

Emacs Auto-Overlays Manual

Version 0.10

Toby Cubitt

This manual describes the Emacs Auto-Overlays package, version 0.10

Copyright © 2007, 2008 Toby Cubitt

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Table of Contents

1	Overview	1
2	Auto-Overlay Functions	3
2.1	Defining Regexps.....	3
2.2	Starting and Stopping Auto-Overlays.....	5
2.3	Searching for Overlays.....	6
3	Worked Example	8
4	Extending the Auto-Overlays Package	19
4.1	Auto-Overlays in Depth.....	19
4.2	Integrating New Overlay Classes	20
4.3	Functions for Writing New Overlay Classes	21
4.3.1	Standard Parse and Suicide Functions	21
4.3.2	Functions for Modifying Overlays.....	21
4.3.3	Functions for Querying Overlays.....	22
4.4	Auto-Overlay Hooks	23
4.5	Auto-Overlay Modification Pseudo-Hooks	23
5	To-Do	25
Appendix A	Function Index	26
Appendix B	Variable Index	27
Appendix C	Concept Index	28
Appendix D	Copying this Manual	30
D.1	GNU Free Documentation License	30
D.1.1	ADDENDUM: How to use this License for your documents	36

1 Overview

The auto-overlays package automatically creates, updates and destroys overlays based on regular expression matches in the buffer text. The overlay is created when text is typed that matches an auto-overlay regexp, and is destroyed if and when the matching text is changed so that it no longer matches.

The regexps are grouped into sets, and any number of different sets of regexps can be active in the same buffer simultaneously. Regexps in different sets are completely independent, and each set can be activated and deactivated independently (see [Section 2.1 \[Defining Regexps\], page 3](#)). This allows different Emacs modes to simultaneously make use of auto-overlays in the same buffer.

There are different “classes” of auto-overlay, used to define different kinds of overlay behaviour. Some classes only require a single regexp, others require separate regexps to define the start and end of the overlay (see [Section 2.1 \[Defining Regexps\], page 3](#)). Any additional regexps, beyond the minimum requirements, act as alternatives; if more than one of the regexps matches overlapping regions of text, the one that appears earlier in the list will take precedence. The predefined regexp classes are: `word`, `line`, `self`, `nested` and `flat`, but the auto-overlay package can easily be extended with new classes.

- | | |
|---------------------|--|
| <code>word</code> | These are used to define overlays that cover the text matched by the regexp itself, so require a single regexp. An example use would be to create overlays covering single words. |
| <code>line</code> | These are used to define overlays that stretch from the text matching the regexp to the end of the line, and require a single regexp to define the start of the overlay. An example use would be to create overlays covering single-line comments in programming languages such as c. |
| <code>self</code> | <p>These are used to define overlays that stretch from one regexp match to the next match for the same regexp, so naturally require a single regexp. An example use would be to create overlays covering strings delimited by <code>"</code>.</p> <p>Note that for efficiency reasons, <code>self</code> overlays are <i>not</i> fully updated when a new match is found. Instead, when a modification is subsequently made at any position in the buffer after the new match, the overlays are updated <i>up to</i> that position. The update occurs just <i>before</i> the modification is made. Therefore, the overlays at a given buffer position will not necessarily be correct until a modification is made at or after that position (see Chapter 5 [To-Do], page 25).</p> |
| <code>nested</code> | These are used to define overlays that start and end at different regexp matches, and that can be nested one inside another. This class requires separate start and end regexps. An example use would be to create overlays between matching braces <code>{}</code> . |
| <code>flat</code> | These are used to define overlays that start and end at different regexp matches, but that can not be nested. Extra start matches within one of these overlays are ignored. This class requires separate start and end regexps. An example use would be to create overlays covering multi-line comments in code, e.g. <code>c++</code> block comments delimited by <code>/*</code> and <code>*/</code> . |

By default, the entire text matching a regexp acts as the “delimiter”. For example, a `word` overlay will cover all the text matching its regexp, and a `nested` overlay will start at the end of the text matching its start regexp. Sometimes it is useful to be able to have only part of the regexp match act as the delimiter. This can be done by grouping that part of the regexp (see [Section 2.1 \[Defining Regexprs\], page 3](#)). Overlays will then start and end at the text matching the group, instead of the text matching the entire regexp.

Of course, automatically creating overlays isn’t much use without some way of setting their properties too. Overlay properties can be defined along with the regexp, and are applied to any overlays created by a match to that regexp. Certain properties have implications for auto-overlay behaviour.

priority This is a standard Emacs overlay property (see [section “Overlay Properties” in GNU Emacs Lisp Reference Manual](#)), but it is also used to determine which regexp takes precedence when two or more regexps in the same auto-overlay definition match overlapping regions of text. It is also used to determine which regexp’s properties take precedence for overlays that are defined by separate start and end matches.

exclusive

Normally, different auto-overlay regexps coexist, and act completely independently of one-another. However, if an auto-overlay has non-nil `exclusive` and `priority` properties, regexp matches within the overlay are ignored if they have lower priority. An example use is ignoring other regexp matches within comments in code.

2 Auto-Overlay Functions

To use auto-overlays in an Emacs package, you must load the overlay classes that you require by including lines of the form

```
(require 'auto-overlay-class)
```

near the beginning of your package, where *class* is the class name. The standard classes are: `word`, `line`, `self`, `nested` and `flat` (see [Chapter 1 \[Overview\]](#), page 1), though new classes can easily be added (see [Chapter 4 \[Extending the Auto-Overlays Package\]](#), page 19).

Sometimes it is useful for a package to make use of auto-overlays if any are defined, without necessarily requiring them. To facilitate this, the relevant functions can be loaded separately from the rest of the auto-overlays package with the line

```
(require 'auto-overlay-common)
```

This provides all the functions related to searching for overlays and retrieving overlay properties. See [Section 2.3 \[Searching for Overlays\]](#), page 6. Note that there is no need to include this line if any auto-overlay classes are `required`, though it will do no harm.

This section describes the functions that are needed in order to make use of auto-overlays in an Emacs package. It does *not* describe functions related to extending the auto-overlays package. See [Chapter 4 \[Extending the Auto-Overlays Package\]](#), page 19.

2.1 Defining Regexps

An auto-overlay definition is a list of the form:

```
(class &optional :id entry-id regexp1 regexp2 ...)
```

class is one of the regexp classes described in the previous section (see [Chapter 1 \[Overview\]](#), page 1). The optional `:id` property should be a symbol that can be used to uniquely identify the auto-overlay definition.

Each *regexp* defines one of the regexps that make up the auto-overlay definition. It should be a list of the form

```
(rgxp &optional :edge edge :id subentry-id @rest property1 property2 ...)
```

The `:edge` property should be one of the symbols `'start` or `'end`, and determines which edge of the auto-overlay this regexp corresponds to. If `:edge` is not specified, it is assumed to be `'start`. Auto-overlay classes that do not require separate `start` and `end` regexps ignore this property. The `:id` property should be a symbol that can be used to uniquely identify the regexp. Any further elements in the list are cons cells of the form `(property . value)`, where *property* is an overlay property name (a symbol) and *value* its value. In its simplest form, *rgxp* is a single regular expression.

If only part of the regexp should act as the delimiter (see [Chapter 1 \[Overview\]](#), page 1), *rgxp* should instead be a cons cell:

```
(rx . group)
```

where *rx* is a regexp that contains at least one group (see [section “Regular Expressions” in GNU Emacs Lisp Reference Manual](#)), and *group* is an integer identifying which group should act as the delimiter.

If the overlay class requires additional groups to be specified, *rgxp* should instead be a list:

```
(rx group0 group1 ...)
```

where *rx* is a regexp. The first *group0* still specifies the part that acts as the delimiter, as before. If the entire regexp should act as the delimiter, *group0* must still be supplied but should be set to 0 (meaning the entire regexp). None of the standard classes make use of any additional groups, but extensions to the auto-overlays package that define new classes may. See Chapter 4 [Extending the Auto-Overlays Package], page 19.

The following functions are used to load and unload regexp definitions:

```
(auto-overlay-load-definition set-id definition &optional pos)
```

Load a new auto-overlay *definition*, which should be a list of the form described above, into the set identified by the symbol *set-id*. The optional parameter *pos* determines where in the set's regexp list the new regexp is inserted. If it is `nil`, the regexp is added at the end. If it is `t`, the regexp is added at the beginning. If it is an integer, the regexp is added at that position in the list. Whilst the position in the list has no effect on overlay behaviour, it does determine the order in which regexps are checked, so can affect efficiency.

```
(auto-overlay-load-regexp set-id entry-id regexp &optional pos)
```

Load a new *regexp*, which should be a list of the form described above, into the auto-overlay definition identified by the symbol *entry-id*, in the set identified by the symbol *set-id*. *regexp* should be a list of the form described above. The optional *pos* determines the position of the regexp in the list of regexps defining the auto-overlay, which can be significant for overlay behaviour since it determines which regexp takes precedence when two match the same text.

```
(auto-overlay-unload-set set-id)
```

Unload the entire regexp set identified by the symbol *set-id*.

```
(auto-overlay-unload-definition set-id entry-id)
```

Unload the auto-overlay definition identified by the symbol *entry-id* from the set identified by the symbol *set-id*.

```
(auto-overlay-unload-regexp set-id entry-id subentry-id)
```

Unload the auto-overlay regexp identified by the symbol *subentry-id* from the auto-overlay definition identified by the symbol *entry-id* in the set identified by the symbol *set-id*.

```
(auto-overlay-share-regexp-set set-id from-buffer @optional to-buffer)
```

Share the set of regexp definitions identified by the symbol *set-id* in buffer *from-buffer* with the buffer *to-buffer*, or the current buffer if *to-buffer* is null. The regexp set becomes common to both buffers, and any changes made to it in one buffer, such as loading and unloading regexp definitions, are also reflected in the other buffer. However, the regexp set can still be enabled and disabled independently in both buffers. The same regexp set can be shared between any number of buffers. To remove a shared regexp set from one of the buffers, simply unload the entire set from that buffer using `auto-overlay-unload-regexp`. The regexp set will remain defined in all the other buffers it was shared with.

2.2 Starting and Stopping Auto-Overlays

A set of regexps is not active until it has been “started”, and can be deactivated by “stopping” it. When a regexp set is activated, the entire buffer is scanned for regexp matches, and the corresponding overlays created. Similarly, when a set is deactivated, all the overlays are deleted. Note that regexp definitions can be loaded and unloaded whether the regexp set is active or inactive, and that deactivating a regexp set does *not* delete its regexp definitions.

Since scanning the whole buffer for regexp matches can take some time, especially for large buffers, auto-overlay data can be saved to an auxiliary file so that the overlays can be restored more quickly if the same regexp set is subsequently re-activated. Of course, if the text in the buffer is modified whilst the regexp set is disabled, or the regexp definitions differ from those that were active when the overlay data was saved, the saved data will be out of date. Auto-overlays automatically checks if the text has been modified and, if it has, ignores the saved data and re-scans the buffer. However, no check is made to ensure the regexp definitions used in the buffer and saved data are consistent (see [Chapter 5 \[To-Do\], page 25](#)); the saved data will be used even if the definitions have changed.

The usual time to save and restore overlay data is when a regexp set is deactivated or activated. The auxiliary file name is then constructed automatically from the buffer name and the set-id. However, auto-overlays can also be saved and restored manually.

`(auto-overlay-start set-id @optional buffer save-file no-regexp-check)`

Activate the auto-overlay regexp set identified by the symbol *set-id* in *buffer*, or the current buffer if the latter is `nil`. If there is an file called ‘`auto-overlay-’buffer-name-’set-id`’ in the containing up-to-date overlay data, it will be used to restore the auto-overlays (*buffer-name* is the name of the file visited by the buffer, or the buffer name itself if there is none). Otherwise, the entire buffer will be scanned for regexp matches.

The string *save-file* specifies the where to look for the file of saved overlay data. If it is `nil`, it defaults to the current directory. If it is a string specifying a relative path, then it is relative to the current directory, whereas an absolute path specifies exactly where to look. If it is a string specifying a file name (with or without a full path, relative or absolute), then it overrides the default file name and/or location. Any other value of *save-file* will cause the file of overlay data to be ignored, even if it exists.

If the overlays are being loaded from a file, but optional argument `no-regexp-check` is non-`nil`, the file of saved overlays will be used, but no check will be made to ensure regexp definitions are the same as when the overlays were saved.

`(auto-overlay-stop set-id @optional buffer save-file leave-overlays)`

Deactivate the auto-overlay regexp set identified by the symbol *set-id* in *buffer*, or the current buffer if the latter is `nil`. All corresponding overlays will be deleted (unless the *leave-overlays* option is non-`nil`, which should only be used if the buffer is about to be killed), but the regexp definitions are preserved and can be reactivated later.

If *save-file* is non-`nil`, overlay data will be saved in an auxiliary file called ‘`auto-overlay-’buffer-name-’set-id`’ in the current directory, to speed up subsequent reactivation of the regexp set in the same buffer (*buffer-name* is the

name of the file visited by the buffer, or the buffer name itself if there is none). If *save-file* is a string, it overrides the default save location, overriding either the directory if it only specifies a path (relative paths are relative to the current directory), or the file name if it only specifies a file name, or both.

`(auto-overlay-save-overlays set-id @optional buffer file)`

Save auto-overlay data for the regexp set identified by the symbol *set-id* in *buffer*, or the current buffer if `nil`, to an auxiliary file called *file*. If *file* is `nil`, the overlay data are saved to a file called `'auto-overlay-'buffer-name'-set-id` in the current directory (*buffer-name* is the name of the file visited by the buffer, or the buffer name itself if there is none). Note that this is the only name that will be recognized by `auto-overlay-start`.

`(auto-overlay-load-overlays set-id @optional buffer file no-regexp-check)`

Load auto-overlay data for the regexp set identified by the symbol *set-id* into *buffer*, or the current buffer if `nil`, from an auxiliary file called *file*. If *file* is `nil`, it attempts to load the overlay data from a file called `'auto-overlay-'buffer-name'-set-id` in the current directory (*buffer-name* is the name of the file visited by the buffer, or the buffer name itself if there is none). If *no-regexp-check* is `no-nil`, the saved overlays will be loaded even if different regexp definitions were active when the overlays were saved. Returns `t` if the overlays were successfully loaded, `nil` otherwise.

2.3 Searching for Overlays

Auto-overlays are just normal Emacs overlays, so any of the standard Emacs functions can be used to search for overlays and retrieve overlay properties. The auto-overlays package provides some additional functions.

`(auto-overlays-at-point @optional point prop-test inactive)`

Return a list of overlays overlapping *point*, or the point if *point* is null. The list includes *all* overlays, not just auto-overlays (but see below). The list can be filtered to only return overlays with properties matching criteria specified by *prop-test*. This should be a list defining a property test, with one of the following forms (or a list of such lists, if more than one property test is required):

```
(function property)
(function property value)
(function (property1 property2 ...) (value1 value2 ...))
```

where *function* is a function, *property* is an overlay property name (a symbol), and *value* can be any value or lisp expression. For each overlay, first the values corresponding to the *property* names are retrieved from the overlay and any *values* that are lisp expressions are evaluated. Then *function* is called with the property values followed by the other values as its arguments. The test is satisfied if the result is non-`nil`, otherwise it fails. Tests are evaluated in order, but only up to the first failure. Only overlays that satisfy all property tests are returned.

All auto-overlays are given a non-`nil` `auto-overlay` property, so to restrict the list to auto-overlays, *prop-test* should include the following property test:

(`'identity 'auto-overlay`)

For efficiency reasons, the `auto-overlays` package sometimes leaves overlays hanging around in the buffer even when they should have been deleted. These are marked with a non-nil `inactive` property. By default, `auto-overlays-at-point` ignores these. A non-nil `inactive` will override this, causing inactive overlays to be included in the returned list (assuming they pass all property tests).

(`auto-overlays-in start end @optional prop-test within inactive`)

Return a list of overlays overlapping the region between `start` and `end`. The `prop-test` and `inactive` arguments have the same behaviour as in `auto-overlays-at-point`, above. If `within` is non-nil, only overlays that are entirely within the region from `start` to `end` will be returned, not overlays that extend outside that region.

(`auto-overlay-highest-priority-at-point @optional point prop-test`)

Return the highest priority overlay at `point` (or the point, if `point` is null). The `prop-test` argument has the same behaviour as in `auto-overlays-at-point`, above. An overlay's priority is determined by the value of its `priority` property (see section “Overlay Properties” in *GNU Emacs Lisp Reference Manual*). If two overlays have the same priority, the innermost one takes precedence (i.e. the one that begins later in the buffer, or if they begin at the same point the one that ends earlier; if two overlays have the same priority and extend over the same region, there is no way to predict which will be returned).

(`auto-overlay-local-binding symbol @optional point`)

Return the “overlay-local” binding of `symbol` at `point` (or the point if `point` is null), or the current local binding if there is no overlay binding. An “overlay-local” binding for `symbol` is the value of the overlay property called `symbol`. If more than one overlay at `point` has a non-nil `symbol` property, the value from the highest priority overlay is returned (see `auto-overlay-highest-priority-at-point`, above, for an explanation of “highest priority”).

3 Worked Example

The interaction of all the different regexp definitions, overlay properties and auto-overlay classes provided by the auto-overlay package can be a little daunting. This section will go through an example of how the auto-overlay regexps could be defined to create overlays for a subset of L^AT_EX, which is complex enough to demonstrate most of the features.

L^AT_EX is a markup language, so a L^AT_EX document combines markup commands with normal text. Commands start with ‘\’, and end at the first non-word-constituent character. We want to highlight all L^AT_EX commands in blue. Two commands that will particularly interest us are ‘\begin’ and ‘\end’, which begin and end a L^AT_EX environment. The environment name is enclosed in braces: ‘\begin{environment-name}’, and we want it to be highlighted in pink. L^AT_EX provides many environments, used to create lists, tables, titles, etc. We will take the example of an ‘equation’ environment, used to typeset mathematical equations. Thus equations are enclosed by ‘\begin{equation}’ and ‘\end{equation}’, and we would like to highlight these equations in yellow. Another example we will use is the ‘\$’ delimiter. Pairs of ‘\$’s delimit mathematical expressions that appear in the middle of a paragraph of normal text (whereas ‘equation’ environments appear on their own, slightly separated from surrounding text). Again, we want to highlight these mathematical expressions, this time in green. The final piece of L^AT_EX markup we will need to consider is the ‘%’ character, which creates a comment that lasts till the end of the line (i.e. text after the ‘%’ is ignored by the L^AT_EX processor up to the end of the line).

L^AT_EX commands are a good example of when to use `word` regular expressions (see [Chapter 1 \[Overview\], page 1](#)). The appropriate regexp definition is loaded by

```
(auto-overlay-load-definition
  'latex
  '(word ("\\\\\\[[[:alpha:]]*?\\([^[[:alpha:]]\\|\\$\\)"
         (face . (background-color . "blue")))))
```

We have called the regexp set `latex`. The `face` property is a standard Emacs overlay property that sets font properties within the overlay. See [section “Overlay Properties” in GNU Emacs Lisp Reference Manual](#). “\\\\\\” is the string defining the regexp that matches a *single* ‘\’. (Note that the ‘\’ character has a special meaning in regular expressions, so to include a literal one it must be escaped: ‘\\’. However, ‘\’ also has a special meaning in lisp strings, so both ‘\’ characters must be escaped there too, giving '\\\\.) [[:alpha:]]*? matches a sequence of zero or more letter characters. The ? ensures that it matches the *shortest* sequence of letters consistent with matching the regexp, since we want the region to end at the first non-letter character, matched by [^[[:alpha:]]. The \\| defines an alternative, to allow the L^AT_EX command to be terminated either by a non-letter character or by the end of the line (\$). See [section “Regular Expressions” in GNU Emacs Lisp Reference Manual](#), for more details on Emacs regular expressions.

However, there’s a small problem. We only want the blue background to cover the characters making up a L^AT_EX command. But as we’ve defined things so far, it will cover all the text matched by the regexp, which includes the leading ‘\’ and the trailing non-letter character. To rectify this, we need to group the part of the regexp that matches the command (i.e. by surround it with ‘\(' and ‘\)'), and put the regexp inside a `cons` cell containing the regexp in its `car` and a number indicating which subgroup to use in its `cdr`:

```
(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[:alpha:]]*?\\([[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))
```

The ‘\$’ delimiter is an obvious example of when to use a **self** regexp (see [Chapter 1 \[Overview\], page 1](#)). We can update our example to include this (note that ‘\$’ also has a special meaning in regular expressions, so it must be escaped with ‘\’ which itself must be escaped in lisp strings):

```
(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[:alpha:]]*?\\([[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))
```

```
(auto-overlay-load-definition
 'latex
 '(self ("\\$" (face . (background-color . "green")))))
```

This won’t quite work though. \LaTeX maths commands also start with a ‘\’ character, which will match the **word** regexp. For the sake of example we want the entire equation highlighted in green, without highlighting any \LaTeX maths commands it contains in blue. Since the **word** overlay will be within the **self** overlay, the blue highlighting will take precedence. We can change this by giving the **self** overlay a higher priority (any priority is higher than a non-existent one; we use 3 here for later convenience). For efficiency reasons, it’s a good idea to put higher priority regexp definitions before lower priority ones, so we get:

```
(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green")))))
```

```
(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[:alpha:]]*?\\([[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))
```

The ‘`\begin{equation}`’ and ‘`\end{equation}`’ commands also enclose maths regions, which we would like to highlight in yellow. Since the opening and closing delimiters are different in this case, we must use **nested** overlays (see [Chapter 1 \[Overview\], page 1](#)). Our example now looks like:

```
(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green")))))
```

```
(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
```

```

    (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

```

```

(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[[:alpha:]]*?\\\[[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue"))))

```

Notice how we've used separate `start` and `end` regexps to define the auto-overlay. Once again, we have had to escape the `\` characters, and increase the priority of the new regexp definition to avoid any `LATEX` commands within the maths region being highlighted in blue.

`LATEX` comments start with `%` and last till the end of the line: a perfect demonstration of a line regexp. Here's a first attempt:

```

(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green"))))

```

```

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

```

```

(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[[:alpha:]]*?\\\[[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue"))))

```

```

(auto-overlay-load-definition
 'latex
 '(line ("% (face . (background-color
        . ,(face-attribute 'default :background))))))

```

We use the standard Emacs `face-attribute` function to retrieve the default background colour, which is evaluated before the regexp definition is loaded. (This will of course go wrong if the default background colour is subsequently changed, but it's sufficient for this example). Let's think about this a bit. We probably don't want anything within a comment to be highlighted at all, even if it matches one of the other regexps. In fact, creating overlays for `\begin` and `\end` commands which are within a comment could cause havoc! If they

don't occur in pairs within the commented region, they will erroneously pair up with ones outside the comment. We need comments to take precedence over everything else, and we need them to block other regexp matches, so we boost the overlay's priority and set the exclusive property:

```
(auto-overlay-load-definition
'latex
'(line ("% (priority . 4) (exclusive . t)
        (face . (background-color
                . ,(face-attribute 'default :background))))))

(auto-overlay-load-definition
'latex
'(self ("\\$" (priority . 3) (face . (background-color . "green")))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

(auto-overlay-load-definition
'latex
'(word (("\\\\\\[[[:alpha:]]*?\\([^[[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))
```

We're well on our way to creating a useful setup, at least for the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ commands we're considering in this example. There is one last type of overlay to create, but it is the most complicated. We want environment names to be highlighted in pink, i.e. the region between `\begin{` and `}`. A first attempt at this might result in:

```
(auto-overlay-load-definition
'latex
'(line ("% (priority . 4) (exclusive . t)
        (face . (background-color
                . ,(face-attribute 'default :background))))))

(auto-overlay-load-definition
'latex
'(self ("\\$" (priority . 3) (face . (background-color . "green")))))

(auto-overlay-load-definition
'latex
```

```

'(nested
  ("\\begin{"
    :edge start
    (priority . 2)
    (face . (background-color . "pink")))
  ("}")
  :edge end
  (priority . 2)
  (face . (background-color . "pink"))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{equation}"
    :edge start
    (priority . 1)
    (face . (background-color . "yellow")))
  ("\\end{equation}"
    :edge end
    (priority . 1)
    (face . (background-color . "yellow"))))

(auto-overlay-load-definition
'latex
'(word ("\\\\"[[:alpha:]]*?\\\[^[[:alpha:]]\\|\\$\\]" . 1)
  (face . (background-color . "blue"))))

```

However, we'll hit a problem with this. The '}' character also closes the '\end{' command. Since we haven't told auto-overlays about '\end{', every '}' that should close an '\end{' command will instead be interpreted as the end of a '\start{' command, probably resulting in lots of unmatched '}' characters, creating pink splodges everywhere! Clearly, since we also want environment names between '\end{' and '}' to be pink, we need something more along the lines of:

```

(auto-overlay-load-definition
'latex
'(line ("% (priority . 4) (exclusive . t)
  (face . (background-color
    . ,(face-attribute 'default :background)))))

(auto-overlay-load-definition
'latex
'(self ("\\$" (priority . 3) (face . (background-color . "green"))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{"

```

```

      :edge start
      (priority . 2)
      (face . (background-color . "pink")))
    ("\\end{"
      :edge start
      (priority . 2)
      (face . (background-color . "pink")))
    ("}"
      :edge end
      (priority . 2)
      (face . (background-color . "pink"))))

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

(auto-overlay-load-definition
 'latex
 '(word ((("\\\\[[[:alpha:]]*?\\([^[[:alpha:]]\\|\\$\\)" . 1)
          (face . (background-color . "blue")))))

```

We still haven't solved the problem though. The '}' character doesn't only close `\begin{` and `\end{` commands in \LaTeX . *All* arguments to \LaTeX commands are surrounded by '{' and '}'. We could add all the commands that take arguments, but we don't really want to bother about any other commands (at least in this example). All we want to do is prevent predictive mode incorrectly pairing the '}' characters used for other commands. Instead, we can just add '{' to the list:

```

(auto-overlay-load-definition
 'latex
 '(line ("% (priority . 4) (exclusive . t)
        (face . (background-color
                . ,(face-attribute 'default :background))))))

(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green"))))

(auto-overlay-load-definition
 'latex

```

```

'(nested
  ("{"
    :edge start
    (priority . 2))
  ("\\begin{"
    :edge start
    (priority . 2)
    (face . (background-color . "pink")))
  ("\\end{"
    :edge start
    (priority . 2)
    (face . (background-color . "pink")))
  ("}"
    :edge end
    (priority . 2))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{equation}"
    :edge start
    (priority . 1)
    (face . (background-color . "yellow")))
  ("\\end{equation}"
    :edge end
    (priority . 1)
    (face . (background-color . "yellow")))))

(auto-overlay-load-definition
'latex
'(word ((("\\\\[[[:alpha:]]*?\\([^[[:alpha:]]\\|\\$\\|\\)" . 1)
        (face . (background-color . "blue")))))

```

Notice how the { and } regexps do not define a background colour (or indeed any other properties), so that any overlays they create will have no effect other than making sure all ‘{’ and ‘}’ characters are correctly paired.

We’ve made one mistake though: by putting the { regexp at the beginning of the list, it will take priority over any other regexp in the list that could match the same text. And since { will match whenever `\begin{}` or `\end{}` matches, environments will never be highlighted! The { regexp must come *after* the `\begin{}` and `\end{}` regexps, to ensure it is only used if neither of them match (it doesn’t matter whether it appears before or after the { regexp, since the latter will never match the same text):

```

(auto-overlay-load-definition
'latex
'(line ("% (priority . 4) (exclusive . t)
        (face . (background-color
                  . ,(face-attribute 'default :background))))))

```

```

(auto-overlay-load-definition
'latex
'(self ("\\$" (priority . 3) (face . (background-color . "green")))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("\\end{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("{"
   :edge start
   (priority . 2))
  ("}"
   :edge end
   (priority . 2))))

(auto-overlay-load-definition
'latex
'(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

(auto-overlay-load-definition
'latex
'(word ((("\\\\[[[:alpha:]]*?\\\[[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))

```

There is one last issue. A literal ‘{’ or ‘}’ character can be included in a L^AT_EX document by escaping it with ‘\’: ‘\{’ and ‘\}’. In this situation, the characters do not match anything and should not be treated as delimiters. We can modify the { and } regexps to exclude these cases:

```

(auto-overlay-load-definition
'latex
'(line ("% " (priority . 4) (exclusive . t)

```

```

                (face . (background-color
                        . ,(face-attribute 'default :background))))))

(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green")))))

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("\\end{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("\\([^\]\\\|^\)"
   :edge start
   (priority . 2))
  ("\\([^\]\\\|^\)"
   :edge end
   (priority . 2))))

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow")))
  ("\\end{equation}"
   :edge end
   (priority . 1)
   (face . (background-color . "yellow"))))

(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[[:alpha:]]*?\\([^\[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))

```

The new, complicated-looking regexps will only match ‘{’ and ‘}’ characters if they are *not* preceded by a ‘\’ character (see [section “Regular Expressions” in GNU Emacs Lisp Reference Manual](#)). Note that the character alternative `[^\]\\\|^` can match any character that isn’t a ‘\’ *or* the start of a line. This is required because matches to auto-overlay regexps are not allowed to span more than one line. If ‘{’ or ‘}’ appear at the beginning of

a line, there will be no character in front (the newline character doesn't count, since it isn't on the same line), so the `[^\]` will not match.

However, when it does match, the `}` regexp will now match an additional character before the `}`, causing the overlay to end one character early. (The `{` regexp will also match one additional character before the `{`, but since the beginning of the overlay starts from the *end* of the `start` delimiter, this poses less of a problem.) We need to group the part of the regexp that should define the delimiter, i.e. the `}`, by surrounding it with `\(` and `\)`, and put the regexp in the `car` of a cons cell whose `cdr` specifies the new subgroup (i.e. the 2nd subgroup, since the regexp already included a group for other reasons; we could alternatively replace the original group by a shy-group, since we don't actually need to capture match data for that group). Our final version looks like this:

```
(auto-overlay-load-definition
 'latex
 '(line ("% (priority . 4) (exclusive . t)
         (face . (background-color
                  . ,(face-attribute 'default :background))))))

(auto-overlay-load-definition
 'latex
 '(self ("\\$" (priority . 3) (face . (background-color . "green"))))

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("\\end{"
   :edge start
   (priority . 2)
   (face . (background-color . "pink")))
  ("\\([^\]|\\|^\|)"
   :edge start
   (priority . 2))
  ("\\([^\]|^\|)\|)" . 2)
  :edge end
  (priority . 2)))

(auto-overlay-load-definition
 'latex
 '(nested
  ("\\begin{equation}"
   :edge start
   (priority . 1)
   (face . (background-color . "yellow"))))
```

```

("\\end{equation}"
 :edge end
 (priority . 1)
 (face . (background-color . "yellow")))))

(auto-overlay-load-definition
 'latex
 '(word (("\\\\\\[[:alpha:]]*?\\([^[[:alpha:]]\\|\\$\\)" . 1)
        (face . (background-color . "blue")))))

```

With these regexp definitions, \LaTeX commands will automatically be highlighted in blue, equation environments in yellow, inline maths commands in green, and environment names in pink. \LaTeX markup within comments will be ignored. And ‘{’ and ‘}’ characters from other commands will be correctly taken into account. All this is done in “real-time”; it doesn’t wait until Emacs is idle to update the overlays. Not bad for a bundle of regexps!

Of course, this could all be done more easily using Emacs’ built-in syntax highlighting features, but the highlighting was only an example to show the location of the overlays. The main point is that the overlays are automatically created and kept up to date, and can be given any properties you like and used for whatever purpose is required by your Elisp package.

4 Extending the Auto-Overlays Package

The auto-overlay package can easily be extended by adding new overlay classes¹. The next sections document the functions and interfaces provided by the auto-overlays package for this purpose.

Often, a new class is a minor modification of one of the standard classes. For example, it may work exactly like one of the standard classes, but in addition call some function whenever an overlay is created or destroyed. In this case, it is far better to build the new class on top of the existing class, using functions from the class-specific Emacs files, rather than starting from scratch. See [Section 4.3.1 \[Standard Parse and Suicide Functions\]](#), page 21.

4.1 Auto-Overlays in Depth

In order to write new classes, a deeper understanding is required of how the auto-overlay package works. In fact, two kinds of overlays are automatically created, updated and destroyed when auto-overlays are active: the auto-overlays themselves, and “match” overlays, used to mark text that matches an auto-overlay regexp.

For overlay classes that only require one regexp to fully define an overlay (the `word` and `line` classes are the only standard classes like this²), the auto-overlays are always matched with the corresponding match overlay. For classes that require two regexp matches to define the start and end of an overlay (all other standard classes), each edge of an auto-overlay can be matched with a match overlay. The match overlays define where the edge of the auto-overlay is located. There will always be at least one matched edge, since an auto-overlay is only created when a regexp match is found, but it is possible for the second edge to not yet be matched (for many classes, the unmatched edge will be located at the beginning or end of the buffer).

If a match overlay delimits the start of an auto-overlay, the match overlay is stored in the auto-overlay’s `start` property. The match overlay is also stored in the `start` property for auto-overlays that only require a single match. If a match overlay delimits the end of an auto-overlay, the match overlay is stored in the auto-overlay’s `end` property. Conversely, a “link” to the auto-overlay is always stored in the match overlay’s `parent` property³.

Whenever a buffer is modified, the lines containing the modifications are scanned for new regexp matches. If one is found, a new match overlay is created covering the matching text, and then passed as an argument to the appropriate “parse” function⁴ for its class. This deals with creating or updating the auto-overlays as appropriate. If the text within a match overlay is modified, the match overlay checks whether the text it covers still matches the regexp. If it no longer matches, the match overlay is passed as an argument to the appropriate “suicide” function for its class, which deals with updating or deleting its parent auto-overlay (and possibly more besides).

¹ Or rather, it is easy to integrate new overlay classes into the package. Whether writing a new overlay class is easy or not depends on what you’re trying to do, and how good your coding skills are ;-)

² Although the `self` class only requires one regexp definition, the auto-overlays themselves require two matches to that same regexp to set the start and end of the overlay.

³ The “parent” terminology is admittedly very poor, and is a relic of a previous incarnation of the auto-overlays package, when it made more sense.

⁴ More bad terminology.

To summarise, the core of the auto-overlays package deals with searching for regexp matches, and creating or deleting the corresponding match overlays. It then hands over the task of creating, updating or deleting the auto-overlays themselves to class-specific functions, which implement the correct behaviour for that class.

4.2 Integrating New Overlay Classes

To add a new overlay class, all that is required is to write new “parse” and “suicide” functions, and inform the auto-overlays package of their existence. A “match” function can also optionally be defined. It is called whenever a match overlay in the class becomes matched with the edge of an auto-overlay (see [Section 4.3.2 \[Functions for Modifying Overlays\], page 21](#)). The parse, suicide and match functions are conventionally called `auto-o-parse-class-match`, `auto-o-class-suicide` and `auto-o-match-class`, where *class* is the name of the class, though the convention is not enforced in any way.

parse function

A parse function is passed a single argument containing a match overlay. It should return a list containing any new auto-overlays it creates, or `nil` if none were created.

```
o-list = (auto-o-parse-class-match o-match)
```

Note that the parse function itself is responsible for calling the `auto-o-update-exclusive` function if a new exclusive overlay is created. See [Section 4.3.2 \[Functions for Modifying Overlays\], page 21](#).

suicide function

A suicide function is passed a single argument containing a match overlay. Its return value is ignored.

```
(auto-o-class-suicide o-match)
```

The text covered by the match overlay should be considered to no longer match its regexp, although in certain cases matches are ignored for other reasons and this may not really be the case (for example if a new, higher-priority, exclusive overlay overlaps the match, see [Chapter 1 \[Overview\], page 1](#)).

match function

A match function is passed a single argument containing a match overlay that has just been matched with an edge of an auto-overlay (see [Section 4.3.2 \[Functions for Modifying Overlays\], page 21](#)). Its return value is ignored.

```
(auto-o-match-class o-match)
```

The auto-overlay it is matched with is stored in the match overlay’s `parent` property.

To integrate the new class into the auto-overlays package, the parse and suicide functions must be added to the property list of the symbol used to refer to the new class, denoted here by *class*:

```
(put 'class 'auto-overlay-parse-function
    'auto-o-parse-class-match)
(put 'class 'auto-overlay-suicide-function
    'auto-o-class-suicide)
```

If the optional match function is defined, it should similarly be added to the symbol’s property list:

```
(put 'class 'auto-overlay-match-function
     'auto-o-match-class)
```

4.3 Functions for Writing New Overlay Classes

Some functions are provided by the auto-overlays package for use in new parse and suicide functions. The functions that modify overlays carry out tasks that require interaction with the core of the auto-overlays package, and provide the only reliable way of carrying out those tasks. The other functions are used to query various things about auto-overlays and match overlays. Again, they are the only reliable interface for this, since the internal implementation may change between releases of the auto-overlays package.

4.3.1 Standard Parse and Suicide Functions

All the standard overlay classes define their own parse and suicide functions (none of them require a match function), which can be used to create new “derived” classes based on the standard ones. This is the easiest and most common way to create a new class. For example, the new class may behave exactly like one of the standard classes, but perform some additional processing whenever an overlay is created, destroyed, or matched. The parse and suicide functions for the new class should perform whatever additional processing is required, and call the standard class functions to deal with creating and destroying the overlay.

All the standard parse and suicide functions follow the same naming convention (see [Section 4.2 \[Integrating New Overlay Classes\]](#), page 20), where *class* is the name of the overlay class (one of *word*, *line*, *self*, *nested* or *flat*, see [Chapter 1 \[Overview\]](#), page 1):

```
(auto-o-parse-class-match o-match)
```

Parse a new match overlay *o-match* whose class is *class*. This will create or update auto-overlays, as appropriate for the class.

```
(auto-o-class-suicide o-match)
```

Delete or update auto-overlays as appropriate for overlay class *class*, due to the match overlay *o-match* no longer matching.

4.3.2 Functions for Modifying Overlays

These functions modify auto-overlays and match overlays as necessary to perform a particular update. They should *always* be used to carry out their corresponding tasks, rather than doing it separately, since these tasks require interaction with the core of the auto-overlays package.

```
(auto-o-update-exclusive set-id beg end old-priority new-priority)
```

Update the region between *beg* and *end* in the current buffer as necessary due to the priority of an exclusive overlay overlapping the region changing from *old-priority* to *new-priority*. If the exclusive overlay did not previously overlap the region, *old-priority* should be null. If it no longer overlaps the region, *new-priority* should be null. (If both are null, nothing will happen!) The return value is meaningless.

`(auto-o-match-overlay overlay start @optional end no-props no-parse protect-match)`

Match or unmatch the start and end of the auto-overlay *overlay*, update all appropriate properties (such as `parent`, `start` and `end` properties, and any properties specified in regexp definitions), and update other auto-overlays in the region covered by *overlay* if required because the `exclusive` or `priority` properties of *overlay* have changed.

If *start* or *end* are match overlays, match the corresponding edge of *overlay*. The edge is moved to the location defined by the match overlay, and the `parent` property of the match overlay and the `start` and `end` properties of *overlay* are updated accordingly. The *start* argument should be a match overlay corresponding either to the unique regexp if only one is needed for that overlay class, or to a start regexp if the overlay class uses separate start and end reg-exps. The *end* argument should then be a match overlay corresponding to an end regexp in such a class (see [Chapter 1 \[Overview\], page 1](#)). You're responsible for enforcing this; no check is made.

If *start* or *end* are numbers or markers, move the corresponding edge of *overlay* to that location and set it as unmatched. The `start` or `end` property of *overlay* and the `parent` property of any corresponding match overlay are set to `nil`). If *start* or *end* are non-`nil` but neither of the above, leave the corresponding edge of *overlay* where it is, but set it unmatched (as described above). If *start* or *end* are null, don't change the corresponding edge. However, for convenience, if *end* is null but *start* is a match overlay corresponding to a match for an end-regexp, match the end of *overlay* rather than the start.

The remaining arguments disable some of the tasks normally carried out by `auto-o-match-overlay`. If `no-props` is non-`nil`, overlay properties specified in regexp definitions are ignored and not updated. If `no-parse` is non-`nil`, auto-overlays in the region covered by *overlay* are not updated, even if the `exclusive` or `priority` properties of *overlay* have changed. If `protect-match` is non-`nil`, the `parent` properties of the *start* and *end* match overlays are left alone.

`(auto-o-delete-overlay overlay @optional no-parse protect-match)`

Delete auto-overlay *overlay* from the buffer, and update overlays and overlay properties as necessary. The optional arguments disable parts of the updating process, as for `auto-o-match-overlay`, above.

4.3.3 Functions for Querying Overlays

These functions query certain things about auto-overlays or match overlays, or retrieve certain values associated with them. A few are merely convenience functions, but most depend on the internal implementation details of the auto-overlays package, and provide the only reliable interface for whatever they return.

`(auto-o-class o-match)`

Return the class of match overlay *o-match*.

`(auto-o-regexp o-match)`

Return the regular expression matched by the text covered by match overlay *o-match*.

- `(auto-o-regexp-group o-match)`
Return the regexp group defined in the regexp definition corresponding to match overlay *o-match* (see [Section 2.1 \[Defining Regexps\]](#), page 3).
- `(auto-o-props o-match)`
Return the list of overlay properties defined in the regexp definition corresponding to match overlay *o-match* (see [Section 2.1 \[Defining Regexps\]](#), page 3).
- `(auto-o-edge o-match)`
Return edge (the symbol `start` or `end`) of match overlay *o-match*.
- `(auto-o-parse-function o-match)`
Return appropriate parse function for match overlay *o-match*.
- `(auto-o-suicide-function o-match)`
Return appropriate suicide function for match overlay *o-match*.
- `(auto-o-match-function o-match)`
Return match function for match overlay *o-match*, if any.
- `(auto-o-edge-matched-p overlay edge)`
Return non-nil if *edge* (the symbol `start` or `end`) of auto-overlay *overlay* is matched.
- `(auto-o-start-matched-p overlay)`
Return non-nil if auto-overlay *overlay* is start-matched.
- `(auto-o-end-matched-p overlay)`
Return non-nil if auto-overlay *overlay* is end-matched.

4.4 Auto-Overlay Hooks

The auto-overlays package defines two hooks, that are called when auto-overlays are enabled and disabled in a buffer. These are intended to be used by overlay classes to set up any extra buffer-local variables and settings they require, and clean them up afterwards. (There is no point leaving auto-overlay variables and settings hanging around in a buffer when auto-overlays are not in use.)

`auto-overlay-load-hook`

This hook is run when the first auto-overlay regexp set in a buffer is started, using the `auto-overlay-start` function. See [Section 2.2 \[Starting and Stopping Auto-Overlays\]](#), page 5.

`auto-overlay-unload-hook`

This hook is run when the last auto-overlay regexp set in a buffer is stopped, using the `auto-overlay-stop` function. See [Section 2.2 \[Starting and Stopping Auto-Overlays\]](#), page 5.

4.5 Auto-Overlay Modification Pseudo-Hooks

The auto-overlays package adds functions to buffer and overlay modification hooks in order to update the overlays as the buffer text is modified (see [section “Modification Hooks” in GNU Emacs Lisp Reference Manual](#)). The order in which all these modification hooks are

called is undefined in Emacs⁵. Therefore, the auto-overlays package provides a mechanism to schedule functions to run at particular points during the overlay update process.

There are two stages to the overlay update process: first, any match overlay suicide functions are called, then modified buffer lines are scanned for new regexp matches. Three pseudo-hooks are defined that are called before, after and in between these stages. Their values are lists containing elements of the form:

```
(function arg1 arg2 ...)
```

where *function* is the function to be called by the hook, and the *arg*'s are the arguments to be passed to that function. The list elements are evaluated in order. The pseudo-hooks are cleared each time after they have been called.

auto-o-pending-pre-suicide

Pseudo-hook called before any suicide functions.

auto-o-pending-post-suicide

Pseudo-hook called after any suicide functions but before scanning for regexp matches.

auto-o-pending-post-update

Pseudo-hook called after scanning for regexp matches.

These pseudo-hooks can be used to ensure that a function that would normally be added to a modification hook will be called at a particular point in the auto-overlay update process. To achieve this, a helper function must be added to the modification hook instead. The helper function should add the function itself to the appropriate pseudo-hook by adding a list element with the form described above. The `push` and `add-to-list` Elisp functions are the most useful ways to add elements to the list.

⁵ Or at least undocumented, and therefore unreliable.

5 To-Do

Things that still need to be implemented (in no particular order):

1. There needs to be an **eager-self** overlay class, similar to the existing **self** class but updated immediately, rather than waiting for buffer modifications. This will be significantly less efficient, but is necessary for applications that require overlays to be up to date all the time, not just when the buffer is being modified.
2. Currently, it's difficult to deal with **nested** class regexps for which the **end** regexps match some **start** regexps of interest but also others that are irrelevant. E.g. '{' and '}' in \LaTeX when you're only interested in ' $\text{\somecommand\{}$ ' **start** regexps. Or matching parens in LISP, when you're only interested in function bodies, say. The only solution is to include all **start** regexps, but not set any of their properties. This can end up creating a lot of overlays! A variant of the **nested** class that avoids this problem is needed.

Appendix A Function Index

auto-o-{class}-suicide	20, 21	auto-o-update-exclusive.....	21
auto-o-class	22	auto-overlay-highest-priority-at-point	7
auto-o-delete-overlay.....	22	auto-overlay-load-definition.....	4
auto-o-edge	23	auto-overlay-load-overlays	6
auto-o-edge-matched-p.....	23	auto-overlay-load-regexp.....	4
auto-o-end-matched-p	23	auto-overlay-local-binding	7
auto-o-match-{class}.....	20	auto-overlay-save-overlays	6
auto-o-match-function.....	23	auto-overlay-share-regexp-set.....	4
auto-o-match-overlays	22	auto-overlay-start	5
auto-o-parse-{class}-match.....	20, 21	auto-overlay-stop.....	5
auto-o-parse-function.....	23	auto-overlay-unload-definition.....	4
auto-o-props	23	auto-overlay-unload-regexp.....	4
auto-o-regexp	22	auto-overlay-unload-set.....	4
auto-o-regexp-group.....	23	auto-overlays-at-point	6
auto-o-start-matched-p.....	23	auto-overlays-in.....	7
auto-o-suicide-function.....	23		

Appendix B Variable Index

auto-o-pending-post-suicide	24	auto-overlay-load-hook	23
auto-o-pending-post-update	24	auto-overlay-unload-hook	23
auto-o-pending-pre-suicide	24		

Appendix C Concept Index

A

adding new overlay classes	19
auto-overlay definitions	4
auto-overlay definitions, unloading	4
auto-overlays in depth	19
auto-overlays, defining	4
auto-overlays, loading	4

B

buffers, sharing regexp sets between	4
--	---

C

class, flat	1
class, line	1
class, line example	10
class, nested	1
class, nested example	9, 11
class, self	1
class, self example	9
class, standard parse functions	21
class, standard suicide functions	21
class, word	1
class, word example	8
classes of overlay	1
classes, adding new	19
classes, integrating new	20

D

defining auto-overlays	4
defining regexps	3, 4
deleting overlays	22
delimiter	2

E

example	8
example, line class	10
example, nested class	9, 11
example, self class	9
example, word class	8
exclusive property	2, 21
extending the auto-overlays package	19
extending, deleting overlays	22
extending, functions	21
extending, functions for modifying overlays	21
extending, functions for querying overlays	22
extending, integrating new overlay classes	20
extending, matching overlays	22
extending, standard parse functions	21
extending, standard suicide functions	21

extending, updating exclusive	21
-------------------------------------	----

F

FDL, GNU Free Documentation License	30
finding overlays	6
flat overlay class	1
functions	3
functions, defining regexps	3, 4
functions, loading and saving overlays	5
functions, loading and unloading regexps	3, 4
functions, match function	20
functions, modifying overlays	21
functions, parse function	20
functions, querying overlays	22
functions, scheduling	23
functions, searching for overlays	6
functions, starting and stopping overlays	5
functions, suicide function	20
functions, writing new overlay classes	21

G

grouping in regexps	2
---------------------------	---

H

highest priority overlay	7
hooks	23
hooks, loading and unloading	23
hooks, modification	23

I

integrating new classes, match function	20
integrating new classes, parse function	20
integrating new classes, suicide function	20
integrating new overlay classes	20

L

LaTeX	8
line overlay class	1
line overlay class example	10
loading auto-overlay definitions	4
loading overlays	5
loading regexps	4
loading the package	3
local-binding	7

M

match function	20
----------------------	----

matching overlays 22
 modification pseudo-hooks 23

N

nested overlay class 1
 nested overlay class example 9, 11

O

overlay class, flat 1
 overlay class, line 1
 overlay class, line example 10
 overlay class, nested 1
 overlay class, nested example 9, 11
 overlay class, self 1
 overlay class, self example 9
 overlay class, word 1
 overlay class, word example 8
 overlay classes 1
 overlay classes, functions for writing new 21
 overlay classes, integrating new 20
 overlay classes, match function 20
 overlay classes, parse function 20
 overlay classes, standard parse functions 21
 overlay classes, standard suicide functions 21
 overlay classes, suicide function 20
 overlay properties 2, 6
 overlay property, exclusive 2, 21
 overlay property, priority 2
 overlay-local binding 7
 overlays, deleting 22
 overlays, finding 6
 overlays, functions for modifying 21
 overlays, functions for querying 22
 overlays, local-binding 7
 overlays, matching 22
 overlays, priority 7
 overlays, saving and loading 5
 overlays, starting and stopping 5
 Overview 1

P

package, extending 19
 package, hooks 23
 package, in depth 19
 package, loading 3

parse function 20
 priority property 2

R

regexp definitions, unloading 4
 regexp groups 2
 regexp sets 1
 regexp sets, sharing between buffers 4
 regexp sets, starting and stopping 5
 regexp sets, unloading 4
 regexps, defining 3, 4
 regexps, loading 4
 regexps, loading and unloading 3, 4
 regexps, unloading 4
 require 3

S

saving overlays 5
 scheduling functions after modification 23
 searching for overlays 6
 self overlay class 1
 self overlay class example 9
 sets of regexps 1
 sharing regexp sets 4
 standard parse and suicide functions 21
 starting and stopping auto-overlays 5
 suicide function 20

T

to-do 25

U

unloading regexp definitions 4
 unloading regexp sets 4
 unloading regexps 4
 updating exclusive regions 21
 using auto-overlays 3

W

word overlay class 1
 word overlay class example 8
 worked example 8

Appendix D Copying this Manual

D.1 GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

- be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
 - C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
 - D. Preserve all the copyright notices of the Document.
 - E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
 - F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
 - G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
 - H. Include an unaltered copy of this License.
 - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
 - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
 - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
 - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
 - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
 - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

D.1.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the GNU Free Documentation License, Version 1.2  
or any later version published by the Free Software Foundation;  
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover  
Texts. A copy of the license is included in the section entitled ‘‘GNU  
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with  
the Front-Cover Texts being list, and with the Back-Cover Texts  
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.