ktv-texdata package

Kỳ Anh
(kyanh@inic.biz, kyanh@linuxmail.org)

package: version 05.34, last update 2003/10/06
documentation: version 1544, last update 2003/11/20

Abstract

This package provides a simple way to use the TEX input files whose contents are in the numbered environments.

This package is useful for **the teachers of mathematics**, who often work with large libraries of mathematical exercises.

Contents

1 How will you do?

Assume that you have an input file (named '`foo.tex`') that specifies 16 exercies

```
% --- first line of 'foo.tex'
\begin{exercice}\label{ex:1}
      This is the first exercice.
\end{exercice}

\begin{exercice}\label{ex:2}
      This is the second exercice.
\end{exercice}

...
\begin{exercice}[*]\label{ex:16}
      This is the 16th exercice
      (with a star mark *).
\end{exercice}
% --- last line of 'foo.tex'
```

On Tuesday, for e.g., you want to create a student test that contains the first 8 exercises of the '`foo.tex`'. However, on Wednesday, you want to create another test that contains the last 8 exercises of the '`foo.tex`'.

Of course, the simplest way to do that is *copying* and *pasting*. Of course, this way becomes too complex in case, for e.g., you need only the exercies that are numbered oddly (1, 3, 5, 7, 9, 11, 13, 15).

You may think of a solution like this

```
\getonly{1,2,3,4,5,6,7,8}        % on Tuesday
\getonly{9,10,11,12,13,14,15,16} % on Wednesday
\getonly{1,3,5,7,9,11,13,15}     % on Friday
```

Yes, here we go....

2 Introduction

2.1 Convention

Instead of using `#1`, `#2`, etc., to specify the first, the second,... parameter or argument of a macro, we will use `#foo`, `#xfoo`,... where `#foo` is a short description of `#1`, and `#xfoo` is a short description of `#2`, etc.

Parameter is something declared in the macro's definitons.

Argument is something you pass to a macro when you call it.

The optional parameters are enclosed in the brackets ('`[`' and '`]`').

2.2 Data item 'bxx'

2.2.1 Form

Each '`bxx`', or a *data item*, is of the form

```
\bxx(#env)[#thm]ID;
something to typeset
\exx
```

Note that '`\exx`' must be located at *the beginning of a new line* and the semicolon '`;`' is mandatory.

`ID` `ID` is any non-empty string of characters, numbers or punctuations:

```
      ID  ⟶ char-num / xpunct / ID ID
char-num  ⟶ a..z / A..Z / 0..9
  xpunct  ⟶ : / . / ,
```

For example, '`ex:1`', '`Ex:2003`' are the `ID`s, but '`ex;1`' isnot.

`#env` `#env` is any predefined environment. Parameter `(#env)` is *optional*.

`#thm` `#thm` is the *optional argument* for the environment `#env`. It is also the *optional parameter* for the macro `\bxx`, because it is enclosed in the brackets; so you can specify a '`bxx`' like this

```
\bxx(#env)ID;
something to typeset
\exx
```

or more simply (because `(#env)` is optional)

```
\bxx ID;
something to typeset
\exx
```

2.2.2 Meaning

\bxx What does '`bxx`' mean?
\exx Yes, it's very familar with `\begin{#env}` and `\end{#env}`.

`\bxx(#env)[#thm]ID;`	*means*	`\begin{#env}[#thm]`
something to typeset	*means*	*itself*
`\exx`	*means*	`\end{#env}`

2.2.3 Identifying the 'bxx'

We use `ID` and `#env` to identify a '`bxx`' by associating that '`bxx`' with the string `#envID`. This string is named *string-id* of the '`bxx`'. Then two '`bxx`'s are different if their *string-id*s are different. For e.g., after the declaration

```
\bxx(exercise)100;
...
\exx
```

the *string-id* of the '`bxx`' is '`exercise100`'.

2.3 Data file. Example

An input file containing one or more '`bxx`' is called a *data fille*. Using the '`bxx`', we reedit the 'foo.tex' in the first section. Then, '`foo.tex`' becomes a *data file*.

```
% --- first line of the new 'foo.tex'
\bxx(exercice)ex:1;
\label{ex:1}
      This is the first exercice.
\exx
\bxx(exercice)ex:2; \label{ex:2}
      This is the second exercice.
\exx
...
\bxx(exercice)ex:16;\label{ex:16}
      This is the 16th exercice.
\exx
% --- last line of the new 'foo.tex'
```

3 User's macros

3.1 Turning on/off the detail(s)

`\xdetailon` `\xdetailoff`

Syntax

```
\xdetailon
\xdetailoff
```

By default, the package shows *string-id* of the '`bxx`' you want to get on the margin. You can turn this feature on/off by these marcos. The macros can be put anywhere in your document. See the package's test for an illustration.

You can also pass an option to the package, like this

`\usepackage[detailon]{ktv-texdata}`, or
`\usepackage[detailoff]{ktv-texdata}`

3.2 Specifying the default environment

`\xenv`

Syntax

```
\xenv(#env)
```

This macro specifies the default environment used for the macros '`bxx`', '`\xget`', '`\xgetall`', etc. Here `#env` is any predefined environment.

This macro can be put anywhere. It keeps the effect until the next `\xenv`.

3.3 Getting/Ignoring the 'bxx'

There are 7 macros used to get/remove the 'bxx' from the data file.

```
\xget  \xgetall  \xgetallbut
\xkill \xkillall \xkillallbut
       \xspec
```

The macro `\xspec` is described in subsection 3.6.

`\xgetall` `\xkillall`

Syntax

```
\xgetall
\xkillall
```

`\xgetall` means that you want to *get all the data items from the data file*, while `\xkillall` means that you want to *ignore all....* These macros remove the effects of any `\xget`, `\xkill` that is called before them.

`\xget`

Syntax

```
\xget(#env){#ID-list}
```

where

- `#env` is an environment name. The parameter `(#env)` is optional.

- `#ID-list` is a list of `ID`s that're seperated by the comma (‘`,`’), each `ID` can be prefixed by a plus sign (‘`+`’) or by a minus sign (‘`-`’).

For e.g., the specification

```
\xget(exercise){-12,-3,+5,6,-99,+100}
```

means that you want to get from the data file all the ‘`bxx`’s whose *string-id*s are

`exercise5`, `exercise6`, `exercise100`

Because you specified `-12`, `-3`, `-99`, all the ‘`bxx`’ whose *string-id*s are `exercise12`, `exercise3`, `exercise99` are ignored. Of course, `exercise7` is ignored, too.

`Trick` :: **TRICK**

The parameter `(#env)` is optional; so if you specified

```
\xenv(exercise)
```

then you can write shortly

```
\xget{-12,-3,+5,6,-99,+100}
```

`Trick` :: **TRICK**

A sequence of two `\xget`s can be replaced by one `\xget`, for e.g.,

`\xget{1,6} \xget{3,-4}` ⟶ `\xget{1,6,3,-4}`

`\xkill` The syntax and tricks are the same as `\xget`. Try to guess the usage of this macro.

`\xgetallbut` The syntax is the same as `\xget`.

`\xkillallbut` The macro `\xgetallbut` means that you want to *get all the data items* but *the data items specified in the* `#ID-list`.

Inversely, the macro `\xkillallbut` means that you want to *ignore all the data items* but *the data items specified in the* `#ID-list`.

`Trick` :: **TRICK**

These macros are affected by `\xget` and `\xkill`. For e.g., the sequence

```
\xget{1,2,3} \xgetallbut{3,4,7}
```

will be understood as `\xgetallbut{1,2,3,4,7}`.

Here're some other examples (may be useless for you)

```
 \xget{1,2} \xgetallbut{2,3,4}  ⟶ \xgetallbut{1,2,3,4}
\xkill{1,2} \xgetallbut{2,3,4}  ⟶ \xgetallbut{2,3,4}
 \xget{1,2} \xkillallbut{2,3,4} ⟶ \xkillallbut{1,2,3,4}
\xkill{1,2} \xkillallbut{2,3,4} ⟶ \xkillallbut{2,3,4}

  \xgetallbut{2,3,4} \xget{1,2}          ⟶ \xgetallbut{1,2,3,4}
  \xgetallbut{2,3,4} \xkill{1,2}         ⟶ \xgetallbut{3,4}
\xkillallbut{2,3,4} \xget{1,2}           ⟶ \xkillallbut{1,2,3,4}
\xkillallbut{2,3,4} \xkill{1,2}          ⟶ \xkillallbut{3,4}
```

3.4 Openning the data file

\xopenlib Syntax

```
\xopenlib foo;
```

where foo[1] is a data file. The semicolon ';' is mandatory.

The commands \xget, \xkill, etc., in the previous section play no rule with the data file. They just tell \xopenlib that the 'bxx' should be got/ignored. It's \xopenlib that opens the data file, looks for the 'bxx' and does many other things. So if you want to \xget{1,3,2}, the really code is

```
\xget{1,3,2} \xopenlib foo;
```

See subsection 3.5 for a trick.

3.5 Specifying the default data file

\xlib It's convenient to specify a default data file. Let's do it by

```
\xlib foo;
```

The file extension '.tex' can be omitted.

Trick :: **TRICK**

After specifying the default data file, you can open that file by

```
\xopenlib;
```

3.6 Getting orderly the data items

Do you have any questions about the order of the 'bxx's?

Yes, if you had, the answer is: the calling of \xget, \xgetall, \xkill, etc., will produce the output in which the order of the data items is same as the order of data items in the data file. More concretely, if in 'foo.tex' you specify

```
\bxx 15;
      ITEMA.
\exx
\bxx 14;
      ITEMB.
\exx
```

then \xget{14,15} \xopenlib foo;
or \xget{15,14} \xopenlib foo;
gives the same results: 'ITEMA' is put before 'ITEMB'.

So we need the macro described below.

\xspec Syntax (same as \xget)

[1] The extension '.tex' may be omitted, so \xopenlib foo; is the same as \xopenlib foo.tex;.

```
\xspec(#env){#ID-list}
```

`(#env)` is also an optional parameter.

This macro opens the data file (so you needn't to specify a `\xopenlib...` after it), extract the data items from the data file within the order in the `#ID-list`.

So if you want to put '`ITEMB`' before the '`ITEMA`' (see above example), you should call

```
\xspec{14,15}
```

4 Advanced features

4.1 Enabling/Disabling section in data file

A data file may be divided into sections. But when looking for data file, this package will be typeset normally anything that isnot inside any a '`bxx`' (for e.g, a `\section` command), so you can see some strange outputs.

`\xenablesection` `\xdisablesection`

Syntax

```
\xdisablesection
\xenablesection
```

The first macro disables three commands

```
\section \subsection \subsubsection
```

The second enables three above section commands by restoring them to the values that this package captured at the beginning of the document. (So if you redefine a section command inside `\begin{document}` and `\end{document}`, you may lost that new command after using the sequence `\xdisablesection \xenablesection`.

NOTE:

`\xdisablesection` cannot disable the command

```
\section[optional]{...}
```

4.2 Data item followed by 'hint' environment

`hint`

If a data item, or a '`bxx`', in data file is an exercise, it is often followed by a hint (a solution). All the hints should be collected in a private place. The '`hint`' environment helps you in this behaviour. You put '`hint`' just after the declaration of a data item, like this

```
\bxx 100;
      This is Exercise A
\exx
\begin{hint}
      This is the hint of excercise A.
\end{hint}
```

When you get the above '`bxx`', for e.g., by `\xget{100}`, the content of the hint, here's *"This is the hint..."* will be written automatically to a file called '`hintfile`'.

`\xopenhint` If you want to open the hint file, just call

`\xopenhint`

IF YOU WANT TO KNOW MORE...

The '`hintfile`' is really a data file, whose name is

`\jobname.KTVhint`

`\jobname` is the name of current job, and '`.KTVhint`' is the extension[2] provided by the author.

By default, all the contents of the '`hintfile`' will be loaded (`\xgetall`).

It's too complex to explain the structure of the '`hintfile`'. Let's typset the test, look for the '`hintfile`' and the results for more details. Thank you!

5 Important notes (to the users)

a) (bug) You must put `\xenv(#env)` before any calling `\xget`, `\xkill`, etc. Try this bug to know how the package works.

b) Donot put `\xdisablesection` just before `\tableofcontents`. (Otherwise, an error will be reported.)

c) The environment(s) of the data items in the data file must be predefined.

d) `\xopenhint` should be called at the very end of the document, and must be called after any `\xspec`, `\xopenlib` commands. (`\xopenhint` will closed the '`hintfile`'; after that the writting operations have no effect on this file.)

e) (bug) Labelling the '`bxx`' maynot work well. If you try to load twice a '`bxx`', then two loadings will have a same label. In next version, we will redefine the `\label` and `\ref` to fix this bug.

6 Generating package and example

Excuting

latex `ktv-texdata.ins`

to generate the package (`ktv-texdata.sty`) and a test (*two files:* `ktv-test.tex`; `ktv-data.tex`). Then typeset file `ktv-test.tex`.

[2]If your system doesnot support the long extension file name, please report to the author.

7 Implementation

7.1 Notes

The implementation is the document **used only** by the package's hacker!!!

This package is written by `top-down` technique: A macro `\fooA` sometimes calls the macro `\fooB` whose the definitions appears after the defintions of `\fooA`.

This package uses some advanced and interesting macro techniques (*unknown-number-of-parameters-macro*, *two-optional-parameters-macro*, and much more). If you're a newbie of LaTeX programming, you will learn some new things by learning the code of package.

7.2 Requirements. Options

```
1 ⟨*package⟩
2 \RequirePackage{verbatim}
```

`\@@xdetail` This macro shows *string-id* of the current '`bxx`' on the margin.

```
3 \def\@@xdetail{\marginpar{{\bf\@xenv}\@xlbl}}
```

`detailon` `detailon` is the default option. Anything else means `detailoff`.

```
4 \DeclareOption{detailon}{\let\@xdetail\@@xdetail}
5 \DeclareOption*{\let\@xdetail\relax}
6 \ExecuteOptions{detailon}
7 \ProcessOptions
```

7.3 General purpose macros

`\b@sy` `\b@sy` means 'not `\relax`'.

```
8 \def\b@sy{}
```

`\if@xNIL` Check if string `#1` is empty.

```
9 \def\if@xNIL#1;{\ifx\relax#1\relax}
```

7.4 Scanning and setting flag for *string-id*

First, consider the example when the user calls

```
\xget(exercise){ex:1,-ex:3,ex:5,ex:7,ex:9,ex:11,ex:13,ex:15}
```

Here he (the user) wants to get the '`bxx`' that is numbered oddly, and that uses the environment '`exercise`' from the data file (**but** he doesn't want to get `ex:3`).

Because the number of exercises he wants to get is unknown (he just specifies a list of the items he needs), we must design a macro that scans for all the items in the list. Moreover, this macro will set the flag '`get`' or '`donotget`' for each item (in the above example, the flag for `ex:3` is '`donotget`', for the others is '`get`').

\usr@xenv These macros are used to store the result of the scanning.
\usr@xlbl

```
\def\usr@xenv{}
\def\usr@xlbl{}
```

\@multact Scan and set the flag (`get`, `donotget`) for the items of a list. The syntax is

```
\@multact(#usr-env){#ID-list}
```

where `#ID-list` is a list of `ID`(s)[3] that're seperated by the comma (‘`,`’). Moreover, each `ID` can be prefixed by a plus/minus sign (`+`, `-`).

```
#ID-list  ⟶ XID / #ID-list, XID
     XID  ⟶ ID / +ID / -ID
```

If `ID` is prefixed by a `+` (resp., a `-`), it means that this `ID` has a ‘*positive*’ (resp., ‘*negative*’) meaning[4] in the current context. An `ID` without prefix is the same as `+ID`.

The `#usr-env` is any environment (maynot be predefined). This `#usr-env` is concatenated with every `ID` in the `#ID-list`. The concatenation will create the *string-id*(s) for which we will later set the flag.

Here we use (`#usr-env`) instead of (`#env`) because the environment `#usr-env` is used at the time the user wants to *get some thing* from the data file (for e.g., by the command `\xget{ex:1,ex:2}`). This environment may be different from the environment we specify in the data file, it maynot even be predefined.

Note that `(#usr-env)` is optional parameter of macro `\@multact`. In case that `(#usr-env)` is omitted, `#usr-env` will get the default value saved in `\@@xenv`.

```
\def\@multact{\futurelet\@tchar\chk@multact}
```

We first check if the next char is a ‘`(`’.

```
\def\chk@multact{%
    \ifx(\@tchar
        \let\@txen\opt@multact
    \else
        \let\@txen\nop@multact
    \fi
    \@txen}
```

If the next char is a ‘`(`’

```
\def\opt@multact(#1)#2{%
    \def\usr@xenv{#1}%
```

then read the `#ID-list` by calling `\@xmultarg` (see subsection 7.5 below)

```
    \@xmultarg{#2}}
```

If the next char is not a ‘`(`’, `\usr@xenv` gets the default value

```
\def\nop@multact#1{%
    \def\usr@xenv{\@@xenv}
```

[3] see subsection 2.2.1.

[4] If the user wants to get something, the `+` means ‘*he wants*’, but `-` means ‘*he doesn't want*’. If the user doesn't want to get someting, the `+` means ‘*no, he doesn't want*’, while the `-` means ‘*yes, he wants*’.

then read the `#ID-list` by calling `\@xmultarg`

```
25      \@xmultarg{#1}}
```

7.5 Extracting ID from the #ID-list

`\@action` `\@action` is *something* we want to affect each *string-id*[5] found. We first let `\@action` to `\relax`, so we avoid seeing the error '`! Undefined control...`'. (This letting existed in some previous versions of package. Now the author doesnot know what's going on if this line is removed!)

```
26 \let\@action\relax
```

`\@@MINUS` `\@@MINUS` `\@@ZERO` There are three kinds of actions (`plus` action, `minus` action, and `nosign` action). We'll see in subsection 7.11 that every action is named '`MINUS`*something*', or '`ZERO`*something*', or '`PLUS`*something*'.

```
27 \def\@@MINUS{MINUS}
28 \def\@@PLUS{PLUS}
29 \def\@@ZERO{ZERO}
```

`\@xmultarg` `\@xmultarg` is used to extract the `ID`(s) from the `#ID-list`. Here we use the macro technique specified in [VE]. First, we add to the argument `#1` the terminator[6] `\@endlbl` ('`@@@`'). Then we call `\@@xmultarg`.

```
30 \def\@xmultarg#1{\@@xmultarg#1,@@@,}
31 \def\@endlbl{@@@}
```

`\@@xmultarg` Now is the extracting.

```
32 \def\@@xmultarg#1,{%
33     \def\@xtempi{#1}
34     \ifx\@endlbl\@xtempi
35         % do nothing
36     \else\@@@xmultarg#1,
37         \expandafter\@@xmultarg
38     \fi}
```

`\@@@xmultarg` Every `ID` extracted will be concatenated with the `#usr-env` to create the *string-id*. The *string-id* will be affected by one of three actions (`plus`, `minus` or `nosign`/`zero`). We first check the sign of the `ID` in the `#ID-list`.

```
39 \def\@@@xmultarg{\futurelet\@tchar\chk@@@xmultarg}
40 \def\chk@@@xmultarg{% 2003/05/14
41     \ifx-\@tchar\relax
42         \let\@txen\@@MINUS
43     \else\ifx+\@tchar\relax
44             \let\@txen\@@PLUS
45         \else
46             \let\@txen\@@ZERO
47         \fi
48     \fi
```

[5] *string-id* is created by concatenation `#usr-env` and `ID`. See subsection 2.2.3.

[6] Because that '`@`' is never used to create an `ID`, the terminator '`@@@`' works well.

To affect `\@action` on the *string-id*, we must specify '`type`' of the action. The type is saved in `\@txen` (see previous codes).

```
\csname\@txen\@action\endcsname}
```

7.6 Comment creator macro

`\c@mm@nt` If we donot want to get a '`bxx`' from the data file, we just let temporarily macro '`\bxx`' to '`\c@mm@nt`'. This macro scans and passes (or *ignore*) the input *line by line*, until it finds the fisrt '`\exx`'. That's why every '`bxx`' must be ended by a '`\exx`' in a single line! And that's why '`bxx`' cannot be nested!

```
\gdef\c@mm@nt{%
    \begingroup
    \catcode`\^^M=12 %
    \x@comment}
{\catcode`\^^M=12 \endlinechar=-1 %
\gdef\x@comment#1^^M{%
    \def\@xtest{#1}%
    \ifx\@xtest\exx
        \let\@txen=\endgroup
    \else
        \let\@txen\x@comment
    \fi
    \@txen}}
```

7.7 Hint file. Hint creator environment

An exercise, for e.g., may be followed by a hint (or a solution). In a test, a book of excercises, all hints should be collected in a single file (named '`hint file`') that is genererated automatically by the package. The `hint file` is also a data file; of course, it contains only the hint that follows the '`bxx`' we want to get from the data file (the *active* '`bxx`'s).

Because a hint sometimes will be disabled, we need a boolean test `\if@xhint`.

```
\newif\if@xhint
```

The name of hint file is '`\jobname.KTVhint`'.

```
\newwrite\@xfhint
\immediate\openout\@xfhint=\jobname.KTVhint
```

We put some information in the first two lines of hint file.

```
{\catcode`\%=12
\immediate\write\@xfhint{%% File created automatically by `ktv-texdata'.}
\immediate\write\@xfhint{%% DONOT EDIT THIS FILE MANUALLY.}}
```

`hint` Here is a trick about 'hint' environment. We collect hints in a same file. So we need something to know exactly what '`bxx`' a 'hint' (in the hint file) associates with. *So.... we assume* that '`#env`' specified in '`bxx`' (in the data file) is a numberable environment. Then when a 'hint' follows a '`bxx`' (named `bxxA`) is written to hint

file, its number is the same as the number of '`bxxA`' in the output. So we need a counter[7] to index the active '`bxx`'.

```
\newcount\c@bxx
```

```
\newenvironment{hint}{% begin-part of 'hint'
```

If the hint is enabled

```
\if@xhint
    \let\exx\relax
```

In the numbered environment '`#env`', LaTeX automatically creates a counter[8] named '`c@#env`'. We let '`c@bxx`' to the current value of '`c@#env`', then subtract '`c@bxx`' by 1 (LaTeX will increase '`c@bxx`' later). We create an data item in the hint file, whose *string-id* is the same as the current '`bxx`'.

```
    \c@bxx=\csname c@\@xenv\endcsname%
    \advance\c@bxx by-1 %
    \immediate\write\@xfhint{\string\setcounter{\@xenv}{\the\c@bxx}}
    \immediate\write\@xfhint{\string\bxx(\@xenv)\@xlbl;}
```

The contents of the hint environment now will be written to the hint file.

```
    \let\do\@makeother\dospecials\catcode`\^^M\active%
    \def\verbatim@processline{%
        \immediate\write\@xfhint{\the\verbatim@line}}%
    \expandafter\verbatim@start
```

If the hint is disable, let it start a comment environment. Here we use '`\comment`' defined in `verbatim` package, not the '`\c@mm@nt`' defined in this package[9].

```
\else
    \def\exx{\exx}%
    \expandafter\comment
\fi}%
```

Now the end-part of hint environment. If the hint is enabled, we put an '`\exx`' to terminate '`\bxx`' in the hint file. The line `\noexpand\exx` was added here but the author forgot the reason. Please help him!

```
{%
\noexpand\exx
\if@xhint
    \immediate\write\@xfhint{\string\exx}
\fi}
```

7.8 Replacements of macro \bxx

Macro `\bxx` will be replaced by one of these macros `\@bxx`, `\@bxy`, `\@ball`, `\@bnone`, `\@bnonebut`, `\@ballbut`. The replacement depends on what the macro `\xgetfoo`, `\xkillfoo`, ... are specified by the user.

[7] Please try to figure out this counter.

[8] We should not check for the existing of this counter: the checking may eat so much time!

[9] *A fun story:* in the first version of package, the author used '`\comment`' by a mistake, but package ran very well. He later found this error, and changed '`\comment`' to '`\c@mm@nt`', then the package dumped into the errors!

For e.g., if the user calls `\xgetall`, then `\bxx` is replaced by `\@ball`; if they call `\xkillall`, then `\@bnone` takes place.

For more details about the replacement, see the definition of `\xget`, `\xgetall`, `\xkill`, `xkillall`, etc. in the subsection 7.13.

Here's the summary of the replacements.

Calling	The replacement of `\bxx`
`\xget`	`\@bxx`
`\xgetall`	`\@ball`
`\xgetallbut`	`\@ballbut`
`\xkill`	`\@bxx`
`\xkillall`	`\@bnone`
`\xkillallbut`	`\@bnonebut`
`\xspec`	`\@bxy`

The common syntax for the macros `\@bfoo` is

`\@bfoo(#env)[#thm]{#ID}`;

Here `#thm` is the optional argument of the environment `#env`. Both '`[#thm]`' and '`(#env)`' are the optional parameters of the `\@bfoo` (this means you can omit '`[#thm]`' or '`(#env)`' or both of them while callling '`\@bfoo`').

`\@bxx, \@bxy` Each macro `\@bxx`, `\@bxy` does two things.

Firstly, it gets the information about the current '`bxx`' (information contains: `#env`, `#thm` and `#ID`). We use the `\@bxxarg` macro .

Secondly, it calls `\@bfoodone` to typeset the content of the '`bxx`'.

```
\def\@bxx#1;{%
    \@bxxarg#1;%
    \@bxxdone}
\def\@bxy#1;{%
    \@bxxarg#1;%
    \@bxydone}
```

`\@ball` Macro `\@ball` first gets the information about the '`bxx`' (like `\@bxx`), but then it calls '`\@bdone@kern`', the kernel of the `\@bdone`.

```
\def\@ball#1;{%
    \@bxxarg#1;%
    \@bdone@kern}
```

`\@bnone` Macro `\@bnone` starts the comment by calling `\c@mm@nt`. This macro also disable the hint environment that follows the '`bxx`'.

```
\def\@bnone#1;{%
  \@xhintfalse
  \def\exx{\exx}
  \expandafter\c@mm@nt}
```

`\@bnonebut` Macro `\@bnonebut` ignores the contents of the '`bxx`' if and only if the command '`\csname\@xenv\@xlbl\endcsname`' equals to '`\relax`'.

First, the macro scans for information.

```
\def\@bnonebut#1;{%
    \@bxxarg#1;%
```

Then it checks the status of the command '`\csname\@xenv\@xlbl\endcsname`'. If this command is undefined, we turn off the next 'hint' and start the comment command (to ignore the content of the '`bxx`').

```
    \expandafter\ifx\csname\@xenv\@xlbl\endcsname\relax
        \@xhintfalse
        \def\exx{\exx}
    \expandafter\c@mm@nt
```

If this command is defined, the contents of the '`bxx`' will be typeset. Before calling '`\@bdone@kern`', we let the status of the command '`...\@xenv\@xlbl...`' to '`\relax`'. By doing this, we ensure that two '`bxx`'s in the data file that have the same *string-id* will be typeset only once.

```
    \else
        \expandafter\let\csname\@xenv\@xlbl\endcsname\relax
        \expandafter\@bdone@kern
    \fi}
```

`\@ballbut` Macro `\@ballbut` is inverse from the macro '`\@bnonebut`'.

```
\def\@ballbut#1;{%
    \@bxxarg#1;
    \expandafter\ifx\csname\@xenv\@xlbl\endcsname\relax
        \expandafter\@bdone@kern
    \else
        \expandafter\let\csname\@xenv\@xlbl\endcsname\relax
        \@xhintfalse
        \def\exx{\exx}
        \expandafter\c@mm@nt
    \fi}
```

7.9 Getting information from 'bxx'

`\@bxxarg` This macro reads the information about the '`bxx`'. The information contains `#env` (saved in '`\@xenv`'), `#thm` (saved in '`\xhead`'), `#ID` (saved in '`\xlbl`').

We first set the initial value for three macros:

```
\def\@xenv{}
\def\@xhed{}
\def\@xlbl{}
```

We define two macros that save the default values for the `#env` and `#thm`. The default value of `#thm` should be null (why?). Now the default `#thm` is aslo null, but in the subsection 7.13, we define macro `\xenv` to change this behaviour.

```
\def\@@xenv{}
\def\@@xhed{}
```

Now we define `\@bxxarg`.

```
\def\@bxxarg{\futurelet\@tchar\chk@bxxarg}
```

We check if the next char is a '`(`'.

```
\def\chk@bxxarg{%
    \ifx(\@tchar
        \let\@txen\env@bxxarg
    \else
        \let\@txen\nop@bxxarg
    \fi
    \@txen}
```

If the next char is a '`(`', we hope that the '`bxx`' specifies the '`(#env)`'. Here the author uses an '`\edef`', but he really doesnot know the reasons. The '`\def`' version sometimes causes a strange error.

```
\def\env@bxxarg(#1)#2;{%
    \edef\@xenv{#1}%
```

Now we look for the rest of the argument by calling `\@@bxxarg`.

```
  \@@bxxarg#2;}
```

If the next char isnot a '`(`', '`bxx`' uses the default environment.

```
\def\nop@bxxarg#1;{%
    \edef\@xenv{\@@xenv}%
```

Now we look for the rest of the argument (call `\@@bxxarg`).

```
  \@@bxxarg#1;}
```

`\@@bxxarg` The calling of `\@@bxxarg` will first check if the next char is a '`[`'.

```
\def\@@bxxarg{\futurelet\@tchar\chk@@bxxarg}

\def\chk@@bxxarg{%
    \ifx[\@tchar
        \let\@txen\hed@@bxxarg
    \else
        \let\@txen\nop@@bxxarg
    \fi
    \@txen}
```

If the next char is a '`[`', we hope that '`bxx`' specifies the '`[#thm]`'

```
\def\hed@@bxxarg[#1]#2;{%
    \edef\@xhed{#1}%
    \edef\@xlbl{#2}}
```

If the next char isnot a '`[`', the '`#thm`' gets the default value.

```
\def\nop@@bxxarg#1;{%
    \edef\@xhed{\@@xhed}%
    \edef\@xlbl{#1}}
```

7.10 Typesetting the content of 'bxx'

`\@bxxdone` This macro starts a `\c@mm@nt` (if the command '`...\@xenv\@xlbl...`' is '`\relax`'), or calls `\@bdone@kern` (the kernel of the `\@bxxdone`).

```
\def\@bxxdone{%
    \expandafter\ifx\csname\@xenv\@xlbl\endcsname\relax
        \@xhintfalse
        \def\exx{\exx}
        \expandafter\c@mm@nt
    \else
        \expandafter\let\csname\@xenv\@xlbl\endcsname\relax
        \expandafter\@bdone@kern
  \fi}
```

`\@bdone@kern` This macro typesets the contents of the '`bxx`' by starting the environment '`#env`'. It also turns on the next hint environment.

If the '`bxx`' specifies the '`#thm`', we should call '`\begin{#env}[#thm]`'; otherwise, we just call '`\begin{#env}`'. (Then the '`\exx`' equals to '`\end{#env}`'.)

Note that '`\begin{#env}[]`' is different from '`\begin{#env}`. So we must check if the '`#thm`' is empty.

```
\def\@bdone@kern{%
    \@xhinttrue%
    \def\exx{\end{\@xenv}}
    \expandafter\if@xNIL\@xhed;
        \def\@txen{%
            \begin{\@xenv}\@xdetail}
    \else
        \def\@txen{%
            \begin{\@xenv}[\@xhed]\@xdetail}
    \fi
    \@txen}
```

`\@bxydone` This macro just passes the arguments to `\@bxy@kern`.

```
\def\@bxydone{\@bxy@kern(\@xenv){\@xlbl}}
```

`\@bxy@kern` This macro is the kernel of `\@bxydone`. It opens the data file, searchs for the '`bxx`' whose *string-id* is '`\@bxy@id`'. If such '`bxx`' is found, the macro stops the seaching[10] and calls '`\@bdone@kern`' to typeset the contents of that '`bxx`'. Any '`bxx`' in the data file whose *string-id* doesnot match the string saved in `\@bxy@id` will be ignored.

Because a *string-id* never accepts the value '@', we just let `\@bxy@id` to '@' to stop the searching.

```
\def\@bxy@id{@}
\def\@bxy@kern(#1)#2{%
    \edef\@xtempi{#1#2}
```

[10] We really donot stop the searching. We just ignore the other '`bxx`'(s) in the data file. *Trick:* Each time this macro opens the data file, it gets at most one data item, while, for e.g., `\bdone@kern` may get more than one.

```
    \ifx\@bxy@id\@xtempi
        \def\@bxy@id{@}
        \expandafter\@bdone@kern
    \else
        \@xhintfalse
        \def\exx{\exx}
        \expandafter\c@mm@nt
    \fi}
```

`\@xspec@i` If you read the above codes carefully, you donot see anything about the '*openning the data file*'. Yes, the truth is that `\@bxy`, `\@bxydone` and aslo `\@bxy@kern` are always used within the `\@xspec@i`. This macro does the openning.

```
\def\@xspec@i(#1)#2{
    \let\bxx\@bxy
    \edef\@bxy@id{#1#2}
```

Now start the searching (openning)

```
    \input \@xlib}
```

7.11 Actions affect on the *string-id*

Each action `foo` must be associated with 3 routines

`PLUSfoo+#1`, `MINUSfoo-#1`, `ZEROfoo#1`,

`@@@setflag` The positive meanings of this action is to define the command

`\csname\usr@xenv#1\endcsname`

by letting this command to '`\b@sy`'.

```
\def\PLUS@@@setflag+#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\b@sy}
\def\MINUS@@@setflag-#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\relax}
\def\ZERO@@@setflag#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\b@sy}
```

`@@@killflag` The positive meanings of this action is to undefine the command

`\csname\usr@xenv#1\endcsname`

by letting this command to '`\relax`'.

```
\def\PLUS@@@killflag+#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\relax}
\def\MINUS@@@killflag-#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\b@sy}
\def\ZERO@@@killflag#1,{%
    \expandafter\let\csname\usr@xenv#1\endcsname\relax}
```

`@@@xspec` This action associates with the `\xspec` (see subsection 7.13 for details). This action, unlike `@@@killflag` nor `@@@setflag`, doesnot define/undefine the command '`...\usr@xenv#1...`'. The positive meaning of this action is *openning data file and getting the specified* '`bxx`'.

```
\def\PLUS@@@xspec+#1,{%
    \@xspec@i(\usr@xenv){#1}}
\def\MINUS@@@xspec-#1,{}
\def\ZERO@@@xspec#1,{%
    \@xspec@i(\usr@xenv){#1}}
```

7.12 Openning the data file

`\@xlib` `\@xlib` stores the data file specified by users.

`\@@xlib` `\@@xlib` stores the data file used by the `\@openlib`.
Currently, two macros save the null values.

```
\def\@xlib{}
\def\@@xlib{}
```

`\@openlib` Open the data file. The syntax of this macro is `\@openlib#1;`, where `#1` is the file name. If `#1` is omitted, the `\@openlib` uses the data file whose name is stored in `\@xlib`. Note that in this case, the calling of macro is `\@openlib;`.

```
\def\@openlib#1;{%
    \if@xNIL#1;
        \expandafter\if@xNIL\@xlib;
            % do nothing
        \else
            \def\@@xlib{\@xlib}
        \fi
    \else
        \def\@@xlib{#1}
    \fi
    \input \@@xlib}
```

7.13 User's macros

`\xspec` This macro provides a special ways to read the data file. After its scanning the data file, the order of the '`bxx`'s in the ouput is the same the order of the `ID`s specified in the `#ID-list`.

```
\def\xspec{%
    \def\@action{@@@xspec}%
    \@multact}
```

`\xlib` `\xlib` specifies the data file. `\xopenlib` opens the data file.

`\xopenlib`

```
\def\xlib#1;{%
    \edef\@xlib{#1}}
\let\xopenlib\@openlib
```

\xkill Macro's names speak that...
\xkillall
\xkillallbut

```
\def\xkill{%
    \let\bxx\@bxx
    \def\@action{@@@killflag}%
    \@multact}
\def\xkillall{%
    \let\bxx\@bnone}
\def\xkillallbut{%
    \let\bxx\@bnonebut
    \def\@action{@@@setflag}%
    \@multact}
```

\xget Macro's names speak that...
\xgetall
\xgetallbut

```
\def\xget{%
    \let\bxx\@bxx
    \def\@action{@@@setflag}%
    \@multact}
\def\xgetall{%
    \let\bxx\@ball}
\def\xgetallbut{%
    \let\bxx\@ballbut
    \def\@action{@@@setflag}%
    \@multact}
```

\xhintready The hint file must be closed ('ready') before being opened.

```
{\catcode`\%=12
\gdef\xhintready{
    \begingroup\catcode`\%=12
    \immediate\write\@xfhint{\string\endinput}
    \immediate\write\@xfhint{%% END OF FILE %%}
    \endgroup\immediate\closeout\@xfhint}}
```

\xopenhint Open the hint file.

```
\def\xopenhint{%
    \xhintready
    \xgetall
    \xlib \jobname.KTVhint;
    \xopenlib;}
```

\xenv Specify the default environment.

```
\def\xenv(#1){%
    \edef\@@xenv{#1}}
```

\xdetailon Turn the details on/off.
\xdetailoff

```
\def\xdetailon{%
    \let\@xdetail\@@xdetail}
\def\xdetailoff{%
    \let\@xdetail\relax}
```

\xenablesection
\xdisablesection

If we put some '\section' commands in the data file, we may want to disable/enable them.

First, we define a null section.

```
\def\nil@section#1{}
```

Then we save the old values of section-relatives just before \begin{document}. We mention only \section, \subsection, \subsubsection.

```
\AtBeginDocument{%
    \let\old@section\section
    \let\old@subsection\subsection
    \let\old@subsubsection\subsubsection}
```

Everytime you want to disable the section, we let \section to \nil@section.

```
\def\xdisablesection{%
    \let\section\nil@section%
    \let\subsection\nil@section%
    \let\subsubsection\nil@section}
```

To restore, let \section to his old value (that we captured at the beginning of document). If you redefine \section inside \begin{document} and \end{document}, you cannot get that new \section after using any \xdisablesection.

```
\def\xenablesection{%
    \let\section\old@section%
    \let\subsection\old@subsection%
    \let\subsubsection\old@subsubsection}
```

7.14 Initialization

See section 8 for a reason.

```
\let\bxx\@bxx
```

⟨/package⟩

8 History

v01.xx	2002/12/xx	the first design
v02.xx	2002/12/yy	*changes forgotten*
v03.14	2002/12/18	the first good version
v03.15	2003/03/26	some litle changes
v03.17	2003/04/19	use two-step macro technique `\bxx`,`\xlib`,`\xopenlib` needn't any ended-char
v04.18	2003/04/22	modify: `\@bxy@kern`
v04.19	2003/04/23	change: use `\par` instead of `\enlinechar` in def. of `\bxx`
v04.19	2003/05/06	use '`;`' as delimiter in definition of `\bxx`
v04.20	2003/05/11	optimize: `\@bnone`
v04.21	2003/05/12	*changes forgotten*
v05.23	2003/05/13	add: `\xgetallbut`, `\xkillallbut`; optimize
v05.24	2003/05/13	remove: `\ifdetail`, `\if@xnil` (boolean var.) and optimize
v05.25	2003/05/14	fix: some litle bugs; remove: `\@act`
v05.26	2003/05/14	remove: bad options
v05.27	2003/05/15	bug: `\bxx(verbatim)...;` failed
v05.28	2003/05/17	bug: `\foo-section` cause error move: `\foo-section` to the very end of package
v05.29	2003/05/18	bug: `\bxx` canot be nested. (nesting: `deactive bxx` causes error) (nesting: `\xgetall` works well, but contents of `hint` is too bad.)
v05.30	2003/08/xx	rewrite: document (vietnamese), optimize
v05.31	2003/09/20	rewrite: document (english), optimize
v05.32	2003/09/20	rewrite: document (english), optimize remove: token list `\everyactivebxx` bug: cannot use `\xgetallbut`, `\xkillallbut`, `\xget`, etc., before any `\xenv(#env)`
v05.33	2003/09/21	remove: initialization add: `\xopenhint`
v05.34	2003/09/22	bug: otions donot work. fix: put `\@@xdetail` before `\DeclareOption{detailon}` bug: if `\xopenlib;` before any `\xget...` then `\bxx` is `unknown control sequence` fix: `\let\bxx\@bxx` to initialize
v05.39	2003/09/20	(future) optimize: `\@xspec`

9 Miscellanea

The author write this package because he has some big libraries of mathematical exercises, but he really doesn't like `copying` and `pasting` everytime he edits

a new test for students.

This package may contains some bugs. So the author hopes that you can help him, even a bit.

In section 1, the author mentioned the ability of getting the data items whose `ID`s are numbered oddly.... Currently this feature is unsupported . Please wait for the next version, or you should do something by yourself!

The author's English isnot quite well. He often mis-spells and dumps into error. He would like to be sorry!

If you find out that the package is useful for your private works, please send to the author a little notice. Thank you very much!

The emails of the author is

`kyanh@inic.biz, kyanh@linuxmail.org`

References

[VE] Victor EIJKHOUT, *TEX by Topic*, Addison-Wesley, 1992.