MapleV

A GNU Emacs Mode for Maple Developers
version 1.00
26 September 1999

Joseph S. Riel

Copying

The programs currently being distributed that relate to MapleV consist of GNU Emacs Elisp files. These programs are "free"; this means that everyone is free to use them and free to redistribute them on a free basis. The MapleV-related programs are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to MapleV, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the MapleV related programs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to MapleV. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to MapleV are found in the General Public Licenses that accompany them.

Introduction

MapleV is a GNU Emacs major mode for developing source code for Maple V, a computer algebra system (CAS) owned by Waterloo Maple Inc. In this manual *MapleV* refers to the Emacs major mode and *Maple* to the CAS. MapleV is written entirely in Emacs-Lisp and is distributed under the GNU General Public License.

Following is a brief tour of MapleV's major features.

Indentation

Maple source code is grammatically indented, either as you enter it or all at once. Customizable variables permit a limited control of the indentation style. The default settings produce a result that is very close to the pretty printed output of Maple.

Font Lock

Maple reserved words, special words, initial variables, builtin functions, and top-level procedure assignments are font locked. Comments and quotes are syntactically highlighted. The amount of "decoration" can be customized.

Comments

Commands are provided for inserting and aligning Maple comments. Auto-filling can be enabled so that comments automatically wrap.

Shortcuts

Abbreviations for common Maple words are defined and automatically expanded, if enabled. A blank procedure template, including your copyright statement, can be inserted into the source. It queries for the name of the procedure, optional arguments, and a description.

Mint interface

All or portions of the buffer can be sent to `mint`, Maple's syntax checker. The output is displayed in a buffer with a mode that highlights and activates warnings and error messages. Clicking on the activated text either moves the cursor to the appropriate point in the source code or queries to automatically correct the error.

Maple interface

All or portions of the buffer can be sent to the command line version of Maple, which is run in its own buffer. You can work directly in that buffer to exercise the source code.

Online help

Help pages from Maple help databases can be called up and displayed in a buffer. The buffer has a mode that font locks section headings and provides commands for viewing other help pages and recalling previously visited pages.

Library procedures

Procedures from Maple libraries can be displayed in a buffer. They are font locked the same as in a MapleV buffer. Commands are available for displaying other procedures and a history mechanism provides a convenient means to return to previously displayed procedures.

Multiple Maple releases

At installation MapleV is configured to work with a default release of Maple. You may also specify alternate releases of Maple. A MapleV buffer can then be configured to work with a release different than the default; it will access the versions of Maple and Mint appropriate for the release.

1 Basics

For MapleV to properly locate, fontify, and index *top-level procedures*, that is, non-nested procedure assignments, the procedure name *must* be flush left. Indenting the buffer moves top-level procedures to the left margin.

There are a few exceptional cases in which what should be top-level procedures are, in fact, not. The primary example is a Maple script in which procedures are conditionally assigned. See Section 2.3.2 [Preventing indentation], page 6, for an illustration and a method to automatically indent these procedures to the left column.

Most of the higher-level MapleV functions, those that do more than edit text, are available on the menubar.

2 Indentation

Maple source code is indented according to its grammar. The indentation can occur either as you enter the code or all at once; the latter action is useful when working with non-indented source code. A grammatical error, typically an out of place keyword or parenthesis, generates an error and moves the cursor to the place where the error was detected.

2.1 Commands

⟨TAB⟩ Indent the current line (`maplev-electric-tab`).

C-j Indent the current line, insert a new line, and indent that line (`maplev-indent-newline`).

C-c ⟨TAB⟩ ⟨TAB⟩
C-c ⟨TAB⟩ *b*
Indent the buffer (`maplev-indent-buffer`).

C-c ⟨TAB⟩ *p*
Indent a procedure (`maplev-indent-procedure`).

C-c ⟨TAB⟩ *r*
Indent the region (`maplev-indent-region`).

2.2 Customizing

The following variables affect indentation:

`maplev-indent-level`
The amount a subblock is indented. The default is 4.

`maplev-indent-declaration`
The amount the Maple procedure declarations (`local`, `global`, `option`, and `description`) are indented. The default is 0.

`maplev-dont-indent-re`
A regex or nil. If non-nil then lines that begin with a match are not indented. The default, '`"#"`', prevents flush left comment lines from being indented.

2.3 Indentation Tricks

The indentation algorithm is not perfect. It can fail to indent code that should be indented or it may indent code that should not be indented. The following sections give examples and demonstrate workarounds.

2.3.1 Forcing indentation

MapleV's indentation algorithm does not (currently) handle continued expressions. It aligns continuations with the left most character in the preceding line. In an assignment it is preferable to align with the right side of the assignment.

Problem

Indenting the following code causes the continued line to be left aligned with the preceding line, as the following illustrates:

```
---------- Buffer: foo ----------
y := a + ( ... )
     + b;⋆
---------- Buffer: foo ----------

⟨TAB⟩
   ⇒
---------- Buffer: foo ----------
y := a + ( ... )
+ b;⋆
---------- Buffer: foo ----------
```

Solution

Use extra parentheses to prevent the continuation line from being aligned with the opening column:

```
---------- Buffer: foo ----------
y := ( a + ( ... )
       + b );
---------- Buffer: foo ----------
```

2.3.2 Preventing indentation

Problem

Consider an installation script in which the procedures '`foo1`' and '`foo2`' are assigned only when the flag `assign_procs` is '`true`'. The following example shows what happens when the buffer is indented.

```
---------- Buffer: foo ----------
if assign_procs then
foo1 := proc() ... end:
foo2 := proc() ... end:
fi:
---------- Buffer: foo ----------

M-x maplev-indent-buffer
   ⇒
---------- Buffer: foo ----------
if assign_procs then
    foo1 := proc() ... end:
    foo2 := proc() ... end:
fi:
---------- Buffer: foo ----------
```

Because `foo1` and `foo2` are no longer flush left they are not recognized as top-level procedures. Their names are not properly font locked and MapleV commands that operate on top-level procedures do not work on them.

Solution

Because MapleV ignores comment continuations that Maple respects (Chapter 4 [Comments], page 9), we can use the following technique to prevent '`foo1`' and '`foo2`' from being indented.

```
---------- Buffer: foo ----------
if assign_procs then       #\
fi                         # Maple does not see this line
foo1 := proc() ... end:
foo2 := proc() ... end:
#\
if then                    # Maple does not see this line
fi:
---------- Buffer: foo ----------
```

MapleV ignores the comment continuations and determines that each `if` statement is completed on the following line. The procedures `foo1` and `foo2` are not indented. Maple, however, continues the comments and so matches the initial `if` to the final `fi`; it ignores the dummy statements.

2.4 Indentation Details

A grammar table (`maplev--grammar-alist`) defines the grammar used to indent Maple code.

MapleV parses the source to compute the appropriate indentation for each line. To speed this process, information from the last parse is saved and reused. This method allows it to indent entire buffers reasonably quickly; the largest file in the Maple R5 share library ('`gdev.mpl`', 160K, by Bruno Salvy) took twelve seconds to indent on a PC running NTEmacs. During editing, if the buffer is modified above the last indentation location then the indentation information is lost; consequently, you may occasionally notice small delays as the source is reparsed.

3 Font Lock

3.1 Decoration level

The amount of syntactical highlighting, or "decoration", is controlled by the global variable `font-lock-maximum-decoration`, which you may set in your '`.emacs`' file. See Info file '`emacs`', node '`Font Lock`', for information. MapleV mode provides three levels of decoration:

1. Comments, quotes, top-level procedure names and Maple reserved works are highlighted.
2. Everything in level 1 plus Maple special words, initial variables, and the ditto operators are highlighted.
3. Everything in level 2 plus Maple builtin functions are highlighted.

Execute *M-x maplev-reset-font-lock* ⟨RET⟩ *LEVEL* ⟨RET⟩ or use the menubar, *MapleV -> Setup -> Decoration*, to change the decoration in a MapleV buffer. `LEVEL` is an integer from 1 to 3.

3.2 Adding keywords

You can use the usual method to add new keywords to font lock in MapleV mode. For example, the following snippet can be added to your '`.emacs`' file to font lock '`simplify`' and '`printf`' in MapleV mode.

```
(font-lock-add-keywords
 'maplev-mode
 '(("simplify" . maplev-font-special-word-face)
   ("printf"   . maplev-font-special-word-face)))
```

3.3 Display faces

`maplev-special-word-face`
: Display face used for Maple special words. The special words are '`args`', '`nargs`', '`procname`', '`RootOf`' and '`Float`'.

`maplev-initial-variable-face`
: Display face used for Maple initial variables. These are '`Catalan`', '`true`', '`false`', '`FAIL`', '`infinity`', '`Pi`', '`gamma`', '`integrate`', '`libname`', '`NULL`', '`Order`', '`printlevel`', and '`lasterror`'.

4 Comments

MapleV uses standard Emacs commands to enter, align and fill Maple comments. See Info file '`emacs`', node '`Comments`'. The commands are reproduced here for convenience.

M-; Insert or align an inline comment (`indent-for-comment`). The comment character is inserted at column `comment-column`.

C-x ; Set comment column (`set-comment-column`).

C-u - C-x ;
Kill comment on current line (`kill-comment`).

M-q Fill a comment (`fill-paragraph`). Wrap lines at column `fill-column` and insert new comment characters, aligned with the original comment character.

The following variables affect comments:

`maplev-auto-fill-comment-flag`
A boolen flag. If non-nil, the default, comment lines wrap as they are typed. Wrapping, however, does not automatically start in an inline comment; it must be invoked with `fill-paragraph`.

`maplev-comment-string`
String variable inserted by `indent-for-comment`. The default is '`# `'.

`maplev-comment-column`
Initial value of `comment-column`. The default is 40.

`maplev-comment-fill-column`
Initial value of `fill-column`. The default is 79.

Maple comment lines can be continued to the next line by ending them with a backslash. MapleV does *not* recognize this continuation and interprets the following line as code. This can fool the MapleV indentation grammar; however, it can also be used to achieve certain effects. See Section 2.3.2 [Preventing indentation], page 6, for an example.

5 Shortcuts

5.1 Abbreviations

Abbreviations are available for common or lengthy Maple keywords. They are expanded whenever `abbrev-mode` is active. See Info file '`emacs`', node '`Abbrevs`'. The command `maplev-abbrev-help` displays a list of the available abbreviations.

The following variables affect the expansion of abbreviations:

`maplev-initial-abbrev-mode-flag`
: If non-nil `abbrev-mode` is activated when MapleV is started. The default is '`t`'.

`maplev-expand-abbrevs-in-comments-and-strings-flag`
: If non-nil then the Maple abbreviations are expanded in comments and strings. The default is '`nil`'.

5.1.1 Customizing Abbreviations

The predefined MapleV abbreviations are stored in the abbreviation table `maplev-mode-abbrev-table`. The following code may be added to your '`.emacs`' file to assign '`simp`' as an abbreviation for '`simplify`'.

```
(define-abbrev maplev-mode-abbrev-table
        "simp" "simplify" 'maplev--abbrev-hook)
```

The function '`'maplev--abbrev-hook`' prevents the abbreviation from being expanded inside a comment or quote.

To remove an abbreviation from the table assign it `nil`. For example, to prevent '`lib`' from expanding to '`libname`', add the following to '`emacs`':

```
(define-abbrev maplev-mode-abbrev-table "lib" nil nil)
```

5.2 Templates

C-c C-p Insert a procedure template (`maplev-proc-template`). The user is queried for the name, arguments, and a description of the procedure. Any of the entries can be left blank. If the name is blank then an anonymous procedure is inserted, otherwise an assignment is inserted with the procedure assigned to the given name. Backquotes are added automatically to procedure names if required by Maple.

C-; Insert an assignment operator at the end of the current line (`maplev-insert-assignment-operator`).

The following variables affect the shortcuts:

`maplev-insert-copyright-flag`
: If non-nil then a copyright notice is inserted in the `option` declaration of the procedure template. The default is `t`.

`maplev-copyright-owner`
: String inserted as the copyright owner.

`maplev-comment-end-flag`
: If non-nil then the name of the procedure is inserted as a comment to the right of the closing `end` statement.

`maplev-assigment-operator`
: The string inserted by `maplev-insert-assignment-operator`. The default value is ‘ `:=` ’.

6 Imenu support

Executing *maplev-add-imenu* or selecting *MapleV -> Add Index* from the menubar creates an indexed menu of the top-level Maple procedures, global variables, and macro assignments. The menu appears under the 'Index' heading in the menubar. Clicking on an item in the menu moves point to the assignment of that item.

The assignments must be flush left to be indexed. Only the first macro in a `macro` assignment is indexed.

7 Mint

Mint is Maple's syntax checker. It analyzes a Maple program and produces a report about the syntax and variable usage. MapleV can run mint on the entire buffer or a portion of it. The output of mint is displayed in a buffer with a special mode, `mint-mode`, that provides a convenient means for locating and correcting syntax errors.

7.1 Running mint

The following commands send source code in the buffer to Mint:

C-c ⟨RET⟩ *b*
: Run Mint on the buffer (`mint-buffer`).

C-c ⟨RET⟩ *p*
: Run Mint on the current procedure (`mint-procedure`).

C-c ⟨RET⟩ *r*
: Run Mint on the marked region (`mint-region`).

C-c ⟨RET⟩ ⟨RET⟩
: Rerun the previous Mint command (`mint-rerun`).

These commands are available through the menubar, *MapleV -> Mint*. The following variables affect the output of Mint:

`mint-info-level`
: An integer from 0 to 4 that selects the amount of information displayed by Mint. 0 displays no information, 4 displays the most. The default value is 3. This value can be set throught the menubar, *Maplev -> Mint -> Mint level*.

`mint-start-options`
: A string passed to Mint at startup. The default is '`"-q"`', which suppresses the display of the Maple logo. Type *?mint* in Maple for other options.

`mint-coding-system`
: Symbol that defines the coding system used by Mint. The default value is `undecided-dos`.

7.2 Mint mode

Mint mode is applied to mint's output buffer. Warnings and errors are font locked and activated. Moving the mouse pointer over active text highlights it; clicking it (*mouse-2*) either moves the cursor to the appropriate point in the source code or queries to automatically correct an error.

The following commands are available:

s Incremental forward search (`isearch-forward`).

r Incremental backward search (`isearch-backward`).

⟨RET⟩ Reexecute the previous mint command (`mint-rerun`).

⟨DEL⟩ Scroll down (`scroll-down`).

⟨SPC⟩ Scroll up (`scroll-up`).

`mouse-2` Goto location in source, or fix error, depending on the active text.

The following variables set the display faces for the highlighted text in the Mint buffer:

`mint-proc-face`
: Face for procedure names.

`mint-warning-face`
: Face for warnings.

`mint-error-face`
: Face for errors.

`mint-note-face`
: Face for notes (usually '`on line`').

8 Maple

The command line version of Maple can be started in a buffer. All or portions of the code in the MapleV buffer can be passed directly to the Maple process. Maple commands can be directly executed in the buffer.

8.1 Running Maple

The following commands in the MapleV buffer affect the Maple engine:

`C-c C-c b` Send the entire buffer to the Maple engine (`cmaplev-send-buffer`).

`C-c C-c p` Send the current procedure to the Maple engine (`cmaplev-send-procedure`).

`C-c C-c r` Send the marked region to the Maple engine (`cmaplev-send-region`).

`C-c C-c g` Goto the Maple buffer (`cmaplev-goto-buffer`).

`C-c C-c i` Interrupt the Maple engine (`cmaplev-interrupt`).

`C-c C-c k` Kill the Maple engine (`cmaplev-kill`).

These commands are available through the menubar, *MapleV -> Maple*.

8.2 Cmaple mode

The command line version of Maple is run in a buffer with the mode `cmaple-process-mode` that is based on `comint-mode`. In addition to the normal `comint` commands, the following commands are available:

`?`
`C-?` Display a Maple help topic (see Chapter 9 [Help pages], page 16).

`M-?` Display a Maple procedure (see Chapter 10 [Procedures], page 17).

9 Help pages

Help pages can be read from the Maple help databases and displayed in a buffer with major mode `maplev-help-mode`. Text in the buffer is highlighted and cross references are activated.

9.1 Displaying help pages

The following commands display Maple help pages:

C-? Query for a help topic, using the word at point as a default. Display the help page in a buffer (`maplev-help-at-point`).

S-mouse-2
Display the Maple help page for the topic at the click (`maplev-help-follow-mouse`).

Help pages are displayed in a buffer with major mode `maplev-help-mode`.

9.2 MapleV help mode

The major mode `maplev-help-mode` is active in the buffer that displays Maple help pages. Section headers are font locked and text in the '`See Also`' section is activated so that clicking on it opens the help page for the topic. The following commands are available:

s Incremental forward search (`isearch-forward`).

p Previous help topic (`maplev-prev-item`).

n Next help topic (`maplev-next-item`).

P Parent help topic (`maplev-help-parent`).

r Redraw help page (`maple-redo-item`).

?
C-?
⟨RTN⟩ Query for a help topic (`maplev-help-at-point`).

M-? Query for a procedure (`maplev-proc-at-point`).

⟨SPC⟩ Scroll down.

⟨DEL⟩ Scroll up.

MapleV help mode keeps a history of the help topics displayed. Use the command `maplev-clear-history` to erase the history.

The help page for a chosen topic is displayed by sending the string '`?TOPIC`' to the Maple engine and capturing the output. If the Maple engine is busy an error message, '`Maple busy`', is displayed in the message window.

10 Procedures

Procedures can be read from the active Maple libraries and displayed in a buffer with major mode `maplev-proc-mode`. The code is font locked the same as in MapleV mode.

10.1 Displaying procedures

The following commands display Maple procedures:

M-? Query for a procedure name, using the word at point as the default. Read the procedure from the Maple library and display it in a buffer (`maplev-proc-at-point`).

M-S-mouse-2
Read the procedure at the click from the library and display it in a buffer (`maplev-proc-follow-point`).

Procedures are displayed in a buffer with major mode `maplev-proc-mode`.

10.2 MapleV proc mode

The major mode `maplev-proc-mode` is active in the buffer that displays Maple procedures read from a Maple library. It font locks the procedure, highlighting keywords the same as MapleV mode does. Clicking on procedure names in the buffer displays their source code or opens a help page for them. A history mechanism stores the previously displayed procedure.

The following commands are available:

s Incremental forward search (`isearch-forward`).

p Previous procedure (`maplev-prev-item`).

n Next procedure (`maplev-next-item`).

r Redraw procedure (`maple-redo-item`).

?
C-?
⟨RTN⟩ Query for a help topic (`maplev-help-at-point`).

M-? Query for a procedure (`maplev-proc-at-point`).

⟨SPC⟩ Scroll down.

⟨DEL⟩ Scroll up.

MapleV help mode keeps a history of the help topics displayed. Use the command `maplev-clear-history` to erase the history.

A procedure is read from a library and displayed by using the Maple procedure '`maplev_print`' that is assigned when the Maple engine is started. If the Maple engine is busy an error message, '`Maple busy`', is displayed in the message window.

Appendix A Installation

This section describes how to install MapleV into GNU Emacs.

A.1 Compiling

Move the file '`maplev.el`' into your Emacs load path and byte compile it as shown below:

M-x byte-compile-file ⟨RET⟩ `maplev.el` ⟨RET⟩

Add the following line to your '`.emacs`' file:

```
(autoload 'maplev-mode "maplev" "Maple editing mode" t)
```

To have Emacs automagically start in MapleV mode when editing Maple source, add the following to your '`.emacs`' file, modifying the regex '`.mpl`' to an extension appropriate for your usage:

```
(setq auto-mode-alist
      (cons '("\\.mpl\\'" . maplev-mode) auto-mode-alist))
```

A.2 Customizing

You must customize some of MapleV's default settings to be appropriate for your installation. Most significantly, you must specify the locations of the executable files for mint and the command line version of Maple. You can specify multiple versions of mint and Maple. The easiest method is to invoke `customize` using the following commands:

```
M-x load-library ⟨RET⟩ maplev ⟨RET⟩
M-x customize-group ⟨RET⟩ maplev ⟨RET⟩
```

The important options are in the subgroup `maplev-important`. After setting these options, save them to your '`.emacs`' file by clicking on the '`Save for Future Sessions`' button.

A.3 Info documentation

To create the Info documentation for MapleV, convert the TeXinfo file '`maplev.texi`' to an Info file. You may use either the stand-alone utility `makeinfo` or, from inside Emacs, the command `makeinfo-buffer`.

Move the output file '`maplev`' to a directory in the Info load path and then edit the '`dir`' file, that is, the top level node of your Emacs Info structure, to point to '`maplev`'. I added the following menu item to my '`dir`' file:

```
* MapleV: (maplev).       MapleV reference manual.
```

Appendix B Evolution

B.1 Bugs

If you encounter a bug in this package, wish to suggest an enhancement, or want to make a smart remark, then send an email to me, the humble developer.

Joseph S. Riel (Joe Riel) '`joer@k-online.com`'

B.2 Acknowledgements

I'd like to thank a number of people who have contributed, either directly or indirectly, to this package.

Bruno Salvy
: For writing `maple-mode`, a small but useful Emacs mode for editing Maple code.

Michael Smith
: For writing `Gap-mode` and `Gap-process`. These gave me the idea, and showed me how, to display help pages. Displaying source code from the Maple libraries was a natural extension. `Gap` is a CAS specialized for group theory.

Nicholas Thiéry
: For writing `Maple-mode`, another Emacs mode for editing Maple code. It introduced the idea of using a grammar to indent Maple source code.

Bob Glickstein
: For writing *Writing GNU Emacs Extensions*. It allowed me, a novice Elisp programmer, to put it all together.

Christian Pomar
: For courageously agreeing to test a series of alpha versions of this package. He found numerous errors and suggested many improvements.

B.3 Enhancements

The following is a short list of features that I am tentatively planning to add to MapleV.

- Source code debugger. The Maple debugger `DEBUG` provides a useful means to step through code; its interface, however, leaves much to be desired. A more convenient interface would be similar to that of `Edebug`, the Emacs-Lisp source code debugger.
- LaTeX support. I use MapleDoc, a LaTeX macro package that I wrote, for documenting Maple source code. To facilitate its use MapleV should be able to font-lock LaTeX keywords in comments. This will be an optional package.

Key Index

Function Index

Variable Index

Concept Index

P

R

S

T

W

Short Contents

Table of Contents