	character. The default is the current working directory.
name	character. A name prefix for resulting exercises and ZIP file.
quiet	logical. Should output be suppressed when calling <code>xweave</code> ?
edir	character specifying the path of the directory (along with its sub-directories) in which the files in file are stored (see also <code>xexams</code>).
tdir	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
sdir	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
verbose	logical. Should information on progress of exam generation be reported?
resolution, width, height	numeric. Options for rendering PNG graphics passed to <code>xweave</code> .
svg	logical. Should graphics be rendered in SVG or PNG (default)?
encoding	character, ignored. The encoding is always assumed to be UTF-8.
envir	argument passed to <code>xweave</code> (which passes it to <code>knit</code>).
engine	argument passed to <code>xweave</code> indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
num	function or named list applied to numerical (i.e., type num) questions. If num is a function, num will be used for generating the item body of the question, see function <code>make_itembody_testvision()</code> . If num is a named list, these arguments will be passed to function <code>make_itembody_testvision()</code> .

mchoice, schoice, string, cloze	function or named list applied to multiple choice, single choice, string, and cloze questions (i.e., type mchoice, schoice, string, and cloze), respectively. See argument num for more details.
template	character. The IMS QTI 1.2 or 2.1 template that should be used. Currently, the package provides "testvision.xml".
stitle	character. A title that should be used for the sections. May be a vector of length 1 to use the same title for each section, or a vector containing different section titles.
ititle	character. A title that should be used for the assessment items. May be a vector of length 1 to use the same title for each item, or a vector containing different item titles. Note that the maximum of different item titles is the number of sections/questions that are used for the exam.
adescription	character. Description (of length 1) for the overall assessment (i.e., exam).
sdescription	character. Vector of descriptions for each section, omitted if empty (or NULL or FALSE).
maxattempts	integer. The maximum attempts for one question, may also be set to Inf.
solutionswitch	logical. Should the question/item solutionswitch be enabled?
zip	logical. Should the resulting XML file (plus supplements) be zipped?
points	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within an "\expoints{" tag in the .Rnw file. The vector of points supplied is expanded to the number of exercises in the exam.
eval	named list, specifies the settings for the evaluation policy, see function exams_eval .
base64	logical. Should supplementary files be embedded using Base 64 coding? Argument base64 may also be a character vector of file suffixes that should be encoded, e.g. base64 = c("png", "rda") will only encode PNG images and binary .rda files. If set to NULL only image files will be encoded.
converter	character. Argument passed on to make_exercise_transform_html. The default for converter is set to "pandoc" unless "ttm" is required. The default works best in TestVision.
mode	character. See function tth .
shuffle, defaultval, minvalue, maxvalue	arguments used for IMS QTI 2.1 item construction, for details see the XML specification (see IMS Global Learning Consortium, Inc. 2012), especially Section 4.
enumerate	logical. Insert potential solutions in enumerated list?
digits	integer. How many digits should be used for num exercises?
tolerance	logical. Should tolerance intervals be used for checking if the supplied num answer/number is correct? The default is to use tolerance intervals if digits = NULL.
maxchars	numeric. Lower bound for the number of characters in fill-in-blank fields. The actual number of characters is selected as the maximum number of characters of this value and the actual solution.
...	further arguments passed on to make_exercise_transform_html.

Details

TestVision employs an XML format that essentially uses the Question & Test Interoperability (QTI) standard, version 2.1, see IMS Global Learning Consortium, Inc. (2012). However, as this deviates substantially from the plain QTI 2.1 standard in several places, the `exams2qti21` cannot be used directly. Instead, `exams2testvision` is a new interface that allows for meeting TestVision's requirements for XML-imports.

`exams2testvision` produces a `.zip` file that may be uploaded into TestVision. This includes the final XML file of the exam/assessment as well as possible supplement folders that include images, data sets etc. used for the exam.

`exams2testvision` proceeds by (1) calling `xweave` on each exercise, (2) reading the resulting LaTeX code, (3) transforming the LaTeX code to HTML, and (4) embedding the HTML code in a XML file using TestVision's QTI standards for assessments and question items. For steps (1) and (2) the standard drivers in `xexams` are used. In step (3), a suitable transformation function is set up on the fly using `make_exercise_transform_html`, see also the details section in `exams2html`. For step (4), the function will cycle through all questions and exams to generate the final XML file in the TestVision QTI standard. The questions appear as separate files in the system.

The function uses the XML template for TestVision's QTI standards for assessments and items to generate the exam (per default, this is the XML file `testvision.xml` provided in the `xml` folder of this package). The assessment template must provide one or more sections for including the items. `exams2testvision` will then use the single item template to generate all items, as well as the assessment and section specifications set within the template.

The default template will generate exams/assessments that sample one replicate of a question/item for each section. The usual procedure in exam/assessment generation would be to simply copy & paste the XML template of the package and adapt it to the needs of the user. Note that all specifiers that have a leading `##` in the XML template will be replaced by suitable code in `exams2testvision` and should always be provided in the template. I.e., the user may add additional tags to the XML template or modify certain specifications, like the number of replicates/items that should be sampled for each section etc.

Per default, the individual question/item bodies are generated by function `make_itembody_testvision`, i.e., `make_itembody_testvision` checks the type of the question and will produce suitable XML code. Note that for each question type, either the arguments of `make_itembody_testvision` may be set within `num`, `mchoice`, `schoice` and `string` in `exams2testvision`, by providing a named list of specifications that should be used, or for each questiontype, a function that produces the item body XML code may be provided to `num`, `mchoice`, `schoice` and `string`. E.g., `mchoice = list(shuffle = TRUE)` will force only multiple choice questions to have a shuffled answerlist.

Please note the following for `cloze` questions: Although any combination of the four item types can be successfully uploaded into TestVision, `cloze` questions containing `mchoice` do not work properly (it is therefore advised to not use `mchoice` within `cloze`).

Value

`exams2testvision` returns a list of exams as generated by `xexams`.

`make_itembody_testvision` returns a function that generates the XML code for the `itembody` tag in TestVision's version of the IMS QTI 2.1 format.

References

- TestVision (2020). *English introduction tutorial on TestVision*. <https://testvision.nl/en/experience-testvision/tutorials-2/>
- IMS Global Learning Consortium, Inc. (2012). *IMS Question & Test Interoperability (QTI) XSD Binding Version 2.1 Final*. https://www.imsglobal.org/question/qtiv2p1/imsqti_bindv2p1.html
- Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

[exams2qti12](#)

Examples

```
## load package and enforce par(ask = FALSE)
library("exams")
options(device.ask.default = FALSE)

## define an exams (= list of exercises)
myexam <- list(
  "tstat",
  "tstat2",
  "relfreq",
  "essayreg",
  "dist2",
  "boxhist2"
)

## output directory
dir.create(mydir <- tempfile())

## generate .zip with set of TestVision exercises
exams2testvision(myexam, n = 3, dir = mydir)
dir(mydir)
```

exams_eval

Auxiliary Tools for Evaluating Exams

Description

Generation of various helper functions for evaluating exams.

Usage

```
exams_eval(partial = TRUE, negative = FALSE,
  rule = c("false2", "false", "true", "all", "none"))
```

Arguments

<code>partial</code>	logical. Should multiple-choice (<code>mchoice</code>) answers be evaluated as a whole pattern (<code>partial = FALSE</code>) or should partial credits be assigned to each of the choices (<code>partial = TRUE</code>)?
<code>negative</code>	logical or numeric. Handling of negative points for an exercise, for details see below.
<code>rule</code>	character specifying which rule to use for negative partial credits (i.e., only relevant for multiple-choice answers when <code>partial = TRUE</code>).

Details

The function `exams_eval` is a convenience wrapper for specifying various types of evaluation policies. It returns a set of auxiliary functions that may be useful in the evaluation of exams.

Exercises of types `num`, `string`, or `schoice` can essentially be just correct or wrong. In the former case they will give 100 percent of all points, in the latter either 0 percent or some negative percentage can be assigned. Setting `negative = TRUE` is equivalent to setting either `negative = 1` or equivalently `negative = -1`, which all signal that 100 percent of the points for the exercise should be subtracted. Other percentages are also possible, e.g., `negative = 0.25`, which would be a natural choice for "schoice" questions with five answer alternatives. Not that when using an evaluation strategy with negative points for wrong answers, the system that collects the participants' answers should distinguish between 'solved incorrectly' and 'not attempted' (which should always yield 0 percent).

Moreover, for `mchoice` (multiple-choice) answers the evaluation policy can either pertain to the answer pattern as a whole (which can be correct or wrong, see above) or it can employ a partial credit strategy. In the latter case, each selected correct choice will yield the fraction $1/n_{\text{correct}}$ of points. When an incorrect choice is selected, it should lead to negative points. Five strategies are currently implemented: "false" uses $1/n_{\text{wrong}}$ while "false2" uses $1/\max(n_{\text{wrong}}, 2)$; "true" uses $1/n_{\text{correct}}$ (so that each wrong selection cancels one correct selection); "all" uses 1 (so that a single wrong selection cancels all correct selections); and "none" uses 0 (so that wrong selections have no effect at all). When aggregating the partial percentages, the overall points can become negative. By setting `negative` a lower bound can be set: `negative = TRUE` sets no bound while `negative = FALSE` sets the bound to zero. Any other numeric value could be set as well, e.g., `negative = 0.25`.

The functions returned by `exams_eval` try to automatically infer the type of exercise based on the correct answer. However, this cannot always infer the type reliably (e.g., the number 10 vs. the string 10 vs. a multiple-choice question with two elements, true and false). Specifically, multiple-choice vs. single-choice cannot be distinguished automatically. Hence, it is better to explicitly indicate the exercise type with the `type` argument.

Evaluations for `cloze` exercises have to be built by appropriately reusing the building blocks for `num/string/schoice/mchoice`. For example, the components of `cloze` exercises have to be evaluated individually and then aggregated as desired. Different evaluations for different item types may be set as in: `exams2qti12(..., eval = eval1, schoice = list(eval = eval2))`. Then `eval = eval1` is used as the default for all exercise types except `schoice` where `eval = eval2` is used.

Thus, `exams_eval` might not give the complete finished evaluation policy for an entire exam but supplies the most important building blocks for setting this up 'by hand'. Internally, `exams_eval`

is also used by [exams2moodle](#), [exams2qti12](#) and [exams2blackboard](#) for writing the evaluation specifications in the respective XML specifications.

Value

exams_eval returns a list with the input parameters partial, negative, and rule along with the following functions:

checkanswer	function with arguments (correct, answer, tolerance = 0, and type = NULL. It checks whether answer (sufficiently) matches correct or not. It returns 1 for correct, -1 for wrong and 0 for not attempted. In case of partial = TRUE, the functions returns a vector for multiple-choice questions.
pointvec	function with arguments correct = NULL and type = NULL. It computes the vector of points for correct and wrong answers, respectively.
pointsum	function with arguments (correct, answer, tolerance = 0, and type = NULL. It computes the overall number of points.

All of the functions require at least the correct answer and optionally the exercise type (num, mchoice/schoice, or string). By default, the type is inferred from correct which works automatically except in a few edge cases (e.g., to correctly autodetect a string exercise the correct answer must contain at least one character that is not 0 or 1).

See Also

[exams2moodle](#), [exams2qti12](#), [exams2blackboard](#)

Examples

```
## binary evaluation policy with solutions being either correct
## or wrong: partial = FALSE, negative = FALSE
ee <- exams_eval(partial = FALSE, negative = FALSE)

## points that can be achieved are 0/1
ee$pointvec()

## checkanswer() returns 1 for correct, -1 for incorrect and 0 for missing answer
ee$checkanswer(1.23, 1.23)
ee$checkanswer(1.23, "1.23")
ee$checkanswer(1.23, "1,23")
ee$checkanswer(1.23, 1.24)
ee$checkanswer(1.23, 1.24, tolerance = 0.01)
ee$checkanswer(1.23, NA)
ee$checkanswer(1.23, NULL)
ee$checkanswer(1.23, "")

## similarly for logical (mchoice/schoice) answers
## (which allows either string or logical specification)
ee$checkanswer("10000", "10000")
ee$checkanswer(c(TRUE, FALSE, FALSE, FALSE, FALSE), c(TRUE, FALSE, FALSE, FALSE, FALSE))
ee$checkanswer(c(TRUE, FALSE, FALSE, FALSE, FALSE), "10000")
ee$checkanswer("10000", "01000")
```

```

ee$checkanswer("10000", "11000")

## and analogously for strings
ee$checkanswer("foo", "foo")
ee$checkanswer("foo", "bar")
ee$checkanswer("foo", "")

## obtain points achieved
ee$pointsum("10000", "10000")
ee$pointsum("10000", "01000")
ee$pointsum("10000", "00000")
ee$pointsum("10000", NA)

## -----
## evaluation policy with -25% penalty for wrong answers
ee <- exams_eval(partial = FALSE, negative = -0.25)

## points that can be achieved are 1/-0.25 (or zero)
ee$pointvec()

## obtain points achieved
ee$pointsum("10000", "10000")
ee$pointsum("10000", "01000")
ee$pointsum("10000", "00000")
ee$pointsum("10000", NA)
ee$pointsum(1.23, 1.23)
ee$pointsum(1.23, 2.34)
ee$pointsum(1.23, NA)
ee$pointsum(1.23, 1.24)
ee$pointsum(1.23, 1.24, tolerance = 0.1)

## -----
## default evaluation policy with partial points
## (but without negative points overall)
ee <- exams_eval()

## points that can be achieved are 1/3 (1/#true)
## or -1/2 (1/#false)
ee$pointvec("10101")

## obtain points achieved
ee$pointsum("10101", "10101")
ee$pointsum("10101", "10100")
ee$pointsum("10101", "11100")
ee$pointsum("10101", "01010")
ee$pointsum("10101", "00000")

## show individual answer check
ee$checkanswer("10101", "10101")
ee$checkanswer("10101", "10100")
ee$checkanswer("10101", "11100")
ee$checkanswer("10101", "01010")
ee$checkanswer("10101", "00000")

```

```

## numeric/string answers are not affected by partial=TRUE
ee$checkanswer(1.23, 1.23)
ee$pointsum(1.23, 1.23)
ee$checkanswer(1.23, 2.34)
ee$pointsum(1.23, 2.34)

## -----
## evaluation policy with partial points
## (and with up to -25% negative points overall)
ee <- exams_eval(partial = TRUE, negative = -0.25)

## points that can be achieved are 1/3 (1/#true)
## or -1/2 (1/#false)
ee$pointvec("10101")

## obtain points achieved
ee$pointsum("10101", "10101")
ee$pointsum("10101", "01010")
ee$pointsum("10101", "00000")

## show individual answer check
ee$checkanswer("10101", "10101")
ee$checkanswer("10101", "10100")
ee$checkanswer("10101", "11100")
ee$checkanswer("10101", "01010")
ee$checkanswer("10101", "00000")

## numeric/string answers are not affected by partial=TRUE
ee$pointsum(1.23, 1.23)
ee$pointsum(1.23, 2.34)

```

exams_skeleton

Generate Skeleton for Exams Directory/Script

Description

Generate a directory structure which contains ‘demo-*.R’ scripts along with directories containing all available demonstration exercise ‘.Rnw’ or ‘.Rmd’ files and necessary template files (LaTeX, HTML, or XML).

Usage

```

exams_skeleton(dir = ".",
  type = c("num", "schoice", "mchoice", "string", "cloze"),
  writer = c("exams2html", "exams2pdf", "exams2moodle",
    "exams2qti12", "exams2qti21", "exams2arsnova", "exams2nops"),
  markup = "markdown", absolute = FALSE, encoding = "UTF-8")

```

Arguments

dir	character with path to directory. The default is the current working directory.
type	character vector indicating types of exercises that should be included in the ‘demo.R’ script. By default an example for each type of exercise is included.
writer	character vector indicating the exams2xyz writer functions that should be included in the ‘demo.R’ script. By default an example for each type of writer is included.
markup	character vector indicating whether the example exercises use "latex" markup (.Rnw files) or "markdown" markup (.Rmd files).
absolute	logical. Should the paths in the ‘demo.R’ script be absolute? The default is to use relative paths.
encoding	character, ignored. The encoding is always assumed to be UTF-8.

Details

exams_skeleton (or equivalently exams.skeleton) creates a directory with several ‘demo-*.R’ scripts illustrating the use of the various exams2xyz interfaces. Subdirectories with copies of all demonstration exercise .Rnw or .Rmd files and templates for different output formats (LaTeX, HTML, or XML) are also created.

This should provide a starting point for users wishing to start their own collection of exercises with **exams**.

Value

exams_skeleton returns a list of character vectors with the demo scripts invisibly.

See Also

[exams2html](#), [exams2pdf](#), [exams2moodle](#), [exams2qti12](#), [exams2qti21](#), [exams2arsnova](#), [exams2nops](#)

Examples

```
## output directory (replace this with mydir <- "/path/to/your/directory")
dir.create(mydir <- tempfile())

## create exams skeleton with absolute paths in demo.R
exams_skeleton(dir = mydir, absolute = TRUE)

## look at created files
dir(mydir)
dir(mydir, recursive = TRUE)

## now open demo-all.R or any of the other demo-*.R scripts in your
## favorite R code editor and run the examples...
```

Description

Set parameters, defined in the first code chunk of an exercise file, to specific values instead of their definition in the exercise file.

Usage

```
expar(file, ...)
```

Arguments

<code>file</code>	character with (path to) an exercise file.
<code>...</code>	parameters to be fixed within <code>file</code> (or a single list of parameters to be fixed).

Details

To set certain parameters that are randomly generated within an exercise file to specific values, a copy of the exercise file is generated in the temporary directory of the R session. In the temporary copy of the exercise file the first assignment to the specified parameter in the first code chunk is replaced with the definition provided in `expar`.

To work properly, the parameter of interest must be defined with a standard assignment in the first code chunk at the beginning of a line. The original definition of the parameter must be in a single line only (typically using something like `sample` or `runif` etc.).

After replacing the code chunk, `expar` returns the path to the temporary file with the modified exercise. This can then be processed with `exams2xyz` "as usual".

Value

A character string with the file path of the modified exercise.

Examples

```
## fix parameters "a" and "c" in deriv.Rmd
## (but still generate "b" randomly)

## HTML output
if(interactive()){
  exams2html(expar("deriv.Rmd", a = 1, c = 0))
}

## just the question text
x <- xexams(expar("deriv.Rmd", a = 1, c = 0))
writeLines(x[[1]][[1]]$question)
```

Description

Auxiliary functions for displaying numeric elements in exercises.

Usage

```
fmt(x, digits = 2L, zeros = digits < 4L, ...)

round2(x, digits = 0)

char_with_braces(x)

num_to_tol(x, reltol = 0.0002, min = 0.01, digits = 2)

## S3 method for class 'matrix'
toLatex(object, skip = FALSE, fix = getOption("olat_fix"),
        escape = TRUE, ...)

## S3 method for class 'data.frame'
toLatex(object, rotate = FALSE, pad = " ~ ", align = NULL, row.names = FALSE, ...)
```

Arguments

x	numeric vector.
digits	integer. Digits that should be used for rounding.
zeros	logical. Should trailing zeros be added?
reltol	numeric. Relative tolerance (relative to correct solution x).
min	numeric. Minimum absolute tolerance.
object	matrix or data frame, respectively.
skip	logical. Should an additional skip be added between rows?
fix	logical. Should an additional empty column be added between all columns? This is a workaround for OLAT that collapses spaces between columns in MathML.
escape	logical. Should LaTeX commands be escaped (as appropriate for Sweave) or not (as appropriate for knit)?
...	passed to format for <code>fmt</code> .
rotate	logical. Should the table be transposed/rotated by 90 degrees?
pad	character for padding columns of the resulting table.
align	character indicating the alignment of the columns. Can either be a single string like " <code> l rrr </code> " or a vector of characters per column. By default numeric columns are right-aligned and character columns are left-aligned.
row.names	logical. Should a column (or row, if <code>rotate = TRUE</code>) with the row names be included?

Details

Various functions that help displaying numerical results in exercises:

The function `fmt` rounds and adds trailing zeros (by default if `digits` is lower than 4).

The function `round2` does what is known in German as *kaufmaennisches Runden* (rounding away from zero for trailing 5s).

The function `char_with_braces` adds parentheses for negative elements (in order to facilitate their display in equations).

The function `num_to_tol` (or equivalently `num2tol`) computes the absolute tolerance based on a numeric solution `x` and a relative tolerance `reltol`.

The `toLatex` method sets up a matrix array with parentheses.

Examples

```
## emulate how students round
## (rather than using the round-to-even strategy R employs)
round2(c(0.005, 0.015), digits = 2)
round(c(0.005, 0.015), digits = 2)

## this is also employed internally in the fmt() formatting function
fmt(c(0.005, 0.015))

## the main purpose of fmt() is that some numeric result can be displayed
## both at high accuracy and then at the rounding that students should do
## (e.g., with 2 or 3 digits)
sol <- runif(1)
fmt(sol, 6)
fmt(sol, 2)

## but fmt() also assures showing a very high number of significant digits
## (up to 12)
sol <- 123456 + sol
sol
fmt(sol, 6)
fmt(sol, 2)

## and fmt() also takes care of adding trailing zeros (if digits < 4)
fmt(1)
fmt(1, digits = 3)
fmt(1, digits = 6)

## char_with_braces() is for adding parentheses, e.g., before constructing a sum
paste(char_with_braces(-2:2), collapse = " + ")

## for including a matrix in a LaTeX formula
x <- matrix(1:4, ncol = 2)
toLatex(x)
toLatex(x, skip = TRUE)

## for including a data frame as a plain LaTeX tabular (without caption etc.)
d <- data.frame(Label = c("Foo first", "Bar second"), Value = c(12.3, 1234))
```

```

toLatex(d, big.mark = ",", nsmall = 2)

## compute absolute tolerances:
## minimum is 0.01
num_to_tol(1)
## but can be larger for larger solutions
num_to_tol(100)

```

include_supplement *Copy (Static) Supplement Files for Inclusion in Exercises*

Description

Copy (static) files (e.g., graphics, data sets, etc.) for inclusion as supplements in an exercise.

Usage

```
include_supplement(file, dir = NULL, recursive = FALSE, target = NULL)
```

Arguments

file	character. A (vector of) file name(s).
dir	character. The directory where file can be found. If used within the code chunks of exercises, the default is to use the directory in which the exercises are stored.
recursive	logical. Should also sub-directories of dir be searched for file?
target	character. A (vector of) target file name(s), by default taken to be the same as file.

Details

Usually, supplement files are created dynamically within an exercise, e.g., data is simulated and then plotted or stored in a file etc. However, sometimes an exercises wants to include a static supplement file that is available in some directory on the system. Then, the `include_supplement` is a convenience function that copies such a file from its directory into the supplements of an exercise. Then it can be included/referenced as usual in the question/solution text.

Examples

```

## The "Rlogo" exercise uses a static image which is provided
## within the "exams" package.
if(interactive()) {
  exams2html("Rlogo.Rnw")
}

```

include_tikz

Including Figures from TikZ Code in Exercises

Description

Include figures from TikZ code in an exercise after compiling it with `tex2image`.

Usage

```
include_tikz(tikz, name = "tikzpicture", format = NULL,
            library = NULL, width = NULL, markup = "tex", ...)
```

Arguments

<code>tikz</code>	character vector with the TikZ code.
<code>name</code>	character. Name prefix of the graphics file to be produced.
<code>format</code>	character. The graphics format requested from <code>tex2image</code> , e.g., "png" (default), "svg", "pdf". If set to "tex" then <code>tex2image</code> is not called but the tikz code is included directly.
<code>library</code>	character. Names of TikZ libraries required for compiling the tikz code (if any).
<code>width</code>	character. The width with which the resulting graphic should be included in LaTeX.
<code>markup</code>	character. Which type of markup should be written? Can be "tex" (default), "markdown", or "none".
<code>...</code>	arguments passed to <code>tex2image</code> .

Details

The function `include_tikz` takes a character vector with tikz code, if necessary adds a `{tikzpicture}` environment, renders it into a graphics file via `tex2image`, and returns LaTeX or Markdown code that embeds the graphics into an exercise.

If `format = "tex"` and `markup = "tex"` the TikZ code is included directly (possibly adding `library` and `{tikzpicture}`, if necessary).

Value

A character vector is returned. This contains just the name of the graphics file produced (i.e., `name.format`) except for `format = "tex"` where the TikZ code is returned. For `markup = "tex"` or "markdown" the value is returned invisibly.

Examples

```

## TikZ code for a logic gate
tz <- "
  \node[left,draw, logic gate inputs=nn, xor gate US,fill=white,,scale=2.5] (G1) at (0,0) {};
  \draw (G1.output) ---+ (0.5,0) node[right] (y) {$y$};
  \draw (G1.input 1) ---+ (-0.5,0) node[left] {$a$};
  \draw (G1.input 2) ---+ (-0.5,0) node[left] {$b$};
"

## switch to temporary directory
wd <- getwd()
td <- tempfile()
dir.create(td)
setwd(td)
dir()

## produce PDF figure and produce includegraphics statement
include_tikz(tz, name = "logicgate", format = "pdf",
  library = c("arrows", "shapes.gates.logic.US", "calc"),
  width = "2.5cm")
dir()

## alternatively produce just the complete TikZ code
include_tikz(tz, name = "logicgate", format = "tex",
  library = c("arrows", "shapes.gates.logic.US", "calc"))

## switch back to original working directory
setwd(wd)

```

```
make_exercise_transform_pandoc
```

Transform Exercises via Pandoc

Description

Generate an exercise transformer function based on [pandoc_convert](#).

Usage

```
make_exercise_transform_pandoc(to = "latex", base64 = to != "latex",
  attachfile = FALSE, ...)
```

Arguments

`to` character. Specification of the output text format, typically "latex", "html", or "markdown".

base64	logical. Should supplementary files be embedded using Base 64 coding? Argument base64 may also be a character vector of file suffixes that should be encoded, e.g. <code>base64 = c("png", "rda")</code> will only encode PNG images and binary <code>.rda</code> files. If set to <code>NULL</code> only image files will be encoded.
attachfile	logical. Should <code>attachfile</code> rather than <code>url</code> be used in LaTeX, e.g., for embedding data files in PDF.
...	arguments to be passed on to pandoc_convert .

Details

The generator function `make_exercise_transform_pandoc` returns a function that can be used for the transform steps in [xexams](#). It is a wrapper to [pandoc_convert](#) from `rmarkdown` but adds a couple of convenience features that are typically needed in R/exams exercises. Supplementary files can be handled by Base 64 encoding (often used for HTML output) or via `attachfile` in LaTeX (sometimes useful for PDF output). Some additional LaTeX commands and environments are supported, e.g., Sweave environments or negated logical comparison symbols. Finally, some default options from pandoc are changed, namely the defaults `--wrap=preserve` (instead of `--wrap=auto`) and `--columns=99999` (instead of `--columns=72`).

Value

`make_exercise_transform_pandoc` returns a function that is suitable for being supplied as `driver$transform` to [xexams](#).

See Also

[xexams](#), [make_exercise_transform_html](#), [pandoc_convert](#)

Examples

```
## load package and enforce par(ask = FALSE)
options(device.ask.default = FALSE)

## default transformation to LaTeX output
textrafo <- make_exercise_transform_pandoc()

## custom transformation to Markdown output without Base 64 files
mdtrafo <- make_exercise_transform_pandoc(to = "markdown", base64 = FALSE)

## read "lm" exercise via xexams, i.e., without transformation
## Rmd yields Markdown, Rnw yields LaTeX
lm_md <- xexams("lm.Rmd")[[1]][[1]]
lm_tex <- xexams("lm.Rnw")[[1]][[1]]

## original Markdown and transformed LaTeX
writeLines(lm_md$question)
writeLines(textrafo(lm_md)$question)

## original LaTeX and transformed Markdown
writeLines(lm_tex$question)
writeLines(mdtrafo(lm_tex)$question)
```

match_exams_call	<i>Query Information from Last xexams/exams2xyz Call</i>
------------------	--

Description

match_exams_call queries the last call made to xexams (typically through some exams2xyz interface). match_exams_device queries the graphics device used in the last xweave call. match_exams_iteration queries the iteration (within n replications) that is currently processed by xexams.

Usage

```
match_exams_call(which = 1L, deparse = TRUE)
match_exams_device()
match_exams_iteration()
```

Arguments

which	integer. Specifies the hierarchy level at which the exams2xyz call should be extracted.
deparse	logical. Should only the deparsed function name be computed (or the entire call)?

Details

The function match_exams_call is useful for determining within an exercise which exams2xyz interface is used in order to behave slightly differently, e.g., for PDF vs. HTML output. (This feature only works from R 3.2.0 onwards.)

Similarly, the function match_exams_device can be used within an exercise to use the same graphics device that xweave is using.

Finally, the function match_exams_iteration can be used to find out which iteration (within n replication) is currently processed by xexams or exams2xyz. This is useful if an exercise wants to iterate through a certain grid of parameter settings.

Examples

```
## call exams2nops
dir.create(tdir <- tempfile())
exams2nops("tstat2.Rmd", dir = tdir)
match_exams_call()

## graphics device used
match_exams_device()

## exams2nops called exams2pdf called xexams:
match_exams_call(which = NULL)

## get full exams2nops call
```

```

match_exams_call(deparse = FALSE)

## but note that convenience wrappers etc. are included
e2n <- function(...) exams2nops(...)
e2n("tstat2.Rmd", dir = tdir)
match_exams_call(which = NULL)

```

matrix_to_schoice	<i>Generate Single- and Multiple-Choice Question Lists for Matrix Solutions</i>
-------------------	---

Description

Functions for generating single- and multiple-choice question lists for a matrix solution. (Optimized for integer matrices.)

Usage

```

matrix_to_schoice(x, y = NULL, lower = FALSE, name = "a",
  delta = 0.5, digits = 0)

matrix_to_mchoice(x, y = NULL, lower = FALSE, name = "a",
  comparisons = c("==", "<", ">", "<=", ">="), restricted = FALSE)

det_to_schoice(x, y = NULL, range = NULL, delta = 0.5, digits = 0)

```

Arguments

x	matrix (correct result).
y	numeric vector (optional) with (potentially) wrong solutions/comparisons.
lower	logical. Should only elements from the lower triangle be assessed?
name	character. Base name for matrix elements.
delta	numeric. Minimal distance between solutions.
digits	integer. Digits that should be displayed.
comparisons	character. Vector of logical comparisons that should be employed.
restricted	logical. Should the result be restricted to at least one correct and one wrong solution/comparison?
range	numeric vector of length 2 (optional) with range of random wrong solutions.

Details

The function `matrix_to_schoice` (or equivalently `matrix2schoice`) can be used for generating a single-choice question list for a correct result matrix x . One element is picked randomly from the matrix and chosen to be the correct solution. Other values from the observed absolute range are used as wrong solutions by default (if y does not provide an alternative list of potential solutions).

The function `matrix_to_mchoice` (or equivalently `matrix2mchoice`) can be used for generating a multiple-choice question list for a correct result matrix x . Each item from the question list is a logical comparison of one matrix element with a comparison value. By default the comparisons are picked randomly from the observed absolute range (unless y specifies a different list of comparisons).

In both `matrix_to_schoice` and `matrix_to_mchoice` it is also possible to provide a vector x rather than a matrix. Then the corresponding question list is shown with a single index only (say a_1) rather than two indexes (say a_{11}).

The function `det_to_schoice` (or equivalently `det2schoice`) can be used for generating a single-choice question list for the determinant of a 2x2 matrix. It has been optimized for matrices with single-digit integer elements. It may not yield very balanced random solutions for other scenarios.

Value

`matrix_to_schoice/matrix2schoice` returns a list with the following components:

<code>index</code>	numeric vector with matrix index of the correct solution chosen.
<code>name</code>	character with LaTeX code for the correct matrix element chosen.
<code>solutions</code>	a logical vector of length 5 indicating the correct solution,
<code>questions</code>	a character vector of length 5 with question list.

`matrix_to_mchoice/matrix2mchoice` returns a list with the following components:

<code>solutions</code>	a logical vector of length 5 indicating the correct solution,
<code>questions</code>	a character vector of length 5 with question list.
<code>explanations</code>	a character vector of length 5 with explanations why the solutions are correct or wrong.

`det_to_schoice/det2schoice` returns a list with the following components:

<code>solutions</code>	a logical vector of length 5 indicating the correct solution,
<code>questions</code>	a character vector of length 5 with question list.

See Also

[num_to_schoice](#)

Examples

```
A <- matrix(c(-9, 0, 5, -2), ncol = 2)
matrix_to_schoice(A)
matrix_to_mchoice(A)
det_to_schoice(A)
```

```
B <- matrix(1:9, ncol = 3)
matrix_to_schoice(B)
matrix_to_mchoice(B)
```

mchoice2string

Convenience Functions for Exam Formatting

Description

A collection of convenience functions for formatting in exam generation that can be used for switching between suitable logical/text/numeric representations of multiple choice solutions.

Usage

```
mchoice2string(x, single = FALSE)
string2mchoice(x, single = FALSE)
mchoice2text(x, markup = c("latex", "markdown"))
answerlist(..., sep = ". ", markup = c("latex", "markdown"))
```

Arguments

x	an object, see below for examples.
single	logical. Should the function check whether exactly a single answer is true?
...	character vectors to be included in answer lists.
sep	character for separation between vectors, see below for examples.
markup	character indicating which markup (LaTeX vs. Markdown) should be generated.

Details

Three convenience functions for facilitating work with multiple choice solutions of exams. All have almost trivial definitions, see also examples below.

See Also

[exams](#)

Examples

```
## multiple choice answer
mc <- c(TRUE, FALSE, TRUE, FALSE, FALSE)

## switching to string representation
mchoice2string(mc)

## reverse string encoding
string2mchoice("10100")

## switching to text
mchoice2text(mc)

## generating answerlist based on questions,
## solutions and explanations
qu <- c("Zurich is the capital of Switzerland.",
        "Italian is an official language in Switzerland.",
        "Switzerland is part of the European Union.")
sol <- c(FALSE, TRUE, FALSE)
ex <- c("The capital of Switzerland is Bern.",
        "The four official languages are: German, French, Italian, Romansh.",
        "Switzerland is part of the Schengen Area but not the European Union.")
answerlist(qu)
answerlist(iffelse(sol, "True", "False"), ex)
```

moodle2exams

Convert Moodle XML Quizzes to R/exams Exercises

Description

Function to convert Moodle quiz exercises of type numerical, multichoice, shortanswer, and essay to R/exams exercises of type num, schoice/mchoice, and string.

Usage

```
moodle2exams(x, markup = c("markdown", "latex"), dir = ".",
             exshuffle = TRUE, names = NULL)
```

Arguments

x	character. Path to a Moodle XML file. If a character vector with more than one element is provided, it is assumed to be an XML file as read with readLines .
markup	character. Markup language to convert to, i.e., "markdown" (default) corresponds to Rmd exercises and "latex" to Rnw exercises.
dir	character. Directory where the converted exercises should be saved. If set to NULL no files are saved.
exshuffle	logical or numeric. Meta-information tag used for single-choice and multiple-choice items.

names character. Optional file names (without suffix) that should be used for the R/exams exercise files. The default is to use the name tags from the Moodle XML file (with some fix-ups, avoiding certain special characters). Alternatively, names can also be supplied and will then be used for both the file names and the exname meta-information tag, thus overwriting other names specified in the Moodle XML file.

Details

The function aims to facilitate converting an existing Moodle question pool to R/exams exercises. The resulting exercise files can subsequently be edited further, e.g., for making them dynamic.

The function takes a Moodle XML quiz and converts each exercise into a R/Markdown (Rmd) or R/LaTeX (Rnw) R/exams exercise. The HTML answers and questions from the Moodle XML are converted using pandoc (via [pandoc_convert](#)). It is recommended to check the outcome in case certain HTML markup, or mathematical equations, etc., cannot be converted fully automatically. Currently only the Moodle XML exercise types numerical, multichoice, shortanswer, and essay are properly supported.

There is limited support for cloze exercises, but the resulting meta-information needed by R/exams will be incorrect. Hence, a warning is issued when converting cloze exercises.

Value

A list of character vectors containing the R/exams exercise code with one element per exercise. If `dir` is specified (default), these character vectors are saved in one file per exercise (using [writelines](#)). In this case the list is returned invisibly. If `dir = NULL` no files are saved and the list is returned visibly.

See Also

[exams2moodle](#)

Examples

```
if(requireNamespace("xml2")) {
## path to a Moodle XML quiz file (provided in the package)
moodle_xml <- file.path(find.package("exams"), "xml", "moodlequiz.xml")

## create a temporary directory for R/exams exercise files
dir.create(tdir <- tempfile())

## convert all exercises from the Moodle XML to R/Markdown files
ex_converted <- moodle2exams(moodle_xml, dir = tdir)
print(dir(tdir))

## additionally the source code of the Rmd files is also return invisible
## in 'ex_converted' and can be inspected manually, e.g., via writelines()
names(ex_converted)
writeLines(ex_converted[[1]])

## clean up temporary directory
```

```

  unlink(tdir)
}

```

nops_eval

Evaluate NOPS Exams

Description

Evaluate NOPS exams produced with [exams2nops](#), and scanned by [nops_scan](#).

Usage

```

nops_eval(register = dir(pattern = "\\*.csv$"), solutions = dir(pattern = "\\*.rds$"),
  scans = dir(pattern = "^nops_scan_[[:digit:]]*\\.zip$"),
  points = NULL, eval = exams_eval(partial = TRUE, negative = FALSE, rule = "false2"),
  mark = c(0.5, 0.6, 0.75, 0.85), labels = NULL,
  dir = ".", results = "nops_eval", file = NULL, flavor = NULL,
  language = "en", interactive = TRUE,
  string_scans = dir(pattern = "^nops_string_scan_[[:digit:]]*\\.zip$"),
  string_points = seq(0, 1, 0.25),
  ...)

```

```

nops_eval_write(results = "nops_eval.csv", file = "exam_eval",
  dir = ".", language = "en", converter = NULL,
  col = hcl(c(0, 0, 60, 120), c(70, 0, 70, 70), 90), encoding = "UTF-8",
  html = NULL)

```

Arguments

register	character. File name of a CSV file (semicolon-separated) of the registered students. Must contain columns "registration" (registration number), "name" (student name), "id" (some user name or other string unique for each student). The file name should not contain spaces, umlaut or other special characters (e.g., something like "exam-2015-07-01.csv" is recommended).
solutions	character. File name of the RDS exam file produced by exams2nops .
scans	character. File name of the ZIP file with scanning results (containing Daten.txt and PNG files) as produced by nops_scan .
points	numeric. Vector of points per exercise. By default read from solutions.
eval	list specification of evaluation policy as computed by exams_eval .
mark	logical or numeric. If mark = FALSE, no marks are computed. Otherwise mark needs to be a numeric vector with threshold values to compute marks. The thresholds can either be relative (all lower than 1) or absolute. In case results exactly matching a threshold, the better mark is used.
labels	character. Vector of labels for the marks with length(mark) + 1 elements. Default is (length(mark) + 1):1.

dir	character. File path to the output directory (the default being the current working directory).
results	character. Prefix for output files.
file	character. File name for individual report files, in the default nops_eval_write the same as register with suffix .html.
flavor	character. Rather than using the default nops_eval_write it is possible to call nops_eval_write_<flavor>. Currently, only the default writer is available but other formats are planned (e.g., Moodle).
language	character. Path to a DCF file with a language specification. Currently, the package ships: English ("en"), Dutch ("nl"), French ("fr"), German ("de"), Italian ("it"), Romanian ("ro"), Portuguese ("pt"), Spanish ("es"), Turkish ("tr").
interactive	logical. Should possible errors in the Daten.txt file be corrected interactively? Requires the png package for full interactivity.
string_scans	character. Optional file name of the ZIP file with scanning results of string exercise sheets (if any) containing Daten2.txt and PNG files as produced by nops_scan .
string_points	numeric. Vector of length 5 with points assigned to string results, multiplied with the points for the corresponding exercises.
...	further optional arguments passed from nops_eval to nops_eval_write (or any other nops_eval_write_<flavor>).
converter	character. The HTML converter to be used for the language text elements. Can be "none", "tth", or "pandoc".
col	character. Hex color codes used for exercises with negative, neutral, positive, full solution.
encoding	character. Encoding of register, e.g., "latin1" or "UTF-8" (default).
html	character. Alternative specification of the file name for the individual HTML report files. As html was used rather than file in earlier versions of the package, html is preserved as an argument and overwrites file if specified.

Details

nops_eval is a companion function for [exams2nops](#) and [nops_scan](#). It evaluates the scanned exams by computing the sums of the points achieved and (if desired) maps them to marks. Furthermore a HTML report for each individual student is generated by default (e.g., for upload into a learning management system).

Tutorial for NOPS workflow: <https://www.R-exams.org/tutorials/exams2nops/>.

Value

A data.frame with the detailed exam results is returned invisibly. It is also written to a CSV file in the current directory, by default along with a ZIP file containing HTML reports.

See Also

[exams2nops](#), [nops_scan](#)

Examples

```

## --- Preliminaries ---

## load package and enforce par(ask = FALSE)
## additionally, for simplicity, enforce using the basic
## tools::texi2dvi() LaTeX interface instead of the more
## flexible/robust tinytex::latexmk()
library("exams")
oopt <- options(device.ask.default = FALSE, exams_tex = "tools")

## set up a temporary working directory in which all files are managed
odir <- getwd()
dir.create(mydir <- tempfile())
setwd(mydir)

## --- Step 1 ---
## exam generation

## define an exam (= list of exercises)
myexam <- list(
  "tstat2.Rmd",
  "tttest.Rmd",
  "relfreq.Rmd",
  "anova.Rmd",
  c("boxplots.Rmd", "scatterplot.Rmd"),
  "cholesky.Rmd"
)

## create multiple exams on the disk with different numbers of points
## per exercise (see ?exams2nops for more examples)
set.seed(403)
ex1 <- exams2nops(myexam, n = 2, dir = ".", date = "2015-07-29",
  points = c(1, 1, 1, 2, 2, 3), showpoints = TRUE)
dir()

## assume the PDF exams were already printed (and possibly backed up
## in a different directory) so that they are not needed anymore
file.remove(dir(pattern = "pdf$"))

## --- Step 2 ---
## scan results

## assume two participants filled out the printed exam sheets
## and the corresponding scans are in two PNG files,
img <- dir(system.file("nops", package = "exams"), pattern = "nops_scan",
  full.names = TRUE)

## copy the PNG files to the working directory
file.copy(img, to = ".")

```

```

## read the scanned images (all locally available .png files) and collect
## results in a ZIP archive (see ?nops_scan for more details)
nops_scan()
dir()

## the ZIP archive contains copies of the PNG images so that these are
## can be deleted here (possibly after backup in a different directory)
file.remove(dir(pattern = "png$"))

## -- Step 3 ---
## evaluate results

## three files are required: (a) an RDS file with the exam meta-information
## (see Step 1), (b) a ZIP file with the scanned sheets (see Step 2), (c) a
## CSV file with the student information (registration number, name, and some
## for of ID/username)

## here we create the CSV file on the fly but in practice this will typically
## be processed from some registration service or learning management system etc
write.table(data.frame(
  registration = c("1501090", "9901071"),
  name = c("Jane Doe", "Ambi Dexter"),
  id = c("jane_doe", "ambi_dexter")
), file = "Exam-2015-07-29.csv", sep = ";", quote = FALSE, row.names = FALSE)
dir()
## now the exam can be evaluated creating an output data frame (also stored
## as CSV file) and individual HTML reports (stored in a ZIP file),

## as there is only exactly on CSV/RDS/ZIP file in the current directory,
## these are found automatically - furthermore an evaluation scheme without
## partial points and differing points per exercise are used
ev1 <- nops_eval(eval = exams_eval(partial = FALSE, negative = FALSE))
dir()

## inspect evaluated data
ev1

## inspect corresponding HTML reports
if(interactive()) {
  unzip("nops_eval.zip")
  browseURL(file.path(mydir, "jane_doe", "Exam-2015-07-29.html"))
  browseURL(file.path(mydir, "ambi_dexter", "Exam-2015-07-29.html"))
}

## --- Options ---
if(interactive()) {
  ## below three typically needed options are discussed:
  ## (a) using a different evaluation strategy (here with partial credits),
  ## (b) using a different language (here de/German),
  ## (c) an error of the participant when filling in the registration number.

  ## as for (a): partial credits should only be used for multiple-choice questions

```

```

## where at least one alternative is correct and at least one is false
## [note that in this example this is not the case for the first question
## (single-choice) and the third question for Jane Doe (no alternative correct)]

## as for (c): for Ambi Dexter such an error was included in the PNG example
## image, the actual number is "9911071" but the crosses indicate "9901071"

## clean up previous evaluation
file.remove(c("nops_eval.csv", "nops_eval.zip"))

## write correct registration information
write.table(data.frame(
  registration = c("1501090", "9911071"),
  name = c("Jane Doe", "Ambi Dexter"),
  id = c("jane_doe", "ambi_dexter")
), file = "Exam-2015-07-29.csv", sep = ";", quote = FALSE, row.names = FALSE)

## call nops_eval() with modified options, where the error in the registration
## number of Ambi Dexter will trigger an interactive prompt
ev2 <- nops_eval(eval = exams_eval(partial = TRUE, rule = "false2"),
  language = "de")

## inspect evaluated data
ev2
cbind(ev1$points, ev2$points)

## inspect corresponding HTML reports
unzip("nops_eval.zip")
browseURL(file.path(mydir, "jane_doe", "Exam-2015-07-29.html"))
browseURL(file.path(mydir, "ambi_dexter", "Exam-2015-07-29.html"))
}

## switch back to original working directory and options
setwd(odir)
options(exams_tex = oopt$exams_tex)

```

nops_fix

Fix Data from Scanned NOPS Exams

Description

Fix scanned NOPS exams produced with [nops_scan](#) and update the corresponding ZIP file.

Usage

```

nops_fix(scans = dir(pattern = "^nops_scan_[[:digit:]]*\\.zip$"),
  exam = NULL, id = NULL, field = NULL, answer = NULL, check = NULL, display = NULL,
  string = NULL)

```

Arguments

scans	character. File name of the ZIP file with scanning results (containing Daten.txt and PNG files) as produced by <code>nops_scan</code> .
exam	integer. Rows number from the scanned data which should (potentially) be fixed (default: all rows).
id	integer or character. Either a vector with full 11-digit exam IDs for the exams to be checked. Or the last five digits of the full 11-digit ID (where leading zeros can be omitted).
field	character indicating which field(s) from the scanned data should be fixed. One or more of "type" (3-digit exam sheet type), "id" (11-digit exam ID), "registration" (registration ID, can be between 7 and 10 digits), "answers" (individual answer checkboxes). By default all fields with invalid entries have to be fixed.
answer	integer indicating which answer checkboxes should (potentially) be fixed if <code>field</code> contains "answers" (default: all answers).
check	character indicating additional check conditions for the answers: If set to "missing" only the missing answers are considered. For "schoice" answers with more than one checked box are considered. For "mchoice" answers with all boxes checked are considered.
display	character indicating how the scanned image should be displayed. One or both of "plot" (display scanned excerpt in R plot) and "browser" (display full scanned image in browser). By default "plot" is used if the png package is available and "browser" otherwise.
string	logical. Is the ZIP file from scanning marked string exercises (rather than single/multiple choice exercises)? The default is TRUE if the file name starts with <code>nops_string_scan_</code> and FALSE otherwise.

Details

`nops_fix` is a companion function for `exams2nops` and `nops_scan`. To fix problems that potentially occurred when extracting the exam information from the scanned PDF/PNG images, `nops_fix` can iterate through the scanned data and interactively prompt for fields that (potentially) need to be fixed. Typical cases are when errors occurred in entering the registration ID or when the entire exam sheet could not be read correctly. See below for a couple of typical application examples.

For each specified field the user is prompted for a potential update. By just pressing ENTER the current value of the field is preserved. If an update is not a valid specification of the field, the prompt is repeated (along with some additional instructions) until a valid specification is entered by the user.

The type of sheed needs to be a 3-digit number, the exam ID needs to be an 11-digit number, and the registration ID needs to be a number with 7-10 digits.

All answer fields eventually store the multiple-choice answers with 0/1 indicators of length 5. The following input formats are accepted:

- 0/1 indicators of length up to 5. If less then 5 digits are specified, the remaining digits are filled with 0s.
- Letters from a to e indicating the box(es) that have been checked.

- Integers from 1 to 5 indicating which single box has been checked.
- No checked box can be indicated by "0", "-", or " ".

Value

Data frame with one line per scanned file is returned invisibly. The output contains the following columns: file name, sheet ID (11 digits), scrambling (2 digits), type of sheet (3 digits, coding the number of questions rounded up to steps of 5 and the length of the registration number), 0/1 indicator whether the replacement sheet was used, registration number (7-10 digits), 45 multiple choice answers of length 5 (all 00000 if unused).

See Also

[exams2nops](#), [nops_scan](#), [nops_eval](#)

Examples

```
## typical application cases (not run), all assume that there is
## a single nops_scan_*.zip file in the current working directory

## fix all rows/fields that can be detected as incorrect
## nops_fix()

## fix answer 6 in exam 3
## nops_fix(exam = 3, answer = 6)

## fix all answers in exam 3
## nops_fix(exam = 3, field = "answers")

## fix all fields exam 3
## nops_fix(exam = 3, field = c("type", "id", "registration", "answers"))

## fix answer 6 in exam with id 23112900129
## nops_fix(id = "23112900129", answer = 6)
## nops_fix(id = 129, answer = 6)

## fix all answers in questions 1 to 8 where more than one box was checked
## nops_fix(answer = 1:8, check = "schoice")
```

nops_language

Read NOPS Language Specification

Description

Read a NOPS language specification from a DCF file and optionally convert the language text to HTML.

Usage

```
nops_language(file, converter = c("none", "tth", "pandoc"), ...)
```

Arguments

file	character. Path to a DCF file with a language specification. See exams2nops for a list of languages shipped in the package.
converter	character. The HTML converter to be used for the language text elements. Can be "none", "tth", or "pandoc".
...	further arguments passed to the converter (if any), e.g., for obtaining output formats other than HTML.

Details

The NOPS exams infrastructure is internationalized and can be customized through DCF files (borrowing the format from Debian configuration files). For a detailed description see: https://www.R-exams.org/tutorials/nops_language/. The DCF files typically either contain special characters as LaTeX commands or in UTF-8 encoding. To handle the former case, a converter can be applied to convert the language texts to HTML.

Value

A list with all language components provided by the DCF file.

See Also

[exams2nops](#), [nops_eval](#)

Examples

```
## English
en <- nops_language("en")
names(en)
en$NoChanges

## French (LaTeX vs. HTML)
nops_language("fr", converter = "none")$NoChanges
nops_language("fr", converter = "tth")$NoChanges

## French (HTML or Markdown via pandoc)
nops_language("fr", converter = "pandoc")$NoChanges
nops_language("fr", converter = "pandoc", to = "markdown_strict")$NoChanges
```

nops_scan

Read Scanned NOPS Exams

Description

Read scanned NOPS exams produced with [exams2nops](#).

Usage

```
nops_scan(
  images = dir(pattern = "\\\\.PNG$|\\.png$|\\.PDF|\\.pdf$",
    path = dir, full.names = TRUE),
  file = NULL, dir = ".",
  verbose = TRUE, rotate = FALSE, cores = NULL, n = NULL,
  density = 300,
  size = 0.03, threshold = c(0.04, 0.42), trim = 0.3, minrot = 0.002,
  string = FALSE)
```

Arguments

images	character. Names of the PDF/PNG images containing the scanned exams. By default all PDF/PNG images in the current working directory are used.
file	character or logical. Optional file name for the output ZIP archive containing the PNG images and the scan results. If file = FALSE no ZIP archive is created. By default a suitable name using the current time/date is used.
dir	character. Directory in which the ZIP file should be created. By default the current working directory.
verbose	logical. Should progress information be displayed?
rotate	logical. Should the input PDF/PNG images be rotated by 180 degrees first?
cores	numeric. If set to an integer <code>mclapply</code> (or <code>parLapply</code> on Windows) is called internally using the desired number of cores to read the scanned exams in parallel.
n	numeric. The number of answer fields to read (in multiples of 5), i.e., 5, 10, ..., 45. By default taken from the type field.
density	numeric. Resolution used in the conversion of PDF images to PNG. This requires ImageMagick's <code>convert</code> to be available on the system.
size	numeric. Size of the boxes containing the check marks relative to the image height. Should typically be between 0.025 and 0.045.
threshold	numeric. Vector of thresholds for the gray levels in the check mark boxes. If the average gray level is between the gray levels, the box is checked. If it is above the second threshold, some heuristic is employed for judging whether the box contains a cross or not.
trim	numeric. Amount of trimming to shave the borders of the boxes before determining the gray level within the check boxes. Should usually be between 0.25 (default up to version 2.3-1) and 0.35.
minrot	numeric. Minimum angle for rotating images, i.e., images with a lower angle are considered to be ok.
string	logical. Are the files to be scanned manually marked string exercises (rather than single/multiple choice exercises)?

Details

nops_scan is a companion function for [exams2nops](#). Exams generated with exams2nops can be printed and the filled out answer sheet can be scanned. Then, nops_scan can be employed to read the information in the scanned PDF/PNG images. The results are one text line per image containing the information in a very simple space-separated format.

If images only contains PNG files, then the R function [readPNG](#) is sufficient for reading the images into R. If images contains PDF files, these need to be converted to PNG first, which is carried out using the R packages **qpdf** and **magick**. (Instead of using these R packages it is also possible to use system calls to PDFtk, GhostScript, and ImageMagick's convert, which was the only implementation up to R/exams version 2.4-0.)

Tutorial for NOPS workflow: <https://www.R-exams.org/tutorials/exams2nops/>.

Practical recommendations:

The scanned images produced by scanners or copying machines typically become smaller in size if the images are read in just black/white (or grayscale). This may sometimes even improve the reliability of reading the images afterwards. Also make sure that the resulting images have a good contrast and are neither too light nor too dark because too many or too little dark pixels increase the probability of incorrect scanning results.

Make sure that the sheets are fed firmly into the scanner, e.g., by tightening the tracks of the feeder.

The printed exams are often stapled in the top left corner which has to be unhinged somehow by the exam participants. Although this may damage the exam sheet, this is usually no problem for scanning it. However, the copying machine's sheet feeder may work better if the sheets are turned upside down (so that the damaged corner is not fed first into the machine). This often improves the scanning results considerably and can be accomodated by setting rotate = TRUE in nops_scan.

Value

A character vector with one element per scanned file (returned invisibly if written to an output ZIP archive). The output contains the following space-separated information: file name, sheet ID (11 digits), scrambling (2 digits), type of sheet (3 digits, coding the number of questions and the length of the registration number), 0/1 indicator whether the replacement sheet was used, registration number (7-10 digits), 45 multiple choice answers of length 5 (all 00000 if unused).

See Also

[exams2nops](#), [nops_eval](#)

Examples

```
if(requireNamespace("png")) {
  ## scanned example images stored in exams package
  img <- dir(system.file("nops", package = "exams"), pattern = "nops_scan",
    full.names = TRUE)

  ## read content
  res <- nops_scan(img, file = FALSE)
  writeLines(res)
}
```

num_to_schoice

Generate Single-Choice Question List from Numeric Solution

Description

A function for generating a single-choice question list for one correct numeric solution along with four wrong solutions.

Usage

```
num_to_schoice(correct, wrong = NULL, range = c(0.5, 1.5) * correct,
  delta = 1, digits = 2, method = c("runif", "delta"), sign = FALSE,
  verbose = getOption("num_to_choice_warnings"))
```

Arguments

correct	numeric vector of length 1 with correct solution.
wrong	numeric vector (optional) with wrong solutions.
range	numeric vector of length 2 with range of random wrong solutions.
delta	numeric. Minimal distance between solutions.
digits	integer. Digits that should be displayed.
method	character specifying method for generating random results.
sign	logical. Should the sign be changed randomly?
verbose	logical. Should warnings be issued if no suitable set of wrong solutions can be found?

Details

The function `num_to_schoice` (or equivalently `num2schoice`) can be used for generating a single-choice question list for a numeric correct solution. The question list always comprises five elements, one of which is the correct solution. The wrong solutions can be provided or are generated randomly. If `wrong` is provided only up to 2 elements of it are used in order to assure some random solutions.

Two methods can be used to generate the wrong solutions: Either simply `runif` or otherwise a full equi-distant grid for the range with step size `delta` is set up from which a discrete uniform sample is drawn. The former is preferred if the range is large enough while the latter performs better if the range is small (as compared to `delta`).

The function tries to avoid patterns in the question list that could be used for guessing the correct solution, e.g., situations where (almost) always the highest (or always the lowest) answer is the correct one. Therefore, internally `num_to_schoice` first randomly decides how many of the 4 wrong solutions should be to the left or to the right of the correct solution, respectively. And in a second step the sampling method is used to find these fixed numbers of wrong solutions to the left and right (if possible!).

Exercise templates using `num_to_schoice` should be thoroughly tested in order to avoid problems with too small ranges or almost identical correct and wrong answers! This can potentially cause problems, infinite loops, etc. See <https://www.R-exams.org/tutorials/stresstest/> for some comments/hints regarding stress-testing of such exercise templates.

Value

`num_to_schoice/num2schoice` returns either NULL (if no suitable question list can be found) or a list with the following components:

`solutions` a logical vector of length 5 indicating the correct solution,
`questions` a character vector of length 5 with question list.

See Also

[matrix_to_schoice](#)

Examples

```
set.seed(1)
## just a correct solution
num_to_schoice(123.45)

## or equivalently
set.seed(1)
num2schoice(123.45)

## just a correct integer solution
num_to_schoice(123, digits = 0)

## a correct solution with a wider range
num_to_schoice(123.45, range = c(0, 200))

## here, the defaults can't work...
## num_to_schoice(0.1234)

## alternatives could be
num_to_schoice(0.1234, range = c(0, 1), delta = 0.03, method = "delta")
num_to_schoice(0.1234, range = c(-5, 5), delta = 0.05)
num_to_schoice(0.1234, wrong = c(0.2749, 1.9723), delta = 0.05)
num_to_schoice(0.1234, wrong = c(0.2749, 1.9723), range = c(-5, 5), delta = 0.05)
```

read_exercise

Reading LaTeX/Markdown Exercise Files

Description

Reading an exercise in either LaTeX format (i.e., after [Sweave](#) was run) or Markdown format (i.e., after [knit](#) was run).

Usage

```
read_exercise(file, markup = NULL, exshuffle = NULL)
read_metainfo(file, markup = NULL, exshuffle = NULL)
```

Arguments

file	character. Name of the LaTeX (.tex) or Markdown (.md) file that should be read into R.
markup	character specifying whether file is a "latex" or "markdown" exercise. By default (NULL) it is inferred from the file extension.
exshuffle	logical or integer. If the exshuffle argument is non-NULL it is used to overrule the exshuffle tag from the file (e.g., exshuffle = FALSE can be used to keep all available answers without permutation).

Details

read_exercise extracts the LaTeX/Markdown code from the question and solution environments/sections of the exercise file, extracting the corresponding answerlists separately (if any). Paths to supplementary files (such as graphics or data files) are stored and the metainformation is extracted (by calling read_metainfo which also includes sanity checks).

The supported metainformation commands are described in detail in vignette("exams2", package = "exams"), see Table 2. Essentially the ex`type` command in is mapped to the `type` element of the returned list etc. (see the Value section below), using the right storage mode for each command (numeric, character, logical). Additionally, there is an `exextra` command which allows to set up arbitrary additional metainformation elements.

Value

read_exercise returns a list with elements

question	a character vector with LaTeX/Markdown code from the question environment (excluding the answerlist environment, if any).
questionlist	a character vector with LaTeX/Markdown code from the answerlist environment within the question environment (if any).
solution	a character vector with LaTeX/Markdown code from the solution environment (excluding the answerlist environment, if any).
solutionlist	a character vector with LaTeX/Markdown code from the answerlist environment within the solution environment (if any).
metainfo	a list of metainformation options, see below.
supplements	a character vector with paths to supplementary files such as graphics or data files (if any).

read_metainfo returns a list with the following elements. Most elements may also be NULL (or empty) if the underlying information is not specified in the file. If file specifies extra information, there may also be additional list elements.

file	character with file name (without extension) of the exercise template.
------	--

markup	character indicating whether "latex" or "markdown" markup is used in the exercise.
type	character indicating exercise type: num, mchoice, schoice, string, or cloze.
name	character with short name/description (to be used for printing within R).
title	character with a pretty longer title.
section	character with sections for groups of exercises (using slashes for subsections like a URL).
version	character with version of exercise.
solution	correct solution. The type/value depends on the type of the exercise: num leads to a numeric vector (of length 1 or 2), mchoice/schoice lead to logical vector, string to a character vector (of length 1), and cloze leads to a list of solutions (depending on clozetype).
tolerance	numeric tolerance limits (of length 1 or 2) for numeric solutions.
clozetype	character indicating the types of the elements of a cloze exercise.
points	numeric with (default) points for correct solution.
time	numeric with (default) time (in seconds) for solution.
shuffle	logical indicating whether mchoice/schoice answers should be shuffled (in Moodle or other e-learning systems).
single	logical indicating whether radio buttons should be used in Moodle.
length	numeric with length of solution.
string	character with a collapsed string of the solution (and tolerance) for backward compatibility with exams .
maxchars	character with the maximum number of characters allowed in QTI text answers (exercise type: string).
abstention	character with the label to be used for an abstention button in schoice/mchoice answers (currently only supported by Moodle).

See Also

[xexams](#)

Examples

```
## xexams() uses read_exercise() by default to read in
## each individual exercise, e.g., here for only a single
## exam with only a single exercise the result is:
set.seed(1090)
xexams("tstat.Rnw")[[1]][[1]]

## the corresponding Markdown version has:
set.seed(1090)
xexams("tstat.Rmd")[[1]][[1]]
```

Description

In order to check the correct behavior of an exercise it is compiled several times. In each iteration the objects created by the exercise are collected and its values can be inspected.

Usage

```
## Stresstest function.
stresstest_exercise(file, n = 100, verbose = TRUE, seeds = NULL,
  stop_on_error = length(as.character(unlist(file))) < 2, ...)

## Plotting stresstest results.
## S3 method for class 'stress'
plot(x, type = c("overview", "solution", "rank", "runtime"),
  threshold = NULL, variables = NULL,
  spar = TRUE, ask = TRUE, ...)
```

Arguments

file	character. A specification of an exercise file. If multiple files should be tested, argument file can also be a vector, matrix or list of files. The latter case sets argument plot = FALSE.
n	integer. The number of replications.
verbose	logical. Should the seeds used for compiling the exercise be prompted on the console.
seeds	The seeds that should be used when compiling the exercise. The default is seeds = 1:n.
stop_on_error	logical. Should the function stop on error or return the the seed, the file name and the error message. Useful when testing a number of exercises.
x	An object returned from function stresstest_exercise.
type	character. type == "overview" plots the basic overview, i.e, the runtimes, numeric solutions, position/number/rank of correct solution(s), if available. If type == "solution", the numeric solutions are plotted against all input parameters stored in the objects element of x. type == "rank" draws spineplots of ranks vs. input parameters stored in objects. type == "runtime" plots the compiling runtimes vs. objects.
threshold	numeric. Can be used to set a threshold, e.g., for numeric solutions a factor is created, factor(solution <= threshold), that is used on the y-axis of a spineplot .
variables	character. The variables that should be used from the objects for plotting.
spar	logical. Should graphical parameters be set or not.

ask	logical. For multiple plots, should the user be asked to hit the return key to see the next plot.
...	Arguments passed to xexams .

Details

In order to check the correct behavior of an exercise function `stresstest_exercise` runs [xexams](#) `n` times using different seeds. If an error occurs when compiling, the error can be reproduced by setting the seed that is prompted at the console and create the exercise again, e.g., with [exams2html](#). This way errors can be detected systematically.

All objects with length 1, which are created by the exercise, are collected in a data frame. These objects are assumed to be input parameters that control the output of the exercise. This can be used to detect certain input values that, e.g., lead to very long run times, or drive the number of correct answers in multiple choice exercises, etc.

For single and multiple choice type question the position(s) of the correct solution(s) is returned. For single choice questions that are created from a numeric version, e.g., using function [num_to_schoice](#) the answers are again converted to numeric and the rank of the correct solution is reported. The rank is sometimes heavily driven by some input parameters, e.g., the correct solution is always the largest or the smallest. For non-numeric choice questions, the rank is based on the lexicographical order of the answerlist.

Value

Function `stresstest_exercise` returns an object of class "stress" (a named list) with the following elements:

seeds	The seeds that where used.
runtime	Compiling times for each iteration.
objects	A data frame of length 1 objects that are created by the exercise.
solution	The numeric solution, availability is depending on the type of exercise.
position	A matrix indicating the position of correct solutions.
rank	The rank of the correct solution, only available for choice exercises.
ntrue	The number of correct answers in multiple choice type questions.

See Also

[xexams](#), [num_to_schoice](#)

Examples

```
## Not run: ## Stress testing.
t1 <- stresstest_exercise("tstat.Rmd", n = 100)
t2 <- stresstest_exercise("tstat2.Rmd", n = 100)

## Plotting.
plot(t1, type = "overview")
plot(t1, type = "solution")
plot(t1, type = "solution", threshold = 30)
```

```

plot(t2, type = "rank")
plot(t2, type = "runtime")

## For custom inspection, object can be
## transformed to a data.frame.
head(as.data.frame(t2))

## Multiple testing.
files <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  "scatterplot.Rmd",
  "relfreq.Rmd"
)
t3 <- stresstest_exercise(files, n = 100)
plot(t3)

## End(Not run)

```

testvision2exams

Convert TestVision XML Questions to R/exams Exercises

Description

Function to convert TestVision questions of type 'invol (numeriek)', 'een-uit-meer', 'meer-uit-meer', and 'open' to R/exams exercises of type num, schoice, mchoice, and string, respectively.

Usage

```
testvision2exams(x, markup = c("markdown", "latex"), rawHTML = FALSE, dir = ".",
  exshuffle = TRUE, name = NULL, shareStats = FALSE, css = FALSE)
```

Arguments

x	character. Path to a TestVision XML file.
markup	character. Markup language to convert to, i.e., "markdown" (default) corresponds to Rmd exercises and "latex" to Rnw exercises.
rawHTML	logical. If rawHTML = TRUE (and markup = "markdown"), instead of conversion via pandoc, the function merely obtains the raw HTML inside the exercise.
dir	character. Directory where the converted exercises should be saved. If set to NULL no files are saved.
exshuffle	logical or numeric. Meta-information tag used for single-choice and multiple-choice items.
name	character. Optional file name (without suffix) that should be used for the R/exams exercise file. The default is to use the name tag from the TestVision XML file (with some fix-ups, avoiding certain special characters). Alternatively, a name

	can also be supplied and will then be used for both the file name and the exname meta-information tag, thus overwriting other name specified in the TestVision XML file.
shareStats	logical indicating whether the taxonomy of statistics exercises as specified by the shareStats project is incorporated in the TestVision XML file. If shareStats = TRUE, the function searches for a text string as specified under 'Leerstof' ('Study Reference' in the English version) in TestVision and converts it into elements of meta-information, such as exsection and exextra[Level].
css	logical indicating whether css files (typically stored in the zip-file's directory 'css') should be read. This allows for copying style definitions for external images such width and height (for now this only works when markup = "markdown").

Details

The function aims to facilitate converting an existing TestVision question to an R/exams exercise. The resulting exercise file can subsequently be edited further, e.g., for making it dynamic.

The function takes a TestVision XML question and converts it into an R/Markdown (Rmd) or R/LaTeX (Rnw) R/exams exercise. If markup = "latex" the HTML answers and questions from the TestVision XML are converted using pandoc (via [pandoc_convert](#)). Similarly, if markup = "markdown" pandoc converts the content to markdown, but when rawHTML = TRUE the function simply copies the HTML content from the TestVision XML (equations are stored within <math> tags). In the latter case, if conversion aims at creating dynamic exercises and displaying equations, it is advised to select and adjust the content in the <annotation> tag which is a raw latex specification of the equation. It is recommended to check the outcome in case certain HTML markup, or mathematical equations, etc., cannot be converted fully automatically.

Currently only the TestVision XML exercise types 'invul (numeriek)', 'een-uit-meer', 'meer-uit-meer', and 'open' are properly supported. There is not yet support exercises of type cloze, in TestVision called 'invul (meervoudig)'. Hence, in case of cloze questions the execution of the function is stopped and a warning is issued.

The TestVision XML may contain links to media content such as data files and images. In the zip-file that TestVision produces such files are typically stored in the directory 'mediafiles'; the function assumes that the TestVision XML file and this directory (and its subdirectories with full content) are unzipped in the same directory. If media files cannot be found a warning is issued.

Since TestVision uses a single XML file for each question, it may be cumbersome to run the function for each question separately, and it advised to use iteration to convert questions in batch.

Value

A list of character vectors containing the R/exams exercise code. If dir is specified (default), this character vector is saved in a file (using [writeLines](#)). In this case the list is returned invisibly. If dir = NULL no files are saved and the list is returned visibly.

See Also

[exams2testvision](#)

Examples

```

if(requireNamespace("xml2")) {
  ## path to a TestVision XML file (provided in the package)
  testvision_xml <- file.path(find.package("exams"), "xml", "testvision_question.xml")

  ## create a temporary directory for R/exams exercise files
  dir.create(tdir <- tempfile())

  ## convert all exercises from the TestVision XML to R/Markdown files
  ex_converted <- testvision2exams(testvision_xml, dir = tdir)
  print(dir(tdir))

  ## additionally the source code of the Rmd file is also return invisible
  ## in 'ex_converted' and can be inspected manually, e.g., via writeLines()
  names(ex_converted)
  writeLines(ex_converted[[1]])

  ## clean up temporary directory
  unlink(tdir)
}

```

 tex2image

Transforming LaTeX Code Using ImageMagick or pdf2svg

Description

Transformation of LaTeX code into an image by compiling to PDF and then transforming to PNG (by default) via ImageMagick's convert command or to SVG via pdf2svg.

Usage

```

tex2image(tex, format = "png", width = NULL, pt = 12, density = 350,
  dir = NULL, tdir = NULL, idir = NULL,
  width.border = 0L, col.border = "white", resize = 650,
  packages = c("amsmath", "amssymb", "amsfonts"),
  header, header2 = NULL, tikz = NULL, Sweave = TRUE, show = FALSE,
  name = "tex2image")

```

Arguments

tex	character vector or list of character vectors. Each character vector is either the name of a LaTeX file or a vector containing LaTeX code directly.
format	character. Suffix for the type of graphic to convert to.
width	numeric. Width of the text in inch. If NULL (or 0), the width is chosen to fit the image in tex.
pt	numeric. Pointsize of the text.
density	numeric. Resolution density of the image.

<code>dir</code>	character specifying the output directory.
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via tempfile .
<code>idir</code>	character specifying the path additional LaTeX inputs required.
<code>width.border</code>	numeric. Width of the framebox border.
<code>col.border</code>	character. Color of framebox border.
<code>resize</code>	numeric. Number of pixels for resizing the image.
<code>packages</code>	character. Names of LaTeX packages to be included.
<code>header</code>	character. LaTeX code to be included in the header of the LaTeX file before the beginning of the document. By default the parindent is set to 0 and sans serif fonts (<code>phv</code>) are used for both text and math.
<code>header2</code>	character. LaTeX code to be included in the header of the LaTeX file after the beginning of the document.
<code>tikz</code>	character. Options to be passed to <code>\usetikzlibrary{}</code> . If set, the <code>tikz</code> package is loaded per default.
<code>Sweave</code>	logical. Should the LaTeX package <code>Sweave.sty</code> be included in the header?
<code>show</code>	logical. Show the resulting image(s) using browseURL .
<code>name</code>	character. Base name of the image file.

Details

`tex2image` converts LaTeX code to image files, e.g., for inclusion in web pages. It proceeds in the following steps: (1) LaTeX code is embedded into a suitable `.tex` file. (2) This is compiled to PDF using [texi2dvi](#). (3) The PDF is converted to an image file. By default, conversion is to PNG using R package **magick** functionalities or alternatively to SVG via `pdfcrop` followed by `pdf2svg`.

The LaTeX code is fit into the standalone document class using the `tikzpicture` environment as a default.

If `tex` is a list of LaTeX chunks, then these are compiled to separate pages of a single PDF in a single LaTeX run. Each page is subsequently converted to a separate image.

In case of SVG output, the respective image manipulation tools, i.e., `pdfcrop/pdf2svg`, are assumed to be installed and available in the search path.

Value

Character vector with path(s) to image(s) generated from the LaTeX code.

See Also

[texi2dvi](#)

Examples

```
## Not run:
## some simple LaTeX
tex <- c("This is \\textbf{bold} and this \\textit{italic}.",
        "Points on the unit circle:  $x^2 + y^2 = 1$ $.")
```

```

## default settings: PNG with sans serif fonts
tex2image(tex, show = interactive())

## with fixed widths
tex2image(tex, width = 6, show = interactive())
tex2image(tex, width = 2, show = interactive())

## switch off header (-> LaTeX uses its standard serif fonts)
tex2image(tex, header = NULL, show = interactive())

## SVG output (system requirements: pdfcrop & pdf2svg)
tex2image(tex, format = "svg", show = TRUE)

## End(Not run)

```

xexams

Extensible Generation of Exams

Description

Extensible automatic generation of exams including multiple choice questions and arithmetic problems.

Usage

```

xexams(file, n = 1L, nsamp = NULL,
  driver = list(sweave = NULL, read = NULL, transform = NULL, write = NULL),
  dir = ".", edir = NULL, tdir = NULL, sdir = NULL, verbose = FALSE,
  points = NULL, seed = NULL, rds = FALSE, ...)

exams_metainfo(x, class = "exams_metainfo", tags = TRUE, factors = FALSE,
  ...)

```

Arguments

file	character. A specification of a (list of) exercise files, for details see below.
n	integer. The number of copies to be taken from file.
nsamp	integer. The number(s) of exercise files sampled from each list element of file. Sampling without replacement is used if possible. (Only if some element of nsamp is larger than the length of the corresponding element in file, sampling with replacement is used.)
driver	list with elements sweave (weaver function or list of arguments for the default xweave), read (function for reading exercise files, defaulting to read_exercise), transform (function to transform each exercise, by default no transformations are done), write (function to write exams to output files, by default nothing is written). For more details, see below.

<code>dir</code>	character. The output directory passed on to <code>driver\$write</code> .
<code>edir</code>	character specifying the path of the directory (along with its sub-directories) in which the files in <code>file</code> are stored (see also below).
<code>tdir</code>	character specifying a temporary directory, by default this is chosen via <code>tempfile</code> . Note that this is cleaned up (i.e., existing files are deleted) and only certain temporary files are preserved.
<code>sdir</code>	character specifying a directory for storing supplements, by default this is chosen via <code>tempfile</code> .
<code>verbose</code>	logical. Should information on progress of exam generation be reported?
<code>points</code>	integer. How many points should be assigned to each exercise? Note that this argument overrules any exercise points that are provided within the <code>expoints</code> tags of the exercise files (if any). The vector of points supplied should either have length 1 or the number of exercises in the exam.
<code>seed</code>	integer matrix or logical. Either NULL (default), logical, or a matrix of random seeds for each possible exercise to be set prior to calling <code>driver@sweave</code> . If NULL no random seeds are set. If a matrix, the number of rows must be <code>n</code> and the number of columns must correspond to <code>unlist(file)</code> . If TRUE a suitable matrix of seeds is sampled.
<code>rds</code>	logical or character indicating whether the list returned by <code>xexams</code> should also be saved as an RDS data file. If <code>rds = TRUE</code> the file name "metainfo.rds" is used. Alternatively, <code>rds</code> can also be a character file name to be used.
<code>x</code>	a list as returned by <code>xexams</code> (or one of the <code>exams2xyz</code> interfaces).
<code>class</code>	character. Should the meta-information be returned as a list of lists with class "exams_metainfo" or as a "data.frame"?
<code>tags</code>	logical. Should the <code>extags</code> entries be processed into separate columns if <code>class = "data.frame"</code> ?
<code>factors</code>	logical. Should the character columns for <code>class = "data.frame"</code> be turned into factors?
<code>...</code>	currently not used.

Details

`xexams` is meant to provide an extensible framework for generating exams based on exercises in R/LaTeX format (via `Sweave`) or R/Markdown format (via `knit`) and rendering them into various output formats such as PDF, HTML, or XML (e.g., for Moodle or IMS QTI). `xexams` is typically not called by the user directly but is used as a common infrastructure for functions such as `exams2pdf`, `exams2html`, `exams2moodle`, `exams2qti12`, or `exams2lops`.

`xexams` generates exams from lists (or vectors) of Rnw/Rmd source files by: (1) running `driver$sweave` on each exercise (by default `xweave` is used, calling `Sweave` or `knit`), (2) running `driver$read` on the resulting LaTeX/Markdown file which by default uses `read_exercise` to read question/solution texts plus meta-information and stores the result in a list, (3) running `driver$transform` on this list for possible transformations (e.g., from LaTeX to HTML), (4) running `driver$write` on the list of exercises within each exam.

Each exercise in an exam is essentially a standalone source file that xexams knows (almost) nothing about, it just calls `driver$sweave` in each iteration and assumes that `driver$read` can read the resulting LaTeX or Markdown file into a list.

The specification in `file` should be either of form `"foo.Rnw"` (or equivalently just `"foo"`) or `"foo.Rmd"`, where the file should either be in the local directory, the `edir` directory or in the `exercises` directory of the package. If `edir` is specified, the directory along with all its sub-directories is searched for the exercises in `file`. Also, `file` can either be a simple vector or a list of vectors. In the latter case, exercises are chosen randomly within each list element. For example, the specification `file = list(c("a", "b"), "xyz")` will result in an exam with two exercises: the first exercise is chosen randomly between `"a"` and `"b"` while `"xyz"` is always included as the second exercise.

Value

A list of exams (of length `n`), each of which is a list of exercises (whose length depends on the length of `file` and `nsamp`), each of which is a list (whose length/contents depends on `driver$read`).

When using the default reader, the resulting list can be simplified using `exams_metainfo`, returning the same (classed) structure as the older `exams` interface. It is recommended to use this to inspect whether the `'extype'` and `'exsolution'` (and corresponding tolerance, if any) are correctly specified.

References

Zeileis A, Umlauf N, Leisch F (2014). Flexible Generation of E-Learning Exams in R: Moodle Quizzes, OLAT Assessments, and Beyond. *Journal of Statistical Software*, **58**(1), 1–36. doi:10.18637/jss.v058.i01.

See Also

[xweave](#), [exams2pdf](#), [exams2html](#), [exams2moodle](#), [exams2canvas](#), [exams2openolat](#), [exams2nops](#)

Examples

```
## define an exam with five exercises
myexam <- list(
  "boxplots.Rmd",
  c("tstat.Rmd", "ttest.Rmd", "confint.Rmd"),
  c("regression.Rmd", "anova.Rmd"),
  "scatterplot.Rmd",
  "relfreq.Rmd"
)

## run exams with default drivers (i.e., no transformations or writer)
x <- xexams(myexam, n = 2)
## x is a list of 2 exams,
## each of which contains 5 exercises,
## each of which contains LaTeX code for question(list) and solution(list),
## plus metainformation and potential supplements

## The first exercise in each exam is "boxplots.Rmd", a multiple choice question.
## Its general question text is
```

```

x[[1]][[1]]$question
## with a list of multiple choice questions given as
x[[1]][[1]]$questionlist
## the corresponding graphic is in supplementary file
x[[1]][[1]]$supplements

## The metainformation is a list read from the ex* items
x[[1]][[1]]$metainfo

## The metainformation can also be extracted/printed
exams_metainfo(x)
## customize printing: only exam 1 in blocks of up to 3 exercises
print(exams_metainfo(x), which = 1, block = 3)

## The metainformation can also be prepared as a data.frame
exams_metainfo(x, class = "data.frame")

```

xweave

Wrapper Function for Weaving Either Rnw or Rmd Exercises

Description

Simple wrapper function that either calls [Sweave](#) for Rnw exercises or [knit](#) for Rmd exercises.

Usage

```

xweave(file, quiet = TRUE, encoding = "UTF-8", engine = NULL, envir = new.env(),
       pdf = TRUE, png = FALSE, svg = FALSE, height = 6, width = 6, resolution = 100,
       highlight = FALSE, ...)

```

Arguments

file, quiet	arguments passed to Sweave or knit , respectively.
encoding	character, ignored. The encoding is always assumed to be UTF-8.
engine	character indicating whether "Sweave" (default) or "knitr" should be used for rendering Rnw exercises.
envir	argument passed to knit . By default, or if <code>envir = NULL</code> a <code>new.env()</code> is created for each <code>xweave()</code> call.
pdf, png, svg, height, width, resolution, highlight, ...	arguments passed to Sweave or opts_chunk , respectively. In the latter case: pdf/png/svg are mapped to <code>dev</code> ; height/width are mapped to <code>fig.height/fig.width</code> ; and resolution is mapped to <code>dpi</code> . <code>highlight</code> is ignored for Sweave .

Details

Depending on whether file has an `.Rnw` or `.Rmd` suffix, either [Sweave](#) or [knit](#) is called for weaving the file by default. `Rnw` exercises can optionally also be weaved by [knit](#) by setting `engine = "knitr"`.

If `png = TRUE` or `svg = TRUE` when calling `Sweave`, then the resulting `includegraphics` statements are supplemented with the `.png` or `.svg` suffix of the corresponding graphics. For `svg` a simple graphics device hook `.xweave_svg_grdevice` is provided on-the-fly for plug-in into `Sweave`.

See Also

[Sweave](#), [knit](#)

Index

* utilities

- exams, 3
- exams2arsnova, 6
- exams2blackboard, 8
- exams2canvas, 12
- exams2evaexam, 14
- exams2grasple, 16
- exams2html, 18
- exams2ilias, 21
- exams2inspera, 24
- exams2kahoot, 26
- exams2lops, 27
- exams2moodle, 29
- exams2nops, 34
- exams2openolat, 38
- exams2pandoc, 41
- exams2participify, 43
- exams2pdf, 45
- exams2qti12, 48
- exams2qti21, 53
- exams2tcexam, 59
- exams2testvision, 60
- exams_eval, 64
- exams_skeleton, 68
- expar, 70
- fmt, 71
- include_supplement, 73
- include_tikz, 74
- make_exercise_transform_pandoc, 75
- match_exams_call, 77
- matrix_to_schoice, 78
- mchoice2string, 80
- moodle2exams, 81
- nops_eval, 83
- nops_fix, 87
- nops_language, 89
- nops_scan, 90
- num_to_schoice, 93
- read_exercise, 94
- stresstest_exercise, 97
- testvision2exams, 99
- tex2image, 101
- xexams, 103
- xweave, 106

answerlist (mchoice2string), 80

browseURL, 20, 102

char_with_braces (fmt), 71

curlPerform, 7

det2schoice (matrix_to_schoice), 78

det_to_schoice (matrix_to_schoice), 78

exams, 3, 47, 80, 96, 105

exams.skeleton (exams_skeleton), 68

exams2arsnova, 6, 69

exams2blackboard, 8, 51, 57, 66

exams2canvas, 12, 50, 51, 57, 105

exams2evaexam, 14

exams2grasple, 16

exams2html, 11, 18, 28, 32, 51, 57, 63, 69, 98, 104, 105

exams2ilias, 21, 50, 51, 57

exams2inspera, 24

exams2kahoot, 26

exams2lops, 27, 104

exams2moodle, 29, 66, 69, 82, 104, 105

exams2nops, 34, 69, 83, 84, 88–90, 92, 105

exams2openolat, 38, 50, 51, 56, 57, 105

exams2pandoc, 41

exams2participify, 6, 7, 43

exams2pdf, 3–5, 36, 45, 69, 104, 105

exams2qti (exams2qti12), 48

exams2qti12, 11–14, 23, 24, 39, 40, 48, 57, 64, 66, 69, 104

exams2qti21, 25, 39, 40, 51, 53, 63, 69

exams2tcexam, 59

exams2testvision, 51, 57, 60, 100

- exams_eval, [10](#), [23](#), [31](#), [32](#), [50](#), [56](#), [62](#), [64](#), [83](#)
- exams_metainfo (xexams), [103](#)
- exams_skeleton, [68](#)
- expar, [70](#)
- extract_command (read_exercise), [94](#)
- extract_environment (read_exercise), [94](#)
- extract_extra (read_exercise), [94](#)
- extract_items (read_exercise), [94](#)

- fmt, [71](#)
- format, [71](#)

- include_supplement, [73](#)
- include_tikz, [74](#)

- knit, [6](#), [9](#), [16](#), [19](#), [28](#), [30](#), [42](#), [50](#), [56](#), [61](#), [71](#), [94](#), [104](#), [106](#), [107](#)

- latexmk, [46](#)

- make_exams_write_arsnova (exams2arsnova), [6](#)
- make_exams_write_grasple (exams2grasple), [16](#)
- make_exams_write_html (exams2html), [18](#)
- make_exams_write_lops (exams2lops), [27](#)
- make_exams_write_pdf (exams2pdf), [45](#)
- make_exercise_transform_html, [13](#), [16](#), [17](#), [25](#), [29](#), [34](#), [39](#), [52](#), [58](#), [76](#)
- make_exercise_transform_html (exams2html), [18](#)
- make_exercise_transform_pandoc, [7](#), [15](#), [43](#), [44](#), [75](#)
- make_itembody_blackboard (exams2blackboard), [8](#)
- make_itembody_qti (exams2qti12), [48](#)
- make_itembody_qti12 (exams2qti12), [48](#)
- make_itembody_qti21 (exams2qti21), [53](#)
- make_itembody_testvision (exams2testvision), [60](#)
- make_nops_template (exams2nops), [34](#)
- make_question_moodle (exams2moodle), [29](#)
- make_question_moodle23 (exams2moodle), [29](#)
- match_exams_call, [77](#)
- match_exams_device (match_exams_call), [77](#)
- match_exams_iteration (match_exams_call), [77](#)

- matrix2mchoice (matrix_to_schoice), [78](#)
- matrix2schoice (matrix_to_schoice), [78](#)
- matrix_to_mchoice (matrix_to_schoice), [78](#)
- matrix_to_schoice, [78](#), [94](#)
- mchoice2string, [5](#), [80](#)
- mchoice2text (mchoice2string), [80](#)
- mclapply, [91](#)
- moodle2exams, [81](#)

- nops_eval, [36](#), [37](#), [83](#), [89](#), [90](#), [92](#)
- nops_eval_write (nops_eval), [83](#)
- nops_fix, [87](#)
- nops_language, [89](#)
- nops_scan, [36](#), [83](#), [84](#), [87–89](#), [90](#)
- num2schoice (num_to_schoice), [93](#)
- num2tol (fmt), [71](#)
- num_to_schoice, [79](#), [93](#), [98](#)
- num_to_tol (fmt), [71](#)

- openolat_config (exams2openolat), [38](#)
- opts_chunk, [106](#)

- pandoc_convert, [20](#), [26](#), [42](#), [43](#), [75](#), [76](#), [82](#), [100](#)
- parLapply, [91](#)
- plot.stress (stresstest_exercise), [97](#)
- print.exams_metainfo (xexams), [103](#)

- read_exercise, [57](#), [94](#), [103](#), [104](#)
- read_metainfo (read_exercise), [94](#)
- readLines, [81](#)
- readPNG, [92](#)
- round2 (fmt), [71](#)
- runif, [93](#)

- spineplot, [97](#)
- stresstest (stresstest_exercise), [97](#)
- stresstest_exercise, [97](#)
- string2mchoice (mchoice2string), [80](#)
- Sweave, [3–5](#), [71](#), [94](#), [104](#), [106](#), [107](#)

- tempfile, [3](#), [9](#), [19](#), [22](#), [27](#), [28](#), [30](#), [42](#), [46](#), [49](#), [54](#), [61](#), [102](#), [104](#)
- testvision2exams, [99](#)
- tex2image, [20](#), [29](#), [34](#), [52](#), [58](#), [74](#), [101](#)
- texi2dvi, [3–6](#), [15](#), [16](#), [44](#), [46](#), [47](#), [102](#)
- toJSON, [7](#), [17](#)
- toLatex.data.frame (fmt), [71](#)
- toLatex.matrix (fmt), [71](#)

tth, *20, 29, 34, 52, 56, 58, 62*

ttm, *20, 29, 34, 52, 58*

write.table, *15*

write.xlsx, *26*

writelnLines, *82, 100*

xexams, *7, 9, 11, 14, 15, 17, 19, 20, 22, 24–30, 32–34, 37, 40, 42–44, 46, 47, 49, 52, 54, 58–61, 63, 76, 96, 98, 103*

xweave, *6, 7, 9, 11, 15–17, 19, 20, 22, 26–28, 30, 32, 42, 44, 46, 49–51, 54, 56, 57, 59, 61, 63, 103–105, 106*