Customizing Bibliographic Style Files

Patrick W. Daly

This paper describes program `makebst`
version 4.1 from 2003/09/08*
(including additions by Arthur Ogawa, `ogawa@teleport.com`)

1 Introduction

This TEX program is meant to be used together with generic bibliographic style files to produce customized `.bst` files for running with BIBTEX. The generic, or master file, can be processed by docstrip with selected options to achieve the desired bibliographic style. To this end, a docstrip batch job should be made up. However, because of the large number of options available, an interactive, dialogue system would be more convenient.

This program, `makebst`, accomplishes this goal. It defines macros to establish such a docstrip batch job file, and to organize a menu of options. The menu information is contained, however, in the master file itself, since the two are intimately related. Thus different master files with totally different option structures may be accommodated.

The batch job could in fact be made up with an editor without calling `makebst`, but this program does simplify the task.

Incidentally, the docstrip run can only be carried out by means of a batch job. Running docstrip interactively inserts default pre- and postambles in the text, the latter including an `\endinput` command that BIBTEX will not understand.

2 The Master File

The master file is a BIBTEX bibliographic style file containing alternative coding depending on docstrip options. The options are selected when docstrip is run, either interactively or through a batch job.

Suppose that one of the options is called `xyz`. Then the following alternatives are possible:

*Work on `custom-bib` 4.00 was supported by the American Physical Society

`%<xyz>` *one line of coding*

includes the single line of coding;

`%<!xyz>` *one line of coding*

excludes the single line;

`%<*xyz>`
several lines of coding
`%</xyz>`

includes all the bracketed lines;

`%<*!xyz>`
several lines of coding
`%</!xyz>`

excludes all the bracketed lines.

Options may be logically combined: the symbol `|` is a logical or, `&` a logical and, `!` a logical not; parentheses `(` and `)` may be used to group options.

2.1 Using with docstrip

(The docstrip command syntax shown here is that for version 2.4 and later, released December, 1996.)

In order to generate a true BibTeX style file with selected options from the master file, it is necessary to run a docstrip batch job. Suppose that the master file is named `master.mbs`, the resulting BibTeX style file is to be `silly.bst`, and the batch job file itself is called `silly.dbj`. To produce this with options, say, `xyz` and `abc`, the batch job would look something like:

```
\input docstrip

\preamble
This is for Journal of Silly Results
\endpreamble

\postamble
End of customized bst file
\endpostamble

\keepsilent

\askforoverwritefalse
```

```
\generate{\file{silly.bst}{\from{master.mbs}{xyz,abc}}}
\endbatchfile
```

A preamble is not necessary, although it is advisable to include some statement about the application of the bibliographic style. A postamble *is* vital, otherwise the default will add `\endinput` at the end of the file, something that BibTeX will not understand. The `\keepsilent` is optional and just suppresses docstrip output during processing. Similarly the `\askforoverwritefalse` suppresses the warning that a file of the same name is to be overwritten.

2.2 The Menu File

This program, `makebst`, simplifies the creation of the batch job file. To do that, it needs information on the available options. This information must be stored in a special format, in the master file itself. Alternatively, that information may be extracted and stored in a file with the same root name but extension `.opt`. *This feature is not recommended since it can lead to inconsistencies!* The format of the menu information is illustrated below in Section 4.

In the master file, this information must be enclosed within docstrip options `%<*options>` ... `%</options>` and *must* be ended by an `\endoptions` command. It may also include any number of comments. The rest of the file must be enclosed within `%<*!options>` ... `%</!options>` to exclude it when the menu information is extracted with docstrip.

A sample menu in the master file to select one or none of options `xyz` *or* `zyx` would look thus:

```
%<*options>
\mes{Select one of these}
\optdef{f}{xyz}{Option forword}{to do forward stuff}
\optdef{r}{zyx}{Option reverse}{to do reverse stuff}
\optdef{*}{}{None of the above}{}
\getans
\endoptions
%</options>
%<*!options>
. . . . . .
%</!options>
```

An explanation of these commands is to be found in Section 4.

The menu information may be extracted from the master file by means of docstrip and stored in a file with extension `.opt`. If this file is present, `makebst` offers to read it instead of the master file, although this is *not* recommended, as explained above.

3 The Program makebst

3.1 Installing makebst

The makebst program comes as a documented source file named makebst.dtx, which needs to be processed by docstrip to extract the actual 'program' file makebst.tex. The easiest way to do this is simply to process the installation batch file makebst.ins with TEX or LATEX, as

```
tex makebst.ins
      or
latex makebst.ins
```

There are in fact three variants of makebst that may be extracted: the basic one lists by default only those options that have been selected; the more refined one (and the default) lists all options offered with the rejected ones commented out; the third version also adds more detailed comments. Even in the first two versions, the user will be asked interactively if s/he wants the additional features of the others.

One can select the variant by editing makebst.ins.

Another choice that can be made is whether the .dbj files are to conform to docstrip version 2.4 syntax or not. By default, makebst.ins tests the current version and automatically configures makebst.tex to write the correct syntax. This too may be overridden by editing makebst.ins. (Note that the older syntax is still understood by the newer version of docstrip.)

Reminder: the older syntax requires \def\batchfile{⟨*filename*⟩} as the first line in a docstrip batch job, where *filename* is the name of the batch file itself. The newer syntax does not need this line, but requires \endbatchfile at the very end instead. The advantage of the new syntax is that one can edit and rename such a file without having to change its name in the first line. The old syntax leads to great frustration if one forgets to change *filename*.

Another difference in the syntaxes (actually introduced in version 2.3 in June, 1996) is the use of the command \generate instead of \generateFile. Its advantage is that it permits multiple files to be extracted in one pass, something that is not exploited at all by makebst.

3.2 Running makebst

This is actually a TEX program, although it will also run under LATEX. In that sense, it is like docstrip itself. Thus run the program with (something like)

```
tex makebst
      or
latex makebst
```

The program first asks for the name of the master file. This is the file containing all possible bibliographic style commands, with docstrip options for selective output. A default name is offered, as well as a default extension (`.mbs`).

Next, the program asks for the name of the output file, the `.bst` file. The extension here is optional, defaulting to `.bst`. This name also determines the name of the batch job file, which will have the same root name with the extension `.dbj`, for the *docstrip batch job*.

The actual interrogation then begins. All the information for the menus is contained in the master bibliographic style file. The reason for this is that the menu information must conform to the available options in the master file, so it makes sense that one file should contain both. The master file is only read up to the `\endoptions` command.

Finally, the batch job file is closed, and the user is asked if it should be run. If he does not take up this offer, or if he later edits the batch job, then it may be run manually with (something like)

`tex` *bstname*`.dbj`

4 The Menu Information

The set of questions in the interrogation must fit the available options in the master file. For this reason, the menu information is contained in the master file itself. The program `makebst` supplies the macros that are used in the menu file to simplify writing and processing menu information.

`\mes` To print a message to the terminal, use `\mes{`⟨*text*⟩`}`. A new line may be forced within *text* by means of `^^J`.

`\ask` To interrogate the user for a response, use `\ask{\`*com*`}{`⟨*text*⟩`}`, which writes *text* to the terminal, and puts the response in the command `\`*com*.

`\optdef` Almost all interrogations will consist of a list of mutually exclusive options, one of which is the default. For each item in the list, one must specify the keyboard response that is to select it, the actual name of the docstrip option that realizes it, and two pieces of explanatory text. For example,

```
\optdef{a}{abr}{Abbreviations}{of such words}
```

means that `abr` is the true docstrip option name that is selected by typing `a`. The two explanatory texts are written to the terminal immediately as part of the menu, but only the first text is echoed when the selection is made (for confirmation) and is also written to the batch job file (as comment).

The default option must have the response `*`.

A menu is written to the terminal, first with a `\mes` command to state the subject matter, and then with a sequence of `\optdef` statements, each of which also

\getans writes the texts to the terminal. The response is then read in and processed with \getans, which writes the reply to the command \ans and writes the appropriate docstrip option to the batch job file. If the response does not correspond to any of those in the menu list, it is set to *; if there is no * in the list, then \ans is set to the last entry. The command \ans is still available afterwards for any extra testing that might be needed.

An example menu appears as follows:

```
\mes{^^JJOURNAL VOLUME NUMBER:}
\optdef{*}{}{Volume plain}{as vol(num)}
\optdef{i}{vol-it}{Volume italic}%
       {as {\string\em\space vol}(num)}
\optdef{b}{vol-bf}{Volume bold}%
      {as {\string\bf\space vol}(num)}
\optdef{d}{vol-2bf}{Volume and number bold}%
      {as {\string\bf\space vol(num)}}
\getans
```

\beginoptiongroup Further structure for the interrogation is provided by the \beginoptiongroup ... \endoptiongroup sequence, which should act as a container for the \mes ... \optdef ... \getans commands described above. For example:

```
\beginoptiongroup{JOURNAL VOLUME NUMBER:}{}
\optdef{*}{}{Volume plain}{as vol(num)}
\optdef{i}{vol-it}{Volume italic}%
       {as {\string\em\space vol}(num)}
\optdef{b}{vol-bf}{Volume bold}%
      {as {\string\bf\space vol}(num)}
\optdef{d}{vol-2bf}{Volume and number bold}%
      {as {\string\bf\space vol(num)}}
\getans
\endoptiongroup
```

presents the same effect as the previous example. The virtue of the option group is in providing a single markup for all interrogations and having a consistent appearance in the generated file.

This feature has been added with version 4.0 of makebst.

5 More Complex Batch Jobs

Version 3.0 of makebst allows the master file to define more sophisticated batch jobs, such as additional master files with their own options. This is made possible because the options are not written directly in the \generate command, as in earlier versions, but to a command \MBopts. The batch file then contains something like:

```
\def\MBopts{\from{⟨source.ext⟩}{%
  lines from menu session
 }}
\generate{\file{⟨output.ext⟩}{\MBopts}}
```

Normally the *lines from menu session* contain just the docstrip options. However, the master file could add other things to the definition of `\MBopts`, even closing it and starting a new definition. It just has to make sure that the braces are balanced.

`\MBaskfile` A number a macros are provided, which are used by `makebst` itself, to simplify making complex menus. To ask for the name of a file interactively,

```
\MBaskfile{⟨Prompting text⟩}(⟨root.ext⟩){⟨io⟩}\⟨fname⟩
```

may be given, where *root.ext* is the default name of the file, *io* is `i` (for input) if the file must already exist, and `\`*fname* is the command that receives the file name. The root name will be in `\froot`, the extension in `\fext`.

`\wr` Text is written to the batch job file with

```
\wr{⟨text⟩}
```

Any commands in *text* that are to be written literally must be preceded by `\string`.

`\MBswitch` Since any braces in *text* must be balanced, something special must be done to permit them to be printed as normal characters. The command `\MBswitch` accomplishes this; the parentheses `( )` replace `{ }` as the delimiters. This should always be given within `\begingroup` ... `\endgroup`.

As an example, suppose the master file contains only half the coding for the `.bst` file, the other half being in one of several other master files. We must prompt for this second file, include it for its options, and make sure that `\MBopts` knows about it. The following code in the master file will do this.

```
\MBaskfile{Name of second master file}(aa.mbs)i\xfile
\begingroup\MBswitch
\wr(\string\MBopta})
\wr(\string\from{\xfile}{\string\MBoptb}})
\wr(\string\def\string\MBopta{\pc)
\endgroup
regular menu information for first file
\begingroup\MBswitch
\wr(}\string\def\string\MBoptb{\pc)
\endgroup
\input\xfile\relax
\begingroup\MBswitch
\wr({\pc)
```

```
\endgroup
\endoptions
```

The resulting `.dbj` file contains

```
\def\MBopts{\from{first.mbs}{%
\MBopta}
\from{second.mbs}{\MBoptb}}
\def\MBopta{%
first set of options
}\def\MBoptb{%
second set of options
{%
  }}
\generate{\file{sample.bst}{\MBopts}}
```

6 Coding

This section presents and explains the actual coding of the macros. It is nested between `%<*program>` and `%</program>`, which are indicators to docstrip that this coding belongs to the program file.

6.1 Preliminaries

The first thing is to open up i/o devices for communicating with the terminal and files. (Some of this has been borrowed from docstrip.) The terminal input and output are `\ttyin` and `\ttyout` respectively, while the output file if `\outfile`.

```
⟨*program⟩
\newwrite\outfile
\newread\ttyin
\newread\infile
\newwrite\ttyout
```

`\mes` `\wr` `\umes`

The commands for outputting text are defined: `\mes` writes to the terminal, `\wr` writes it argument directly to the output file, while `\umes` writes to the terminal and adds its argument as a comment to the output file.

```
\def\mes{\immediate\write\ttyout}
\def\wr#1{\immediate\write\outfile{#1}}
\def\umes#1{\mes{^^J#1}\wr{\pc#1}}%
```

To assist inserting new lines in the middle of text, define a newline symbol.

```
\newlinechar=‘\^^J
```

\MBswitch There are times when we need to write a line of code to the output file with unbalanced braces in that line. (They are balanced in another line.) Such lines are written with `\wr{...}`. If the braces in the argument are not balanced, then there will be trouble. To get around this, change the category codes of the braces to 'other' and let parentheses take their place.

```
\def\MBswitch{\catcode`\{=12 \catcode`\}=12
              \catcode`\(=1 \catcode`\)=2\relax}
```

The way to employ this is as

```
\begingroup\MBswitch
\wr(..{..)
\endgroup
```

\ask To get a response from the terminal, use `\ask`. However, there are some complications here. If only carriage-return is pressed, then the response command is equal to `\par`; for anything else, a typed-in text includes a trailing blank. We must test for `\par` and remove the blank if it is there.

```
\def\defpar{\par}
\def\remblk#1 @@{#1}
\def\ask#1#2{\mes{#2}\read\ttyin to #1\ifx#1\defpar\def#1{}\else
   \edef#1{\expandafter\remblk#1@@}\fi}
```

\getroot \getext To parse the name of a file into root and extension, use commands `\getroot` and `\getext`.

```
\def\groot#1.#2@@{#1}
\def\getroot#1{\expandafter\groot#1.@@}
\def\gext#1.#2.#3@@{#2}
\def\getext#1{\expandafter\gext#1..@@}
```

\MBaskfile Several times it is necessary to ask for a file name interactively, and maybe test if it exists. This might even be done in the `.mbs` file, so provide a macro to simplify this task. The syntax is

`\MBaskfile{`⟨*Prompting text*⟩`}(`⟨*root.ext*⟩`){`⟨*io*⟩`}\`*fname*

where *root.ext* is the default name for the file sought, and *fname* is the command that contains the final file name. The commands `\froot` and `\fext` will contain the root and extensions of the file name, if they are needed for further parsing. If *io*=`i` (for input), then the resulting file must already exist, else the macro loops again. If *root* is blank, then only the extension is given as default, but a file root name must be entered.

```
\def\MBaskfile#1(#2.#3)#4#5{%
\loop
```

```
  \def\ans{#2.#3}
\if!#2!
 \if!#3!\ask{#5}{#1}\fi
  \ask{#5}{#1 (default extension=#3)}\else
  \ask{#5}{#1 (default=\ans)}
\fi
  \ifx#5\empty \edef#5{\ans}\fi
  \edef\froot{\getroot#5}
  \edef\fext{\getext#5}
  \ifx\fext\empty \def\fext{#3}\fi
  \edef#5{\froot.\fext}
\if#4i
  \def\temp{Cannot find file `#5'}
  \openin\infile#5\relax
  \ifeof\infile \def\ans{}\fi \closein\infile
\else
 \def\temp{There is no default}
 \ifx\froot\empty \def\ans{}\fi
\fi
  \ifx\ans\empty \mes{*** \temp}
\repeat}
```

`\pc` `\pcpc` `\spsp` Now for some special commands to simplify outputting % signs and double spaces to the output file.

```
{\catcode`\%=12
 \gdef\pc{%}
 \gdef\pcpc{%% }
}
\def\spsp{\space\space}
```

`\Now` In order to date-and-time-stamp the resulting batch job file, we need macros to produce the current date and time. (In TEX there is no `\today` command.)

```
\newcount\hours
\newcount\minutes
\def\SetTime{\hours=\time
        \global\divide\hours by 60
        \minutes=\hours
        \multiply\minutes by 60
        \advance\minutes by-\time
        \global\multiply\minutes by-1 }
\SetTime
\def\now{\number\hours:\ifnum\minutes<10 0\fi\number\minutes}
\def\today{\number\year/\ifnum\month<10 0\fi\number\month
   /\ifnum\day<10 0\fi\number\day}
\def\Now{\today\space at \now}
```

6.2 Menu Macros

\optdef For each menu, a general text is written with \mes, followed by a list of available options. The information that will be needed is

1. the response letter to select the option,
2. the actual docstrip option name, as defined in the master bibliographic style file,
3. a piece of text that is printed in the menu list, to be echoed in confirmation of the choice, and also to be written to batch job file as a comment,
4. a second piece of text that is only written to the menu, to enhance the explanation.

The true option name and the two pieces of text are stored as commands prefixed by \opt@, \txt@, and \cmt@ respectively, followed by the response letter. Each option response letter is also stored in a list \optlist which is initialized to empty. The commands \nxtopt and \rstopt are used to extract the next and remaining options from the list.

```
\def\optdef#1#2#3#4{%
  \expandafter\def\csname opt@#1\endcsname{#2}%
  \expandafter\def\csname txt@#1\endcsname{#3}%
  \expandafter\def\csname cmt@#1\endcsname{#4}%
  \edef\optlist{\optlist#1,}%
  \def\OptAns{#1}%
  \mes{(#1) #3\space #4}%
}
\def\optlist{}
\def\nxtopt#1,#2@@{#1} \def\rstopt#1,#2@@{#2}
```

\getans The user selection is read in with \getans, into the command \ans. It then processes the response by first checking if there is an option corresponding to it; if not, the response \ans is set to the default *. If no star response exists, then it takes the last one entered as the default response. It then calls \wropt to write the necessary docstrip option and explanatory comment to the batch job file. Finally, it uses the option list \optlist to clear all the \opt@ commands. This last step is necessary to avoid conflicts with previous menus: without it, a response that is not in the current list might however exist from an earlier menu.

Note that prior to version 4.0 of this code, the \optlist was built via head accretion and traversed from the head back, that is, in LIFO order. As of version 4.0 it is processed in FIFO order. This way, the comments in the .dbj file are in the same order as the \optdef statements in the master file. The flag character (to terminate parsing the \optlist) is now a $\%_{12}$, which cannot be entered as a response by the user, and is appended to the list at the beginning of \getans processing.

```
\newif\ifsw
\def\getans{%
 \edef\optlist{\optlist\pc,}%
 \ask{\ans}{\spsp Select:}%
  \expandafter\ifx\csname opt@\ans\endcsname\relax
  \def\ans{*}%
 \fi
 \expandafter\ifx\csname opt@\ans\endcsname\relax
  \let\ans\OptAns
 \fi
  \edef\ansx{\csname opt@\ans\endcsname}
  \swtrue \loop
    \edef\temp{\expandafter\nxtopt\optlist@@}%
    \edef\optlist{\expandafter\rstopt\optlist@@}%
  \if\temp\pc\swfalse\else
   \if\temp\ans
    \wropt\ans
   \else
    \ifoptlist\wrxopt\temp\fi
      \fi
      \expandafter\let\csname opt@\temp\endcsname\relax
    \fi
  \ifsw \repeat
 \def\optlist{}%
 \ifoptverbose
  \wr{\pc------\string\ans=\ans (==\ansx)-------}%
 \else
   \ifoptlist
     \wr{\pc--------------------}%
   \fi
 \fi
}
```

A special request from Frank Mittelbach asks if a list of unused options cannot be added to the batch job file, to assist editing it by hand. In this way, one knows what the docstrip options are immediately without having to search for them in the `.mbs` documentation.

This feature was added in version 2.1, but by means of a docstrip option, so it could be turned off if necessary. Thus the extracted `makebst.tex` file produced either the full list or only the selected options. Here 'full list' means only those options that were offered. Any options that were conditionally offered, depending on previous selections, could be missing.

For version 4.0, Arthur Ogawa changed this so that the full option list is switched on with the `\ifoptlist` flag, and not by an option at docstrip extraction time. He also added an `\ifoptverbose` flag to include even more comments in the `.dtx` file. The user is asked at run time if s/he wants to activate these features.

Furthermore, one can use the \beginoptiongroup ... \endoptiongroup idiom to handle cases where options should be offered only contingent upon some condition. By doing so, the unused options are still presented as comments in the batch job file, along with a comment showing the dependency and a matching comment showing the scope.

Finally, there are still docstrip options optlist and optverbose which produce makebst.tex with the corresponding flags already set, in which case the features are always activated and the user interrogation is suppressed.

\wropt The actual outputting of the option command to the batch job file is done by \wropt. It tests if the option name is blank (a default in the master bibliographic style, which need not be the menu default), writes out the option name, if present, and adds the explanatory comment.

```
\def\wropt#1{%
 \edef\tempx{\csname opt@#1\endcsname}%
 \if!\tempx!
  \wr{\spsp\spsp\pc: (def)
   \csname txt@#1\endcsname
   \ifoptverbose\space\csname cmt@#1\endcsname\fi
  }%
 \else
  \wr{\spsp\tempx,\pc:
   \csname txt@#1\endcsname
   \ifoptverbose\space\csname cmt@#1\endcsname\fi
  }%
 \fi
 \mes{\spsp You have selected: \csname txt@#1\endcsname}%
}
```

\wrxopt Writing the unused options to the batch job file is done with the \wrxopt command, which is heavily controlled by the flags \ifoptlist and \ifoptverbose.

```
\def\wrxopt#1{%
 \edef\tempx{\csname opt@#1\endcsname}%
 \if!\tempx!
  \wr{\pc\space\spsp\pc: (def)
   \csname txt@#1\endcsname
   \ifoptverbose\space\csname cmt@#1\endcsname\fi
  }%
 \else
  \wr{\pc\space\tempx,\pc:
   \csname txt@#1\endcsname
   \ifoptverbose\space\csname cmt@#1\endcsname\fi
  }%
 \fi
}
```

\beginoptiongroup \endoptiongroup One can structure the master file using the commands \beginoptiongroup

. . . `\endoptiongroup`. The `\beginoptiongroup` command takes two arguments, the prompt text and a control expression.

```
\beginoptiongroup
  {CITATION NUMBERS (if numerical references)}
  {\ifnumerical*\fi}
\optdef{*}{}{arabic numbers}
  {references are numbered 1, 2, 3, etc.}
\optdef{d}{d'nai}{d'nai numerals}
  {references are numbered base-25}
\getans
more commands and option groups
\endoptiongroup
```

In the above example, the master file has defined a `\newif` switch called `\ifnumerical`, and now uses this flag to enable the processing encapsulated within the option group: the control expression executes the active `*` command if `\ifnumerical` is true. More complex expressions are possible; use plain TEX constructions to expand the star.

The prompt text is output to the console and is also recorded as a comment in the generated `.dbj` file.

If the control expression executes the active `*`, then the statements within the option group are executed as usual. If false, then the `.dbj` file will simply contain a record of the options that the user would have been able to chose from. In effect, the interrogation never comes: all the options are unused and are recorded (via `\wrxopt`) as comments.

By this means, one can structure the `.dbj` file so that all options are made visible, even if some of them would not be accessible because of internal dependencies. The `.dbj` file will show as much detail about the menus of the master file as is possible.

To enable a common idiom, we have caused the value of `\ans` to persist past the end of the option group. This means that one may safely test the value of `\ans` after the `\endoptiongroup`. If processing was turned off within the option group, then the value of `\ans` is the untypeable $\$_{12}$.

If the second argument is either nil or is a star, then the option group will be executed normally. Therefore you can employ this structure for all the processing involving the commonly used idiom. If the master file has statements like:

```
\mes{PROMPT:}
\optdef{*}{}{default}{extended comment}%
\optdef{a}{opt-a}{option a}{another extended comment}%
\optdef{b}{opt-b}{option b}{more extended comments}%
\getans
\if\ans a\whatever\fi
```

they should be converted to:

```
\beginoptiongroup{PROMPT:}{}
\optdef{*}{}{default}{extended comment}%
\optdef{a}{opt-a}{option a}{another extended comment}%
\optdef{b}{opt-b}{option b}{more extended comments}%
\getans
\endoptiongroup
\if\ans a\whatever\fi
```

The benefit of this syntax is a single markup for all interrogations and a consistent appearance in the generated file.

```
\def\beginoptiongroup#1{\begingroup\activatestar\OGcontinue{#1}}%
\def\OGcontinue#1#2{%
 \endgroup
 \begingroup
  \let\OGswitch\secondoftwo\def\tempa{#2}%
  \ifx\tempa\empty\expandafter\firstoftwo\else\expandafter\secondoftwo\fi
  {%
   \activestar
  }{%
   \tempa
  }%
  \OGswitch{}{%
    \let\wropt\wrxopt
    \let\ask\nilans
    \def\mes##1{}%
  }%
  \def\OGmessage{#1}%
  \umes{\ifoptverbose<<\fi\OGmessage}%
}
\def\endoptiongroup{%
  \ifoptverbose\umes{>>\OGmessage}\fi
  \aftergroup\let\aftergroup\ans\expandafter
 \endgroup
 \ans
}
\def\activestar{\let\OGswitch\firstoftwo}
\def\activatestar{\catcode`\*13\relax}
{\activatestar\gdef*{\activestar}}
\def\firstoftwo#1#2{#1}
\def\secondoftwo#1#2{#2}
{\catcode`\$=12\gdef\nilans#1#2{\def\ans{$}}}
```

For more examples of using option groups, see the file `merlin.mbs`.

6.3 Initial Messages

The program can now start working. It first introduces itself and asks if the user wants an explanation of how the menus work.

```
\mes{**********************************^^J%
     * This is Make Bibliography Style *^^J%
     **********************************^^J%
     It makes up a docstrip batch job to produce^^J%
     a customized .bst file for running with BibTeX}

\ask{\yn}{Do you want a description of the usage? (NO)}

\if!\yn!\else\if\yn n\else\if\yn N\else
\mes{In the interactive dialogue that follows,^^J%
     you will be presented with a series of menus.^^J%
     In each case, one answer is the default, marked as (*),^^J%
     and a mere carriage-return is sufficient to select it.^^J%
     (If there is no * choice, then the default is the last choice.)^^J%
     For the other choices, a letter is indicated^^J%
     in brackets for selecting that option. If you select^^J%
     a letter not in the list, default is taken.^^J^^J%
     The final output is a file containing a batch job^^J%
     which may be (La)TeXed to produce the desired BibTeX^^J%
     bibliography style file. The batch job may be edited^^J%
     to make minor changes, rather than running this program^^J%
     once again.}
\fi\fi\fi
```

Ask for the name of the master bibliographic style file, suggesting a default name. Test if the file exist (argument `i`). The name of the master file is split up into root and extension.

```
\MBaskfile{^^JEnter the name of the MASTER file}(merlin.mbs)i\mfile
\let\mroot=\froot
\let\mext=\fext
```

Originally, I intended the menu information to be in an `.opt` file, but this is dangerous: that file may not be consistent with the master. So now, issue a warning if an `.opt` file exists, substituting it only if explicitly requested. (This is useful for me when testing changes to `makebst` and I only want a short menu.)

```
\edef\temp{\mroot.opt}
\openin\infile\temp\relax
\let\mnext=\mext
\ifeof\infile\else
  \ask{\yn}{** Warning: a file `\temp' also exists^^J
       \spsp Shall I read it for the menu information? (NO)^^J
       \spsp (Answer `yes' only if you really know what you are doing)}
\if\yn y\def\mnext{opt}\else\if\yn Y\def\mnext{opt}\fi\fi
\mes{Menu information read from `\mroot.\mnext'}
```

```
198 \fi
199 \closein\infile
```

Next, the name of the output `.bst` file is asked for. Here there is to be no default for the root part, although the extension defaults to `.bst`.

```
200 \MBaskfile{^^JName of the final OUTPUT .bst file?}(.bst)o\ofile
201 \let\oroot=\froot
202 \let\oext=\fext
```

A comment line of text is asked for. This will go into the preamble of the final `.bst` file and should describe the nature of the bibliographic style, i.e., for which journal(s) it is meant to apply.

```
203 \ask{\ans}{^^JGive a comment line to include in the style file.^^J%
204           Something like for which journals it is applicable.}
```

The output batch job file is to have the same root name as the output file, but with the extension `.dbj`, for *docstrip batch job*. This file is opened and the initial contents are written.

```
205 \immediate\openout\outfile\oroot.dbj
206 \wr{\pcpc Driver file to produce \oroot.\oext\space from \mroot.\mext}
207 \wr{\pcpc Generated with \filename, version \fileversion\space (\filedate)}
208 \wr{\pcpc Produced on \Now}
209 \wr{\pcpc}
210 \wr{\string\input\space docstrip}
211 \wr{}
212 \wr{\string\preamble}
213 \wr{---------------------------------------}
214 \wr{*** \ans\space ***}
215 \wr{}
216 \wr{\string\endpreamble}
217 \wr{}
218 \wr{\string\postamble}
219 \wr{End of customized bst file}
220 \wr{\string\endpostamble}
221 \wr{}
222 \wr{\string\keepsilent}
223 \wr{}
224 \wr{\string\askforoverwritefalse}
```

The options will be written to the output file during the interrogation when the master file is input. These options are stored in `\MBopts`.

Note: it is necessary to change the catagory codes of `{` and `}` temporarily, and to find substitutes, so that mismatched curly braces could be included in the output text. The same thing is done again at the end to close the braces finally. This is done with `\MBswitch`.

```
225 \begingroup\MBswitch
226 \wr(\string\def\string\MBopts{\string\from{\mroot.\mext}{\pc)
```

```
\endgroup
```

Now each selected option is written on a single line.

6.4 The Interrogation

The menu information is read in from the master file, or from a file with extension `.opt`, but only if one has explicitly requested this. (This is expert stuff; the `.opt` files should be avoided since they might not be up-to-date. Previously they were the default, but this has been changed in version 2.1 to avoid confusion.)

```
\newif\ifoptlist
⟨optlist | optverbose⟩\optlisttrue
\ifoptlist\else
 \ask\yn{Do you want the unused option lines^^J%
         \spsp to appear as comments in the output? (NO)}
 \if!\yn!\else\if\yn n\else\if\yn N\else\optlisttrue\fi\fi\fi
\fi
\newif\ifoptverbose
⟨optverbose⟩\optverbosetrue
\ifoptlist
 \ifoptverbose\else
  \ask\yn{Do you want verbose comments? (NO)}
  \if!\yn!\else\if\yn n\else\if\yn N\else\optverbosetrue\fi\fi\fi
 \fi
\fi
\edef\temp{\mroot.\mnext}
\let\endoptions=\endinput
\input\temp
```

Note that it is necessary to equate `\endoptions` to `\endinput` in case the master file is read in. An `\endinput` command in the master file would interfere with the docstrip operation, but this indirect method gets around that problem.

7 Closing the Output File

The output file is closed by writing the final line that closes the braces that were opened at the beginning. To this end, the catagory codes of `{` and `}` must be temporarily altered, as before.

```
\begingroup\MBswitch
\wr(\spsp}})
\endgroup
```

Now write the line that processes the options stored in `\MBopts`. The batch job file is finished and may be closed.

```
\wr{\string\generate{\string\file{\oroot.\oext}{\string\MBopts}}}
```

```
\wr{\string\endbatchfile}

\immediate\closeout\outfile
\mes{^^JFinished!!^^J%
     Batch job written to file `\oroot.dbj'}
```

The batch job may now be run. It is only necessary to input the file. However, the inputting should not occur with a group or within an `\if ... \fi` clause. Furthermore, under LaTeX, the `\end` command causes problems, because it has been redefined; the command `\@@end` contains the original `\end`.

```
\def\ofile{\oroot.dbj}
\ask{\yn}{Shall I now run this batch job? (NO)}
\def\temp{\relax}
\if!\yn!\else\if\yn n\else\if\yn N\else
\def\temp{\input\ofile}\fi\fi\fi
{\catcode`\@=11 \ifx\@@end\undefined\else
  \global\let\end=\@@end\fi}
\temp
\end
```

⟨/program⟩