AcroTeX.Net

The ran_toks Package

Randomizing the order of tokens

D. P. Story

Prepared: June 6, 2021

www.acrotex.net
Version 1.4, 2021/06/06

The ran_toks Package

Table of Contents

1. Introduction

This is a short package for randomizing the order of tokens. The package is long overdue; users of **AeB** and of eqexam have long asked for a way to randomize the order of the problems in a test or quiz, or anything for that matter.

The examples folder contains nine demonstration files:

1. ran-toks.tex reproduces the sample code of this manual.
2. rt-tst-eqe.tex shows how to use ran_toks to randomize the *questions* of an exam document created by the eqexam package.
3. rt-tst-qz.tex shows how to randomize choices of a multiple choice field in a quiz environment of the exerquiz package, when the choices contain verbatim text.
4. rt-cb.tex is a demonstration of how to use the collectbox package to place verbatim content into a token list (see \ranToks, Section 3.1).

The next five files concern the DB application and require the usedbapp option, they are discussed in Section 5.

5. mc-db.tex is an eqexam file, draws from the database files db1.tex, db2.tex, db3.tex, and db4.tex, to construct the questions of the exam. The questions are drawn at random from the DB files. Refer to Section 5 for a few more details.
6. mc-dbu.tex is an eqexam file that draws from the database test file, draws from the database files db1.tex, db2.tex, db3.tex, and db4.tex, to construct the questions of the exam. However, mc-dbu.tex differs from mc-db.tex for it selects unique questions cross the four versions of the document. Refer to Section 5 for a few more details.
7. viewDB.tex A demo file of how to view the DB files for the purpose of reviewing the questions, modifying them, or adding to them.
8. mc-dbu-ctrld.tex is a variation on mc-dbu.tex, it is not meant to be compiled directly, but by the file ctrl-build.tex.
9. ctrl-build.tex is a simple TEX file that controls the compilation of a targeted file (mc-dbu-ctrld.tex in this example) over the various versions supported by the targeted file. Try it, you'll like it. Uses the shellesc package to control the process.

2. The Preamble and Package Options

The preamble for this package is

```
\usepackage[⟨options⟩]{ran_toks}
```

usedbapp option

The one option of ran_toks is usedbapp, which brings in specialized code to support a

database test application. This option is discussed in Section 5, beginning on page 13.

The requirements for ran_toks are the verbatim package (part of the standard LaTeX distribution, and the macro file random.tex by Donald Arseneau.

3. The main commands and environments

There are two styles for defining a series of tokens to be randomized, using either the \ranToks command or the \bRTVToks/\eRTVToks pair. Each of these is discussed in the next two subsections.

3.1. The \ranToks command

The \ranToks command was the original concept; declare a series of tokens to be randomized.

```
\ranToks{⟨name⟩}{%
  {⟨token1⟩}
  {⟨token2⟩}
  ...
  {⟨tokenn⟩}
}
```

were ⟨$token_k$⟩ is any non-verbatim content;[1] each token is enclosed in braces ({}), this is required. The ⟨*name*⟩ parameter is required, and must be unique for the document; it is used to build the names of internal macros. Of course several such \ranToks can be used in the document, either in the preamble or in the body of the document. Multiple \ranToks commands must have a different ⟨*name*⟩ parameter.

After a \ranToks command has been executed, the number of tokens counted is accessible through the \nToksFor command,

```
\nToksFor{⟨name⟩}
```

The one argument is ⟨*name*⟩, and will expand to the total number of tokens listing as argument in the \ranToks command by the same name.

The \ranToks command does not display the randomized tokens, for that the command \useRanTok is used.

```
\useRanTok{⟨num⟩}
\useRTName{⟨name⟩}
```

The argument of \useRanTok is a positive integer between 1 and \nToksFor{⟨*name*⟩}, the number of tokens declared by \ranToks, inclusive. There is no space created following the \useRanTok command, so if these are to be used "inline", enclose them in braces ({}), for example, {\useRanTok{1}}. The use of \useRTName is optional unless the listing of the \useRanTok commands is separated from the \ranToks command

[1] However, for workarounds, see rt-cb.tex and rt-tst-qz.tex.

that defined them by another \ranToks command of a different name. That should be clear!

Consider this example.

```
\ranToks{myPals}{%
    {Jim}{Richard}{Don}
    {Alex}{Tom}{J\"{u}rgen}
}
```

I have 6 pals, they are Jürgen, Richard, Jim, Don, Alex and Tom. (Listed in the order of best friend to least best friend.) The verbatim listing is,

```
I have {\nToksFor{myPals}} pals, they are \useRanTok{1},
\useRanTok{2}, \useRanTok{3}, \useRanTok{4}, {\useRanTok{5}}
and \useRanTok{6}.
```

Notice that \useRanToks are not enclosed in braces for 1–4 because they are each followed by a comma; the fifth token, {\useRanTok{5}}, is enclosed in braces to generate a space following the insertion of the text.

Repeating the sentence yields, "I have 6 pals, they are Jürgen, Richard, Jim, Don, Alex and Tom", which is the exact same random order. To obtain a different order, re-execute the \ranToks command with the same arguments.[2] Doing just that, we obtain, "I have 6 pals, they are Jim, Richard, Don, Jürgen, Tom and Alex." A new order? An alternative to re-executing \ranToks is to use the \reorderRanToks command:

\reorderRanToks{⟨*name*⟩}

Now, executing \reorderRanToks{myPals} and compiling the sentence again yields, "I have 6 pals, they are Jim, Richard, Don, Jürgen, Tom and Alex." For most applications, re-randomizing the same token list in the same document is not very likely something you need to do.

The \reorderRanToks{⟨*name*⟩} command rearranges the list of tokens associated with ⟨*name*⟩, which may not be what you want; the \copyRanToks command, on the other hand, makes a (randomized) copy of its first required argument ⟨$name_1$⟩ and saves it as ⟨$name_2$⟩, without effecting the order of ⟨$name_1$⟩.

\copyRanToks{⟨$name_1$⟩}{⟨$name_2$⟩}

Thus, if \copyRanToks{myPals}{myPals1} is executed, the token list name myPals1 contains the names of my pals in another randomized order, while maintaining the same order of myPals.

My original application for this, the one that motivated writing this package at long last, was the need to arrange several form buttons randomly on the page. My point is that the listing given in the argument of \ranToks can pretty much be anything that is allowed to be an argument of a macro; this would exclude verbatim text created by \verb and verbatim environments.

[2] \ranToks{myPals}{{Jim}{Richard}{Don}{Alex}{Tom}{J\"{u}rgen}} in this example.

3.2. The \bRTVToks/\eRTVToks pair of commands

Sometimes the content to be randomized is quite large or contains verbatim text. For this, it may be more convenient to use the \bRTVToks/\eRTVToks command pair. The syntax is

```
\bRTVToks{⟨name⟩}   % <-Begin token listing
\begin{rtVW}
  ⟨content_1⟩
\end{rtVW}
...
...
\begin{rtVW}
  ⟨content_n⟩
\end{rtVW}
\eRTVToks           % <-End token listing
```

The \bRTVToks{⟨*name*⟩} command begins the (pseudo) environment and is ended by \eRTVToks. Between these two are a series of rtVW (random toks verbatim write) environments. When the document is compiled, the contents (⟨$content_i$⟩) of each of these environments are written to the computer hard drive and saved under a different name (based on the parameter ⟨*name*⟩). Later, using the \useRanTok commands, they are input back into the document in a random order.

\RTVWHook The rtVW environment also writes the command \RTVWHook to the top of the file. Its initial value is \relax. It can be redefined with \rtVWHook{⟨*arg*⟩}, which expands to \def\RTVWHook{⟨*arg*⟩}.

The use of \useRTName and \useRanTok were explained and illustrated in the previous section. Let's go to the examples,

```
\bRTVToks{myThoughts}
\begin{rtVW}
\begin{minipage}[t]{.67\linewidth}
Roses are red and violets are blue,
I've forgotten the rest, have you too?
\end{minipage}
\end{rtVW}
\begin{rtVW}
\begin{minipage}[t]{.67\linewidth}
I gave up saying bad things like
\verb!$#%%%^*%^&#$@#! when I was just a teenager.
\end{minipage}
\end{rtVW}
\begin{rtVW}
\begin{minipage}[t]{.67\linewidth}
I am a good guy, pass it on! The code for this last sentence is,
\begin{verbatim}
%#$% I am a good guy, pass it on! ^&*&^*
```

```
\end{verbatim}
How did that other stuff get in there?
\end{minipage}
\end{rtVW}
\eRTVToks
```

OK, now, let's display these three in random order. Here we place them in an `enumerate` environment.

1. I gave up saying bad things like $#%%%^*%^&#$@# when I was just a teenager.
2. Roses are red and violets are blue, I've forgotten the rest, have you too?
3. I am a good guy, pass it on! The code for this last sentence is,

   ```
   %#$% I am a good guy, pass it on! ^&*&^*
   ```

   How did that other stuff get in there?

The verbatim listing of the example above is

```
\begin{enumerate}
    \item \useRanTok{1}
    \item \useRanTok{2}
    \item \useRanTok{3}
\end{enumerate}
```

The `\reorderRanToks` works for lists created by the `\bRTVToks`/`\bRTVToks` construct. If we say `\reorderRanToks{myThoughts}` and reissue the above list, we obtain,

1. Roses are red and violets are blue, I've forgotten the rest, have you too?
2. I am a good guy, pass it on! The code for this last sentence is,

   ```
   %#$% I am a good guy, pass it on! ^&*&^*
   ```

   How did that other stuff get in there?
3. I gave up saying bad things like $#%%%^*%^&#$@# when I was just a teenager.

The command `\copyRanToks` works for list created by `\bRTVToks`/`\bRTVToks` as well.

On the `\displayListRandomly` command. In the enumerate example immediately above, the items in the list are explicitly listed as `\item \useRanTok{1}` and so on; an alternate approach is to use the command `\displayListRandomly`, like so,

```
\begin{enumerate}
    \displayListRandomly[\item]{myThoughts}
\end{enumerate}
```

The full syntax for `\displayListRandomly` is displayed next.

`\displayListRandomly[⟨prior⟩][⟨post⟩]{⟨name⟩}`

The action of `\displayListRandomly` is to expand all tokens that are listed in the ⟨*name*⟩ token list, each entry is displayed as ⟨*prior*⟩`\useRanTok{i}`⟨*post*⟩, where `i` goes from 1 to `\nToksFor{`⟨*name*⟩`}`. In the example above, ⟨*prior*⟩ is `\item`, but normally, its default is empty. The defaults for ⟨*prior*⟩ and ⟨*post*⟩ are both empty.

The optional arguments. When only one optional argument is present, if is interpreted as ⟨*prior*⟩. To obtain a ⟨*post*⟩ with no ⟨*prior*⟩ use the syntax,

```
\displayListRandomly[][⟨post⟩]{⟨name⟩}
```

Within *each optional argument*, the four commands `\i`, `\first`, `\last`, and `\lessone` are (locally) defined. The `\i` command is the index counter of the token currently being typeset; `\first` is the index of the first item; `\last` is the index of the last item; and `\lessone` is one less than `\last`. The two optional arguments and the four commands may use to perform logic on the token as it is being typeset. For example:

```
List of pals: \displayListRandomly
    [\ifnum\i=\last and \fi]
    [\ifnum\i=\last.\else, \fi]{myPals}
```

yields,

List of pals: Jim, Richard, Don, Jürgen, Tom, and Alex.

The optional arguments are wrapped to the next line to keep them within the margins, cool.

The example above shows the list of my pals with an Oxford comma. How would you modify the optional argument to get the same listing without the Oxford comma? (Jim, Richard, Don, Jürgen, Tom and Alex.) Hint: a solution involves the other command `\lessone`.

• Nested `\bRTVToks`/`\eRTVToks` command pairs

placement

These is at least one example of using nested `\bRTVToks`/`\eRTVToks`. When nested `\bRTVToks`/`\eRTVToks` command pairs, use the `rtVWi` environment instead of the `rtVW` environment. The nested `\bRTVToks`/`\eRTVToks` pair is placed within a `rtWV` environment; in this way the contents of that `rtWV` environment, itself can be randomized. The `\displayListRandomly` (or `\useRanTok`) command is used to list out the nested items. See the next page for an example.

1. Depth0: Item 5
2. Depth0: Item 3
3. Depth0: Item 2:
    - Depth1: Item 3
    - Depth1: Item 2
        - Depth2: Item 3
        - Depth2: Item 2
        - Depth2: Item 1
    - Depth1: Item 1
4. Depth0: Item 4
5. Depth0: Item 1

The verbatim listing for this example follows.

```
\bRTVToks{Depth0}
  \begin{rtVW}
    Depth0: Item 1
\end{rtVW}
  \begin{rtVW}
    Depth0: Item 2:
    \bRTVToks{Depth1} % begin Depth2 within an rtVW env.
      \begin{rtVWi}
        Depth1: Item 1
\end{rtVWi}
      \begin{rtVWi}
        Depth1: Item 2
        \bRTVToks{Depth2} % begin Depth1 within an rtVWi env.
          \begin{rtVWii}
              Depth2: Item 1
\end{rtVWii}
          \begin{rtVWii}
              Depth2: Item 2
\end{rtVWii}
          \begin{rtVWii}
              Depth2: Item 3
\end{rtVWii}
        \eRTVToks
\begin{itemize}
        \displayListRandomly[\item]{Depth2} % display Depth2 toks
\end{itemize}
\end{rtVWi}
      \begin{rtVWi}
        Depth1: Item 3
\end{rtVWi}
\eRTVToks
```

```
\begin{itemize}
\displayListRandomly[\item]{Depth1} % display Depth1 toks
\end{itemize}
\end{rtVW}
  \begin{rtVW}
      Depth0: Item 3
\end{rtVW}
  \begin{rtVW}
      Depth0: Item 4
\end{rtVW}
  \begin{rtVW}
      Depth0: Item 5
\end{rtVW}
\eRTVToks
\begin{enumerate}
\displayListRandomly[\item]{Depth0} % display Depth0 toks
\end{enumerate}
```

Oops. Did I forget to mention that `ran_toks` supports nested to a depth of two.

Some authors like to indent nested things, but to avoid spurious spaces appearing, `\end{rtVW}` (and `\end{rtVWi}`) should be placed in the far left margin, as shown above. Recall the `rtVW` is a verbatim environment.

The above example is reproduced in the `ran_toks.tex` sample file found in the `examples` folder. Also found in that folder is `nested-matching.tex`, a `exerquiz` quiz that motivated creating nested `\bRTVToks`/`\eRTVToks` command pairs.

4. Additional arguments and commands

The syntax given earlier for `\useRanTok` was not completely specified. It is,

`\useRanTok[`⟨*name*⟩`]{`⟨*num*⟩`}`

The optional first parameter specifies the ⟨*name*⟩ of the list from which to draw a random token; ⟨*num*⟩ is the number of the token in the range of 1 and `\nToksFor{`⟨*name*⟩`}`, inclusive. The optional argument is useful in special circumstances when you want to mix two random lists together.

To illustrate: Jim, I am a good guy, pass it on! The code for this last sentence is,

```
%#$% I am a good guy, pass it on! ^&*&^*
```

How did that other stuff get in there?

The verbatim listing is

```
To illustrate: \useRanTok[myPals]{1}, \useRanTok[myThoughts]{2}
```

The typeset version looks a little strange, but recall, the text of `myThoughts` were each put in a `minipage` of width `.67\linewidth`. Without the `minipage`, the text would wrap around normally.

Accessing the original order. The original order of the list of tokens is not lost, you can retrieve them using the command \rtTokByNum,

\rtTokByNum[⟨*name*⟩]{⟨*num*⟩}

This command expands to the token declared in the list named ⟨*name*⟩ that appears at the ⟨*num*⟩ place in the list. (Rather awkwardly written.) For example, my really best pals are Don and Alex, but don't tell them. The listing is,

```
For example, my really best pals are {\rtTokByNum[myPals]{3}}
and \rtTokByNum[myPals]{4}, but don't tell them.
```

In some sense, \rtTokByNum[⟨*name*⟩] acts like a simple array, the length of which is \nToksFor{⟨*name*⟩}, and whose k^{th} element is \rtTokByNum[⟨*name*⟩]{⟨*k*⟩}.

Turning off randomization. The randomization may be turned off using \ranToksOff or turned back on with \ranToksOn.

\ranToksOff \ranToksOn

This can be done globally in the preamble for the whole of the document, or in the body of the document just prior to either \ranToks or \bRTVToks. For example,

```
\ranToksOff
\ranToks{integers}{ {1}{2}{3}{4} }
\ranToksOn
```

As a check, executing '\useRanTok{3} = \rtTokByNum{3} = 3' yields '3 = 3 = 3'? As anticipated.

To create a non-randomized list of tokens that already have been created (and randomized), use \copyRanToks:

```
\ranToksOff\copyRanToks{myPals}{myOriginalPals}\ranToksOn
```

Then, using \displayListRandomly in a clever way,

```
\displayListRandomly[\ifnum\i=\last\space and \fi(\the\i)~]
    [\ifnum\i=\last.\else,\fi\space]{myOriginalPals}
```

we obtain: (1) Jim, (2) Richard, (3) Don, (4) Alex, (5) Tom, and (6) Jürgen. The original list for myPals remains unchanged: (1) Jim, (2) Richard, (3) Don, (4) Jürgen, (5) Tom, and (6) Alex.

The \useRanTok command—whether it operates on a randomized token list or not—behaves similarly to an array. Thus, if we wanted to extract the third entry of the non-randomized token list (array) myOriginalPals, we do so by expanding the command \useRanTok[myOriginalPals]{3} to produce Don.

Document preparation. The command \ranToksOff is probably best in the preamble to turn off all randomization while the rest of the document is being composed.

The ran_toks auxiliary file. The package writes to a file named `\jobname_rt.sav`, below represents two typical lines in this file.

```
1604051353 % initializing seed value
5747283528 % last random number used
```

The first line is the initializing seed value used for the last compilation of the document; the second line is the last value of the pseudo-random number generator used in the document.

Normally, the pseudo-random number generator provided by `random.tex` produces a new initial seed value every minute. So if you recompile again before another minute, you'll get the same initial seed value.

Controlling the initial seed value. To obtain a new initial seed value each time you compile, place `\useLastAsSeed` in the preamble.

`\useLastAsSeed`

When the document is compiled, the initial seed value taken as the second line in the `\jobname_rt.sav` file, as seen in the above example. With this command in the preamble, a new set of random numbers is generated on each compile. If the file `\jobname_rt.sav` does not exist, the generator will be initialized by its usual method, using the time and date.

The command `\useThisSeed` allows you to reproduce a previous pseudo-random sequence.

`\useThisSeed{⟨init_seed_value⟩}`

This command needs to be placed in the preamble. The value of ⟨*init_seed_value*⟩ is an integer, normally taken from the first line of the `\jobname_rt.sav` file.

When creating tests (possibly using `eqexam`), the problems, or contiguous collections of problems, can be randomly ordered using the `\bRTVToks/\eRTVToks` command pair paradigm. For example, suppose there are two classes and you want a random order (some of) the problems for each of the two classes. Proceed as follows:

1. Compile the document, open `\jobname_rt.sav`, and copy the first line (in the above example, that would be `1604051353`).
2. Place `\useThisSeed{1604051353}` in the preamble. Compiling will bring back the same pseudo-random sequence very time.
3. Comment this line out, and repeat the process (use `\useLastAsSeed` to generate new random sequences at each compile) until you get another distinct randomization, open `\jobname_rt.sav`, and copy the first line again, say its `735794511`.
4. Place `\useThisSeed{735794511}` in the preamble.
5. Label each

```
%\useThisSeed{1604051353} % 11:00 class
%\useThisSeed{735794511}  % 12:30 class
```

To reproduce the random sequence for the class, just uncomment the random seed used for that class.

If you are using eqexam, the process can be automated as follows:

```
\vA{\useThisSeed{1604051353}} % 11:00 class
\vB{\useThisSeed{735794511}}  % 12:30 class
```

Again, this goes in the preamble.

5. Commands to support a DB application

The commands described in this section are only available with the usedbapp option,

```
\usepackage[usedbapp]{ran_toks}
```

This option was designed for an eqexam document.

5.1. Basic functionality

The premise here is that you have an eqexam document (a test) and you have a series of standard questions you ask students. Over the years, you have accumulated questions of a similar type that you like to pose to your students. The questions of a similar type are placed in a DB test file. For example, you have a file named db1.tex containing questions on a certain narrow topic. The format for this file is,

```
%
% Questions on some narrow topic
%
\bRTVToks{DB1} %<-DB ⟨name⟩
\begin{rtVW}
% an eqexam question is contained in this rtVW environment
\end{rtVW}
\begin{rtVW}
% an eqexam question is contained in this rtVW environment
\end{rtVW}
...
\eRTVToks
```

Refer to the file db1.tex, db2.tex,..., db4.tex for more specific examples. The DB ⟨*name*⟩ must be unique among all the DB test files used.

The next step is to input your DB files. To do this, execute either of the commands \useTheseDBs or \useProbDBs prior to the opening of an exam environment (eqexam), or perhaps in the preamble. The syntax is,

```
\useTheseDBs{⟨db1⟩,⟨db2⟩,...,⟨dbn⟩}
\useProbDBs{⟨db1⟩,⟨db2⟩,...,⟨dbn⟩}
```
(An alias for \useTheseDBs)

The argument of \useTheseDBs is a comma-delimited list of file names. Each file name contains a \bRTVToks/\eRTVToks construct. Within this pair are rtVW environments, as described in Section 3.2. The \useTheseDBs command inputs the files listed in its comma-delimited argument; a warning is emitted if one or more of the files are not found. The default extension is .tex, \useTheseDBs{db1,db2} inputs the files db1.tex and db2.tex, if they exist, while \useTheseDBs{db1.def,db2.db} inputs the files db1.def and db2.db, if they exist. The command \useProbDBs is an alias for \useTheseDBs.

Once the database files have been input, in the body of the document you can extract questions using \useRanTok; for example, \useRanTok[DB1]{1} extracts the question with an index of 1 from the database file DB1;[3] \useRanTok[DB2]{2} extracts the question with an index of 2 from the database file DB2, and so on.

Refer to the demonstration file mc-db.tex for an example.

5.2. Striving for uniqueness of choices

By default, when you expand \useRanTok{1} and later you expand \useRanTok{1} again you get the exact same result;[4] this is the normal behavior. However, in the context of posing questions from a database of questions, repeating the questions is not what is wanted necessarily. In the next two subsections, we speak to the problem, first within a single-version document, and secondly, within a multi-version document.

• For documents with a single version

The issue of not wanting to repeat a question from a database comes up when you want to pose several questions from a given database file. We illustrate using the demo file mc-db.tex found in the examples folder. In that file, the exam has two parts, as shown in Figure 1.[5] For Part1, we take two problems (at random) from DB1. In Part2, we take

```
\begin{Part1}
\useRanTok[DB1]{1}
\useRanTok[DB1]{2}
\end{Part1}

%  First approach          % An alternate approach
\begin{Part2}              % \begin{Part2}
\useRanTok[DB1]{1}         % \useRanTok[DB1]{3}
\useRanTok[DB1]{2}         % \useRanTok[DB1]{4}
\end{Part2}                % \end{Part2}
```

Figure 1: Simplified two-part exam

two problems again from DB1; these two will be the *same* as the ones chosen from Part1. To get two *different* questions the natural approach is to write \useRanTok[DB1]{3}

[3] It is assumed that DB1 is the ⟨*name*⟩ is the required argument of \bRTVToks, which is declared in the db1.tex file.

[4] Here, we are assuming the ⟨*name*⟩ of the token list is the same for both expansions of \useRanTok{1}.

[5] The verbatim listing here has been simplified.

and \useRanTok[DB1]{4}, as shown on the right in Figure 1. Now, returning to the First approach, there is a way of forcing ran_toks to choose two different question even though the indexes used are the same.

To have different questions appear when you specify \useRanTok[DB1]{1} and \useRanTok[DB1]{2}, first expand the command \uniqueXDBChoicesOn, perhaps between parts or in the preamble. When \uniqueXDBChoicesOn is expanded, \useRanTok tries to find an "unused" choice.

```
\uniqueXDBChoicesOn
\begin{document}
...
\begin{Part1}
\useRanTok[DB1]{1}
\useRanTok[DB1]{2}
\end{Part1}

%\uniqueXDBChoicesOn

\begin{Part2}
\useRanTok[DB1]{1}
\useRanTok[DB1]{2}
\end{Part2}
```

The command \uniqueXDBChoicesOff turns off the feature of striving to find "unused" choices. Additional discussion on \uniqueXDBChoicesOn is found in the next section; the command \InputUsedIDs, discussed in the next section, is not needed when the eqexam document *does not have* multiple versions. You can experiment with striving to find unused questions in the demo file mc-db.tex.

• For documents with multiple versions

The scheme outlined in Section 5.1 works well for an eqexam document that only has one version in the source file. The way ran_toks works, it will not repeat random choices — unless you sample from a same DB file more times than there are problems in that file; for example, suppose db1.tex has two questions in it, if you execute \useRanTok[DB1]{1}, \useRanTok[DB1]{2}, and \useRanTok[DB1]{3}, then the problem selected by \useRanTok[DB1]{3} is the same as \useRanTok[DB1]{1}. This latter situation is not likely, is it?

One of the very powerful features of eqexam is that a single source file can have multiple versions in it.

```
\examNum{1}
\numVersions{4}
\forVersion{a} % a, b, c, d
\vA{\useThisSeed{54356}}
\vB{\useThisSeed{577867}}
\vC{\useThisSeed{6746788}}
\vD{\useThisSeed{856785}}
```

The above shows how to set up a multi-version eqexam document, see the eqexam documentation for more details. Here, as in the demo file mc-dbu.tex, we declare 4 versions (a, b, c, and d). \forVersion{a} declares the next compile is for version a (or A, either one). List also is a method of passing an initial seed to the pseudo-random number generator, for each version. (These can be arbitrarily typed in, or obtained by the methods discussed in **Controlling the initial seed value** on page 12.)

The problem. For a multi-version eqexam document, where each version samples from the same set of DB Test files, later versions of the exam may have questions that are repeats of the ones posed in earlier versions of the same source file. This may or may not be a problem if the different versions are given to different classes at approximately the same time where there is no opportunity for the details of the test to "leak out" from one class to another.

The solution. The solution to this problem requires the introduction of several new commands.

```
\uniqueXDBChoicesOn   % there is also \uniqueXDBChoicesOff (the default)
\InputUsedIDs         % preamble only (required)
%\viewIDstrue         % to view the IDs of problems used
```

The \uniqueXDBChoicesOn command brings some special code into the \useRanTok command; this special code tries to select problems that have not already been selected by earlier versions of the source document. Key to this selection process is that the special code needs to know which questions were earlier selected; that is the role of the \InputUsedIDs. This latter command uses the value declared in \numVersions, so it must appear *after* the declaration of the number of versions.

When \uniqueXDBChoicesOn is expanded, as the source document is compiled, an auxiliary file named \jobname-ver⟨*Ltr*⟩.cut is written. This file keeps a record of the problem ID of the problems selected, believe it or not. \InputUsedIDs inputs the appropriate CUT files:

- for version B, \jobname-verA.cut is input;
- for version C, \jobname-verA.cut, \jobname-verB.cut are input;
- for version D, \jobname-verA.cut, \jobname-verB.cut, \jobname-verC.cut are input;
- and so on.

In this way, when you compile version ⟨*Ltr*⟩, the document inputs all the information it needs about previous versions to make in informed choice.

Workflow. When compiling a multi-version eqexam document, do the following:

1. Build each version in *alphabetical order*, that is, compile with \forVersion{a}, then \forVersion{b}, and so on.
2. Rename PDF produced to reflect the version ⟨*Ltr*⟩; eg, \jobname-verA.pdf or \jobname-sec02.pdf, or whatever.[6]

[6] The demo file ctrl-build.tex shows how to build all versions and rename the final PDF files all from

Now, I simply must get back to my retirement. ☺

one controlling file.