text.editing User's Manual

Table of Contents

1 Introduction

The text.editing library provides protocols and implementations of those protocols for text editing operations which are required by, for example, text editors or commandline processors (like REPLs). Some of the concepts and naming used in this library are inspired by Emacs but the functionality should be generic enough to be build other kinds of Editors as well. This library relies on the Cluffer library (`https://github.com/Robert-Strandh/cluffer`) for the fundamental concepts of buffers, lines and cursors.

The functionality provided by this library includes:

- Section 3.6.1 [Motion Operations], page 14, (by various "[term-unit], page 3")
- Section 3.6.2 [Insertion Operations], page 14, and Section 3.6.3 [Deletion Operations], page 15, (by various "units")
- Transformations like [Generic-Function text.editing|change-case], page 17, or [Generic-Function text.editing|transpose], page 17,
- Chapter 5 [Expressions], page 30, and other Section 3.6.11 [Operations on Delimiter Pairs], page 18, (like paredit for Emacs (`https://paredit.org/`))
- Undo (work in progress)
- Section 3.6.6 [Copy and Yank Operations], page 17,
- Section 3.4.2 [Multiple Sites Protocol], page 12,
- Chapter 4 [Search], page 23,
- Abbreviations (work in progress)

The text.editing library does not provide:

- Data structures for editor buffers (the Cluffer library (`https://github.com/robert-strandh/cluffer`) does that)
- Input handling or a command processor (This aspect obviously heavily depends on the application and there are many ways to do it. McCLIM (`https://codeberg.org/McCLIM/McCLIM`) is one library which provides both input handling and command processing.)
- Advance parsing (the Incrementalist library (`https://github.com/robert-strandh/incrementalist`) does that)
- Syntax highlighting
- Presentation/rendering/display functions for the contents of text buffers (Like command processing, this aspect heavily depends on the application it. Again, McCLIM (`https://codeberg.org/McCLIM/McCLIM`) is one library that could be used.)

2 Concepts

This section defines a few concepts that are important for the text.editing library and describes how they related to each other. The following figure illustrates some of the concepts and their relations:

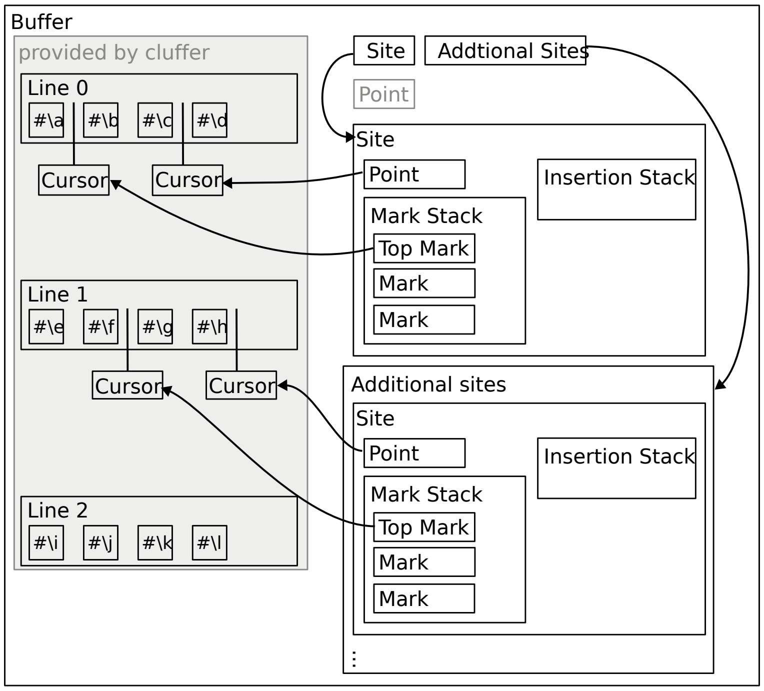

Figure 2.1: Examples of important concepts and their relations for a single buffer.

Buffer

A *buffer* is conceptually a sequence of lines, each of which contains a sequence of items and zero or more attached cursors.

The text.editing library does not provide an implementation of the buffer concept or associated protocols. Instead, it uses the protocols and classes provided by the Cluffer library. It does, however, provide mixin classes like ⟨undefined⟩ [Class text.editing|multiple-site-mixin], page ⟨undefined⟩, that are intended to be used in user-defined buffer classes.

Cursor

A *cursor* is an object that is attached to a buffer line before the first item, after the last item or between two adjacent items.

Some cursors like the distinguished [term-point], page 3, and [term-mark], page 3, cursors are visible to and controlled by the user while others are used programmatically. Editing [term-operation], page 3, generally accept one or more cursor arguments to indicate to which location or region of the buffer the operation should be applied.

The text.editing library does not provide an implementation of the cursor concept or associated protocols. Instead, it uses the protocols and classes provided by the Cluffer library.

Point

The *point* is a distinguished cursor within each [term-site], page 3, which specifies the buffer location at which the next editing [term-operation], page 3, issued by a user will be performed.

Mark

The *mark* is a distinguished cursor within each [term-site], page 3, that has multiple purposes:

- Mark cursors can be used to save buffer locations and return to them later.
- Together with the [term-point], page 3, cursor, the mark cursor can specify a region within the buffer on which [term-operation], page 3, can be performed.

The mark cursor of a site can be either *active* or *inactive*.

Operation

An *operation* changes the buffer content and/or state of [term-site], page 3, in a particular way, usually depending on the values of one or more parameters.

For example, the [Generic-Function text.editing|move], page 14, operation changes the buffer positions of the point cursors of all sites according to the specified *unit* and *direction* arguments.

Region

If a [term-site], page 3, has an active [term-mark], page 3, cursor, the sequence of items between the [term-point], page 3, cursor and the mark cursor forms the *region*. [term-operation], page 3, can be performed on the region using the [unit-region], page 6, [term-unit], page 3. The specified sequence of items is the same regardless of which of the two cursors is closer to the beginning of the buffer.

Unit

Units are a way to designate particular sub-sequences of the sequence of all items in a buffer, often relative to the [term-point], page 3, cursor. For example, the [unit-word], page 6, unit refers to a sequence of non-whitespace, non-punctuation characters that follow (or precede depending on the specified direction) the point cursor.

Site

A *site* ties together pieces of data that are required for performing consecutive editing [term-operation], page 3, around a specific "location", or site, in a buffer. The most important piece of data is the [term-point], page 3, cursor which makes precise the notion of a buffer "location". Other pieces of data include the Section 3.3.2 [Preferred Column Protocol], page 10, of the point cursor, an optional [term-mark], page 3, cursor, a [term-mark-stack], page 4, and an [term-insertion-stack], page 4.

The main reason for storing this data in a dedicated site object instead of directly in a buffer is the possibility of allowing simultaneous editing at multiple sites in a buffer. From the perspective of an editor user, each site would typically appear as a cursor (with its own point, mark, insertion stack, etc.) which would generally act as if it were the only cursor in the buffer (disregarding effects that arise from sites being too close together or overlapping).

Each buffer has exactly one *primary* site and zero or more *secondary* sites. Secondary sites are added and removed by ⟨undefined⟩ [Generic-Function text.editing|add-site], page ⟨undefined⟩. A secondary site and the primary site can [Generic-Function text.editing|rotate-sites], page 13, since the invariant that the associated buffer has to have exactly one primary site is preserved by that operation.

Mark Stack

A *mark stack* is a stack which contains former mark cursors of a [term-site], page 3. Typical operations on the mark stack include pushing a mark cursor that corresponds to the location of the point cursor onto the mark stack and later popping the entry "into" the point cursor. This combination of operations allows remembering buffer locations and returning to them later.

Insertion Stack

The *insertion stack* is a stack the elements of which are recently copied or killed sequences of buffer items which are available for insertion into a buffer. "Kill" and "yank" operations push to and pop from this stack.

This concept is similar to the "Kill Ring" in Emacs with the following differences:

- As the name suggests, the Emacs kill ring can grow to a maximum number of items after which it will start discarding the least recent elements. In practice however, Emacs is often configured to keep a practically unlimited number of kill ring elements. The insertion stack is unlimited by default.
- The Emacs kill ring is global by default and has to be restricted to a local context for extended functionality like editing with multiple cursors. In contrast, each insertion stack is local to a specific site by default.

2.1 Units

[term-site], page 3, [term-unit], page 3, and [term-operation], page 3, are the basic concepts from which most desired behaviors can be constructed.

-

  The site controls where the operation is applied via the point and possibly mark cursor. The site also provides additional context such as the history of previous operations, the mark stack and the insertion stack.

-

  The unit, together with the point cursor and possibly the mark cursor, controls to which buffer items the operation should be applied. Many operations accept a direction argument which also influences the processed items.

-

  The operation selects the basic behavior.

Here are a few examples (see Chapter 6 [Equivalent Emacs Commands], page 38, for more examples):

Operation	**Unit**	**Direction**	**Arguments**	**Equivalent Emacs command**
move	item	forward		`forward-char` (*C-f*)
move	item	backward		`backward-char` (*C-b*)
move	word	forward		`forward-word` (*M-f*)
move	word	backward		`backward-word` (*M-b*)
move	line	forward		`next-line` (*C-n*)
move	line	backward		`previous-line` (*C-p*)
delete	item	forward		`delete-char` (*C-d*)
delete	item	backward		`delete-backward-char` (*<backspace>*)
delete	word	forward		`kill-word` (*M-d*)
delete	word	backward		`backward-kill-word` (*M-<backspace>*)
delete	line	forward		`kill-line` (*C-k*)
delete	line	backward		`kill-line` with `0` prefix (*C-0 C-k*)
change-case	word	forward	`:capital`	`capitalize-word` (*M-c*)

The key observation is that operations, units and directions are mostly orthogonal. In other words, new operations and units that are defined independently should still work together just fine in most cases. This relative independence is achieved via the [Generic-Function text.editing|apply-from-cursor], page 8, which applies a given item-wise operation to a sub-sequence of buffer items specified as a unit and a direction.

2.1.1 Built-in Units

The following units are provided by the text.editing library by default. Users of the library can define additional units.

Name	**Super-units**	**Description**
`edit:buffer`	buffer-unit	The whole buffer.
`edit:buffer-boundary`	buffer-unit	An empty sequence at the beginning or end of the buffer.
`text.editing.expression:expression`	unit	A sequence of buffer items that correspond to a node in a syntax tree associated with the buffer.
`edit:item`	unit	A single item, usually a character, in a buffer.

`edit:line`	line-unit	A line within a buffer.
`edit:line-boundary`	line-unit	An empty sequence at the beginning or end of the line.
`edit::page`	prose-unit	*not documented*
`edit:paragraph`	prose-unit	A sequence of items delimited by two newlines. The beginning and end of the buffer also delimit paragraphs.
`edit:region`	region-unit	The sequence of characters between point and mark. It does not matter whether point or mark is closer to the beginning of the buffer.
`text.editing.expression:region-or-expression`	region-or	The sequence of characters between point and mark or the innermost expression containing point. Like the 'region' unit if the mark is set and active, otherwise like the 'expression' unit.
`edit:region-or-item`	region-or	The sequence of characters between point and mark or a single item. Like the 'region' unit if the mark is set and active, otherwise like the 'item' unit.
`edit:semi-buffer`	buffer-unit	The sequence from the cursor to one end of the buffer.
`edit:semi-line`	line-unit	The sequence from the cursor to the beginning or end of the line.
`edit:sentence`	prose-unit	A sequence of word and whitespace items that is delimited by punctuation. The beginning and end of the buffer also delimit sentences.
`text.editing.expression:toplevel-expression`	expression	The sequence of buffer items that correspond to the toplevel expression node in a syntax tree associated with the buffer.
`edit:word`	prose-unit	A sequence of items that does not contain whitespace or punctuation characters. The beginning and end of the buffer also delimit words.

The hierarchy of built-in unit classes looks like this:

```
unit
+-region-unit
| +-[unit-region], page 6
| ‘-region-or
|   +-[unit-region-or-item], page 6
|   ‘-[unit-region-or-expression], page 6
+-[unit-item], page 5
+-line-unit
| +-[unit-line], page 6
| +-[unit-semi-line], page 6
| ‘-[unit-line-boundary], page 6
+-buffer-unit
| +-[unit-buffer], page 5
| +-[unit-semi-buffer], page 6
| ‘-[unit-buffer-boundary], page 5
+-prose-unit
| +-[unit-word], page 6
| +-[unit-sentence], page 6
| +-[unit-paragraph], page 6
| ‘-[unit-page], page 6
‘-[unit-expression], page 5
  ‘-[unit-toplevel-expression], page 6
```

3 External Protocols

This chapter describes the external protocols provided by the text.editing library.

3.1 Detach Protocol

This protocol is used to sever connections between [term-site], page 3, (and associated objects) and buffers when those objects should no longer be associated with the respective buffer.

`detach` `[text.editing]` [Generic Function]

(object) Detach *object* from any buffer or line it is currently attached to.

3.2 Unit Protocol

`all-units` `[text.editing]` [Function]

Return a sequence of all defined units.

`apply-from-cursor` `[text.editing]` [Generic Function]

continuation cursor unit direction Repeatedly call *continuation* until the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction* has been processed.

continuation is a function the lambda list of which has to be compatible with `(cursor item)`. The function will be called for each item in the indicated sub-sequence with a cursor that is positioned before or after the item as the first argument and the item as the second argument. *cursor* is positioned before *item* if *direction* is `:forward` and after *item* if *direction* is `:backward`.

unit is the unit in or at or around *cursor* that *continuation* should be applied to. Examples of units are [unit-item], page 5, [unit-line], page 6, [unit-word], page 6, and [unit-paragraph], page 6.

direction is the direction in which the processing should be performed from or around *cursor*. Possible values are `:forward` and `:backward`.

`item-transformer` `[text.editing]` [Generic Function]

transform direction Return a function that transforms items via *transform* when passed to [Generic-Function text.editing|apply-from-cursor], page 8.

transform is a function that accepts an item as its sole argument and returns an item (either the item passed to it or a new item).

direction specifies the direction for which the returned function will be valid. In other words, when the returned function is passed to [Generic-Function text.editing|apply-from-cursor], page 8, that call has to use the same value for the *direction* argument as the call to this function.

3.3 Site Protocols

This section describes protocols related to [term-site], page 3, and mixin classes that provide default implementations of those protocols.

3.3.1 Insertion Stack Protocol

This protocol allows querying and manipulating the entries of an insertion stack. This protocol is not concerned with buffers, sites or cursors. See Section 3.6.6 [Copy and Yank Operations], page 17, for a higher-level protocol on top of this one.

`insertion-stack-empty-error` `[text.editing]` [Class]
: This error is signaled when an attempt is made to retrieve an entry from an empty insertion stack.

`forward` `[text.editing]` [Generic Function]
: insertion-entry Return the sequence of items that have been added to *insertion-entry* by forward deletion operations such as `cluffer:delete-item` and `cluffer:join-line`.

`(setf forward)` `[text.editing]` [Generic Function]
: new-value insertion-entry Set the sequence of items for forward deletion operations of *insertion-entry* to *new-value*.

`backward` `[text.editing]` [Generic Function]
: insertion-entry Return the sequence of items that have been added to *insertion-entry* by backward deletion operations such as `cluffer:erase-item`.

`(setf backward)` `[text.editing]` [Generic Function]
: new-value insertion-entry Set the sequence of items for backward deletion operations of *insertion-entry* to *new-value*.

`insertion` `[text.editing]` [Generic Function]
: insertion-entry Return a sequence of items that should be inserted into a buffer to conceptually insert *insertion-entry* into that buffer.

  The returned sequence is the concatenation of the items of the “forward” and “backward” sequences of *insertion-entry* in the appropriate order.

`entry-count` `[text.editing]` [Generic Function]
: insertion-stack Return number of insertion entries in *insertion-stack*.

`top-entry` `[text.editing]` [Generic Function]
: insertion-stack Return the top entry in *insertion-stack* or `nil`.

  The `forward`, `backward` and `insertion` functions can be applied to the returned object.

`find-entry` `[text.editing]` [Generic Function]
: index insertion-stack Return the entry at *index* in *insertion-stack*.

  The `forward`, `backward` and `insertion` functions can be applied to the returned object.

`push-entry` `[text.editing]` [Generic Function]
: insertion-stack Add a new entry to *insertion-stack* and return the new entry.

`pop-entry` `[text.editing]` [Generic Function]
: insertion-stack Remove the top entry from *insertion-stack* and return the removed entry.

  If *insertion-stack* is empty, signal an error of type [Class text.editing|insertion-stack-empty-error], page 9.

3.3.2 Preferred Column Protocol

The purpose of this protocol is tracking in which column the [term-point], page 3, cursor of a site should be placed when the cursor repeatedly moves vertically (between lines) without other movement or operations.

`preferred-column` `[text.editing]` [Generic Function]
: site Return the column number in which the point of *site* should reside by default or `nil`.

  The point cursor should be placed in that column or the rightmost existing column of the current line when the point cursor moves between lines without moving within any line.

`(setf preferred-column` `[text.editing])]` [Generic Function]
: new-value site Set the column number in which the point of *site* should reside by default to *new-value*.

A default implementation of this protocol is provided by the following mixin class:

`preferred-column-tracking-mixin` `[text.editing]` [Class]
: This class is intended to be mixed into site classes that track preferred column of the point cursor.

3.3.3 Operation History Protocol

The purpose of this protocol is recording the sequence of operations that have been applied to a given [term-site], page 3. This allows the command processor or certain operations, for example, to take into account the previous operation. Examples:

- Some operations behave differently when repeated such as setting the [term-mark], page 3, twice to first set the mark then deactivate it.
- The command processor may track runs of deletion operations to collect the deleted items into a single [term-insertion-stack], page 4, entry.
- The command processor may track runs of insertion, deletion or modification operations to create undo groups from multiple primitive operations.

`most-recent-operation` `[text.editing]` [Generic Function]
: site Return the most recent operation of *site* or `nil`.

`push-operation` `[text.editing]` [Generic Function]
: operation site Push *operation* into the operation history of *site*.

  operation should be a list of the form `(`*`operation-name . arguments`*`)` where *operation-name* is a symbol that names an operation function.

A default implementation of this protocol is provided by the following mixin class:

`operation-history-mixin` `[text.editing]` [Class]

This class is intended to be mixed into site classes that track the history of performed operations.

3.3.4 Site Protocol

The site protocol extends the Section 3.1 [Detach Protocol], page 8, that is, sites can be detached.

`point` `[text.editing]` [Generic Function]

site Return the point cursor of *site*.

The returned object is a Cluffer cursor

`mark` `[text.editing]` [Generic Function]

site Return the mark cursor of *site*.

The returned object is a Cluffer cursor.

`mark-active-p` `[text.editing]` [Generic Function]

site Indicate whether the mark cursor of *site* is active.

`(setf mark-active-p)` `[text.editing]` [Generic Function]

new-value site Change whether the mark cursor of *site* is active.

new-value is a generalized Boolean.

`mark-stack` `[text.editing]` [Generic Function]

site Return the mark stack of *site*.

`insertion-stack` `[text.editing]` [Generic Function]

site Return the insertion stack of *site*.

The returned object implements the Section 3.3.1 [Insertion Stack Protocol], page 9.

3.4 Buffer Protocols

This section describes protocols for buffers and mixin classes which provide default implementations of the protocols. The described protocols are extensions of the Cluffer protocols for buffers in the sense that objects which are used with the protocols described here must also implement the Cluffer protocols. Similarly, the mixin classes are intended to be mixed into classes that are also subclasses of the buffer classes provided by Cluffer.

3.4.1 Primary Site Protocol

`site` `[text.editing]` [Generic Function]

buffer Return the [term-primary-site], page 4, of *buffer*.

The generic functions [Generic-Function text.editing|point], page 11, and [Generic-Function text.editing|mark], page 11, defined in the Section 3.3.4 [Site Protocol], page 11, work on buffers as well. The return value is the primary point cursor and the primary mark cursor respectively.

3.4.2 Multiple Sites Protocol

The following condition types are used in the multiple sites protocol:

`singular-site-error` `[text.editing]` [Class]

This error is signaled if an operation that requires multiple [term-site], page 3, is performed on a buffer that contains only a single site.

An implementation of this protocol is provided by the class

`multiple-site-mixin` `[text.editing]` [Class]

This class is intended to be mixed into buffer classes that can contain zero or more secondary [term-site], page 3, in addition to the primary site.

The protocol consists of the following generic functions:

`site-count` `[text.editing]` [Generic Function]

buffer Return the total number of sites that are attached to *buffer*.

The returned count includes the primary site.

`map-sites` `[text.editing]` [Generic Function]

function buffer Call *function* with each site that is attached to *buffer*.

`sites` `[text.editing]` [Generic Function]

buffer Return the sequence of all sites which are attached to *buffer*.

`add-site` `[text.editing]` [Generic Function]

site buffer Add *site* to the sites of *buffer*.

Return *site*.

If the point cursor or the mark cursor of *site* is associated with a buffer other than *bar*, an error is signaled.

If *site* is already one of the sites that are attached to *buffer*, signal an error.

`remove-site` `[text.editing]` [Generic Function]

site buffer ⟨undefined⟩ [Generic-Function text.editing|detach], page ⟨undefined⟩, *site* and remove it from the sites of *buffer*.

Return *site*.

If *site* is not one of the sites that are attached to *buffer*, signal an error.

If *site* is the primary site of *buffer*, signal an error.

`push-site-at` `[text.editing]` [Generic Function]

buffer line position Create a new site at *line* and *position* and attach it to *buffer*.

line and *position* control the location of the point cursor of the new site.

Return the new site.

`push-site-relative` `[text.editing]` [Generic Function]

buffer unit direction Create a new site relative to the primary site and attach it to *buffer*.

unit and *direction* control the location of the point cursor of the new site. The new point cursor starts at the location of the primary point cursor, then moves according to *unit* and *direction*.

Return the new site.

The attempt to move the new point cursor the specified location may result in an error. In that case, the new site is not attached and the error is signaled.

`pop-site` `[text.editing]` [Generic Function]

buffer Remove the most recently added [term-site], page 3, from *buffer*.

Return the removed site.

If no sites beside the primary site are attached to *buffer*, signal an error of type [Class text.editing|singular-site-error], page 12.

`rotate-sites` `[text.editing]` [Generic Function]

buffer direction Swap roles between the primary [term-site], page 3, and secondary sites in *buffer*.

direction controls the direction of the rotation.

If *direction* is `:forward`, sites are rotated as follows:

```
primary          ← first secondary
first secondary  ← second secondary
second secondary ← third secondary
...
last secondary   ← primary
```

direction `:backward` is not supported at the moment

If no sites beside the primary site are attached to *buffer*, signal an error of type [Class text.editing|singular-site-error], page 12.

`other-sites` `[text.editing]` [Generic Function]

buffer Return the sequence of all sites which are attached to *buffer* except the primary site.

`remove-other-sites` `[text.editing]` [Generic Function]

buffer Remove all sites from *buffer* except the primary site.

3.5 Operation Protocol

`perform` `[text.editing]` [Generic Function]

target operation `&rest` operation-arguments Perform *operation* with *operation-arguments* in or on *target*.

target is the object in or at or on which the operation should be performed such as a buffer or a cursor.

operation designates a function which performs the desired operation when called with a target object (not necessarily *target*) as the first argument. The target object in the call to *operation* may be different from *target* when methods on this generic function translate an operation on one target object to one or more operations on other target objects. For example, an operation on a buffer is commonly translated to one operation on each [term-site], page 3, of the buffer and further to one operation on the point cursor of each site of the buffer.

operation-arguments is a list of additional arguments that should be passed to the function designated by *operation*.

This function generally returns the values returned by the *operation* call. Similarly, calls to this function may signal any condition that may be signaled by the *operation* call. However, if *target* is a buffer and multiple sites exist, a different convention may be used in order to return one result for each site or bundle conditions for multiple sites in a single condition (see Section 3.4.2 [Multiple Sites Protocol], page 12).

3.6 Movement and Editing Protocol

note: The operations described in this section can be invoked by calling the respective generic function. However, a more flexible way which, for example, handles multiple sites correctly is the Section 3.5 [Operation Protocol], page 13. The following code invokes an operation *operation* via that protocol

```
(text.editing:perform buffer 'operation unit direction other-
arguments)
```

3.6.1 Motion Operations

`move` [text.editing] [Generic Function]

cursor unit direction Move *cursor* to the beginning or end of the sub-sequence of buffer items indicated by *unit*.

cursor is an attached Cluffer cursor.

unit is a unit of movement such as [unit-item], page 5, or [unit-word], page 6.

If *direction* is `:forward`, *cursor* moves to the end of the sub-sequence. If *direction* is `:backward`, *cursor* moves to the beginning of the sub-sequence.

`back-to-indentation` [text.editing] [Generic Function]

cursor Move *cursor* to the first column of the current line that contains a non-whitespace item.

3.6.2 Insertion Operations

`insert-item` [text.editing] [Generic Function]

cursor item Insert *item* at *cursor*.

`insert-newline` [text.editing] [Generic Function]

cursor Split the current line at the position of *cursor*.

`insert-items` [text.editing] [Generic Function]

cursor items &key start end Insert the items in *items* at *cursor*.

start and *end*, when supplied, select a sub-sequence of *items*.

3.6.3 Deletion Operations

`delete` `[text.editing]` [Generic Function]

cursor unit direction Delete the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction*.

cursor is an attached Cluffer cursor.

unit is a unit of movement such as [unit-item], page 5, or [unit-word], page 6.

direction is either `:forward` or `:backward`.

`delete-indentation` `[text.editing]` [Generic Function]

cursor Join previous and current line, delete whitespace before and after *cursor*.

Keep a single space character unless the delete placed *cursor* on an empty line.

`delete-trailing-whitespace` `[text.editing]` [Generic Function]

cursor Delete trailing whitespace from buffer lines.

cursor determines the first line to be processed. All subsequent lines to the end of the buffer are processed after that.

`fixup-whitespace` `[text.editing]` [Generic Function]

cursor Delete consecutive whitespace before and after *cursor* in the current line.

Keep a single space character unless the deletion placed *cursor* at the beginning of the line.

3.6.4 Items Functions

The following convenience function allow easy retrieval and mutation of sub-sequences of buffer items:

`map-items` `[text.editing]` [Generic Function]

function cursor unit direction Call *function* with each item in the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction*.

`items` `[text.editing]` [Generic Function]

cursor unit direction Return a `cl:sequence` containing the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction*.

`(setf items)` `[text.editing]` [Generic Function]

new-value cursor unit direction Replace the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction* by the items in the `cl:sequence` *new-value*.

3.6.5 Marking Operations

The mark protocol contains operations for managing different aspects of the [term-mark], page 3, cursor of a [term-site], page 3:

- The mark cursor can be set or not.
- A set mark cursor can be active or inactive. When the mark is active, the point cursor and mark cursor define the [term-region], page 3, of the site.
- A mark stack stores previous locations of the mark cursor.

`mark-or-error` `[text.editing]` [Generic Function]
: object Return the mark cursor of *object* or signal an error.

  If the mark cursor of *object* is not set, signal an error of type `mark-not-set-error`.

`activate-mark` `[text.editing]` [Generic Function]
: site Set the state of the mark cursor of *site* to active.

  Signal an error of type `mark-not-set-error` if the mark of *site* is not set.

`deactivate-mark` `[text.editing]` [Generic Function]
: site Set the state of the mark cursor of *site* to inactive.

`set-mark` `[text.editing]` [Generic Function]
: site Set the mark cursor of *site* to the position of the point cursor.

  Push the current mark cursor, if any, onto the mark stack, set a new mark cursor and move it to the position of the point cursor. Activate the mark.

  Return the new mark cursor.

`set-mark-or-toggle-active` `[text.editing]` [Generic Function]
: site Set the mark cursor of *site* or toggle its active state.

  If the previous command was not `set-mark-or-toggle-active`, then push the current mark cursor of *site* onto the mark stack, set a new mark cursor and move it to the position of the point cursor.

  If the previous command was `set-mark-or-toggle-active`, then toggle the active state of the mark cursor of *site*.

  Return two values: a Boolean which indicates whether a new mark cursor was set and another Boolean which indicates whether the mark is active.

`pop-mark` `[text.editing]` [Generic Function]
: site Pop a mark off the mark stack of *site* and move the point cursor to it.

  Destroy the current mark of *site*, if any.

  Return the popped mark cursor.

  Signal an error of type `mark-stack-empty` if the mark stack of *site* is empty.

`exchange-point-and-mark` `[text.editing]` [Generic Function]
: site Exchange the locations of point and mark of *site*.

  Signal an error of type `mark-not-set-error` if the mark of *site* is not set.

`mark-object` `[text.editing]` [Generic Function]
: site unit direction Set region of *site* according to *unit* and *direction*.

  Leave the point cursor of *site* at its current location. Ensure the mark is set and active (see below) and move the mark cursor according to *unit* and *direction*.

  If the mark of *site* is not set, set a new mark cursor at the location of the point cursor and activate it. Then apply the motion according to *unit* and *direction*.

  If the mark of *site* is set but not active, activate the mark cursor and move it to the location of the point cursor. Then apply the motion according to *unit* and *direction*.

  If the mark of *site* is set and active, just apply the motion according to *unit* and *direction*. This last case allows extending the region by marking subsequent objects.

3.6.6 Copy and Yank Operations

The copy and yank protocol offers higher-level functions that implement typical copy and yank operations which abstract from the details of the lower-level Section 3.3.1 [Insertion Stack Protocol], page 9.

`yank` `[text.editing]` [Generic Function]
site direction `&key` pop Insert top insertion stack entry of *site* at the point of *site*.

Insert the items from the top entry of the insertion stack of *site* before or after the point cursor of *site*.

direction controls whether the items are inserted before or after the point cursor, or equivalently, whether the point cursor moves to the beginning or end of the inserted items after the insertion.

pop controls whether the top entry of the insertion stack should be popped off.

`copy` `[text.editing]` [Generic Function]
site unit direction Copy items according at *site* according to *unit* and *direction*.

The items indicated by the point cursor of *site*, *unit* and *direction* are copied into either the current top entry of the insertion stack of *site* or a new entry that is first pushed onto the insertion stack.

Whether a new entry should be created is decided according to an internal protocol that may be exported at some later time.

3.6.7 Case Changing Operations

`change-case` `[text.editing]` [Generic Function]
cursor unit direction case Change the case of the sub-sequence of buffer items indicated by *cursor*, *unit* and *direction* according to *case*.

cursor is an attached Cluffer cursor.

unit is a unit of movement such as [unit-item], page 5, [unit-word], page 6, or `expression`.

direction is either `:forward` or `:backward`.

case has to be one of `:down`, `:up` or `:capital`.

The case of an item is changed by calling `cl:char-downcase` if *case* is `:down`, `cl:char-upcase` if *case* is `:up` and in a fashion analogous to `cl:string-capitalize` if *case* is `capital`.

3.6.8 Transposing Operations

`transpose` `[text.editing]` [Generic Function]
cursor unit direction Exchange the sequences of items defined by *unit* before and after *cursor*.

If *cursor* is within a *unit*, it is first moved to the boundary of that *unit* according to *direction*.

direction is either `:forward` or `:backward` and controls where *cursor* is positioned after the operation.

3.6.9 Filling Operations

`insert-words-fill` `[text.editing]` [Generic Function]

cursor words `&key` prefix suffix per-line-prefix fill-column Insert *words* at *cursor* with added line breaks according to *fill-column*.

words is a sequence of strings.

Each of *prefix*, *suffix* and *per-line-prefix* is a string if supplied.

fill-column, if supplied, is positive integer. If not supplied, *fill-column* defaults to the value of `text.editing:*fill-column*` which in turn is bound to `80` by default.

Roughly proceed as follows:

1. If it has been supplied, insert the *prefix* string.
2. For each string in *words*
   - If inserting the string at the current location would exceed *fill-column*, insert a line break and, if it has been supplied, insert the *per-line-prefix* string.
   - Unless the string is just punctuation, insert a space.
   - Insert the string.
3. If it has been supplied, insert the *suffix* string.

`fill-words` `[text.editing]` [Generic Function]

start-cursor end-cursor words `&key` prefix suffix per-line-prefix fill-column Replace the region between *start-cursor* and *end-cursor* by filling with *words*.

words is a sequence of strings.

prefix, *suffix*, *per-line-prefix* and *fill-column* behave as described for [Generic-Function text.editing|insert-words-fill], page 18.

3.6.10 Commenting Operations

`comment` `[text.editing]` [Generic Function]

cursor unit direction `&key` comment-syntax Comment buffer items at or around *cursor* according to *unit* and *direction*.

comment-syntax, if supplied, has to be a string which specifies the comment syntax to use. The default is `;;` if *unit* is [unit-line], page 6, and `#|` otherwise.

`uncomment` `[text.editing]` [Generic Function]

cursor unit direction Uncomment buffer items at or around *cursor* according to *unit* and *direction*.

3.6.11 Operations on Delimiter Pairs

The operations described in this section together with the operations of the Chapter 5 [Expressions], page 30, module allow structural editing of buffer contents in the sense that they transform a given buffer text that is syntactically valid to a new buffer text that is also syntactically valid. This is in contrast to general operations such as deleting a single item: Consider deleting a single item in the buffer text `(length "hi")`. If the deleted item is the `(`, the `)` or either of the `"`, the resulting buffer text is no longer a syntactically valid s-expression. In this example, deletion operations that preserve the validity can, for example, delete both characters of the `()` and `""` pairs simultaneously once the items in between have been deleted or delete the respective delimiter pair in a single operation, thus "raising" the formerly surrounded items up one level of expression nesting:

```
(length "hi") ⇒ (length "") ⇒ (length ) ⇒ () ⇒
(length "hi") ⇒ length "hi"
```

The above examples illustrate two kinds of operations:

- The first kind are operations which are intended as structure-preserving variants of "ordinary" operations, mainly of insertion and deletion operations since those affect the structural validity when applied to delimiter characters. These operations are described in the following section.
- The second kind, of which the "raising" operation is one example, consider the buffer text as an expression tree and perform modifications on that tree. Those operations are described as a part of the Chapter 5 [Expressions], page 30, module (see Section 5.2.4 [Expression Operations], page 33).

`no-closing-delimiter-error` `[text.editing]` [Class]

This error is signaled when an operation that requires a closing delimiter item is performed on a cursor that is not located at or near such an item.

`insert-delimiter-pair` `[text.editing]` [Generic Function]

cursor opening `&key` closing

Insert the delimiter character *opening* before *cursor* and insert the delimiter character *closing* after *cursor*.

If *closing* is not supplied, the closing delimiter character is determined by looking up the pair that has *opening* as the opening delimiter character in the set of known delimiter pairs.

The return value of this function is unspecified.

This function is intended to be used as a structure-preserving replacement for "ordinary" operations that insert an opening delimiter: When a user performs the operation for inserting the opening delimiter, operations for inserting some content and the operation for inserting the closing delimiter, the first operation has to insert both delimiters to avoid unbalanced delimiters:

```
| ⇒ (|) ⇒ ... ⇒ (foo|) ⇒ (foo)|
```

With this editing model, it is not allowed to insert an opening delimiter by itself. This is also true for closing delimiters (see [Generic-Function text.editing|move-past-closing-delimiter], page 20)

`maybe-move-past-closing-delimiter` `[text.editing]` [Generic Function]

cursor closing `&key` whitespace

If the item after *cursor* is equal to the character *closing*, move *cursor* forward past the closing delimiter.

whitespace which must be either `nil` or `:move-past` or `:delete` controls the behavior in case there are whitespace items between *cursor* and the item that is equal to *closing*. If *whitespace* is *nil* and there are such items, *cursor* does not move. If *whitespace* is *:move-past*, *cursor* moves past the whitespace items and past the closing delimiter. If *whitespace* is `:delete`, the whitespace items are deleted, and *cursor* is moved passed the closing delimiter.

Return true if *cursor* has moved and false otherwise.

`move-past-closing-delimiter` `[text.editing]` [Generic Function]
cursor closing `&key` whitespace

If the item after *cursor* is equal to the character *closing*, move *cursor* forward past the closing delimiter.

whitespace controls the behavior in case *cursor* is separated from the closing delimiter by whitespace. See [Generic-Function text.editing|maybe-move-past-closing-delimiter], page 19, for details.

The return value of this function is unspecified.

If the item after *cursor*, either immediately after or the first non-whitespace item after, is not equal to *cursor*, signal an error of type [Class text.editing|no-closing-delimiter-error], page 19.

This function is intended to be used as a structure-preserving replacement for "ordinary" operations that insert an "heterogeneous" (that is, for example `)` but not `"`) closing delimiter: When a user performs the operation for inserting the opening delimiter, operations for inserting some content and the operation for inserting the closing delimiter, the first operation has to insert both delimiters and the final operation simply has to move past the closing delimiter:

```
| ⇒ (|) ⇒ ... ⇒ (foo|) ⇒ (foo)|
```

With this editing model, it is never necessary and in fact never allowed to insert a closing delimiter by itself which is why this function signals an error if there is no closing delimiter to move past. This is also true for opening delimiters (see [Generic-Function text.editing|insert-delimiter-pair], page 19).

`move-past-closing-delimiter-or-insert-delimiter-pair` `[text.editing]` [Generic Function]
cursor delimiter `&key` whitespace

If the item after *cursor* is equal to the character *delimiter*, move *cursor* forward past the (assumed to be) closing delimiter. If the item after *cursor* is not equal to the character *delimiter*, insert *delimiter* before *cursor* and insert *delimiter* after *cursor*.

whitespace controls the behavior in case *cursor* is separated from the closing delimiter by whitespace. See [Generic-Function text.editing|maybe-move-past-closing-delimiter], page 19, for details.

Return true if a pair of delimiters has been inserted and false otherwise.

This function is intended to be used as a structure-preserving replacement for "ordinary" operations that insert a "homogeneous" (that is, for example `"` but not `)`) delimiter: When a user performs the operation for inserting *delimiter*, operations for inserting some content and the operation for inserting *delimiter* again, the first operation has to insert *delimiter* twice and the final operation simply has move past the second delimiter:

```
| ⇒ "|" ⇒ ... ⇒ "foo|" ⇒ "foo"|
```

With this editing model, it is never necessary and in fact never allowed to insert a single delimiter by itself which is why this function inserts a delimiter pair if there is no closing delimiter to move past.

Examples:

```
  (move-past-closing-delimiter-or-insert-delimiter-pair cursor #\") in
  "foo|"
⇒ "foo"|

  (move-past-closing-delimiter-or-insert-delimiter-pair cursor #\") in
  (length |)
⇒ (length "|")
```

`delete-delimiter-pair-or-item` `[text.editing]` [Generic Function]
cursor direction `&key` if-not-empty

If there is one, delete the pair of delimiter characters which contains *cursor*. Otherwise delete an item in *direction*.

In this context, a delimiter pair *contains cursor*, if either the delimiter characters surround *cursor* or if *direction* is `:forward` and the opening delimiter is the item after *cursor* or if *direction* is `:backward` and the closing delimiter is the item before *cursor*.

if-not-empty controls the behavior in case the operation is applied to a delimiter pair that is not empty in the sense that the opening and the closing delimiter surround other buffer items:

`nil` Do nothing, do not move *cursor* and do not delete any items.

`:move-past`
Move *cursor* past the “obstructing” delimiter and into the delimited content so that subsequent deletion operations will delete the content item by item until the delimited content is empty and the delimiter pair can be deleted.

`:delete-inside`
Do not move *cursor* but delete one item from the content that is delimited by the “obstructing” delimiter. This behavior can be repeated until the delimited content is empty and the delimiter pair can be deleted.

`a-function`
Call the supplied function with two arguments, *cursor* and an description of the “obstacle” which is either `:outside` or `:inside`. The function can delete items near *cursor* or move *cursor* as appropriate.

The return value of this function is unspecified.

This function is intended to be used as a structure-preserving replacement for “ordinary” operations that delete a single item: If the to-be-deleted item is a closing delimiter or an opening delimiter, the “opposite” delimiter has to be deleted in the same operation to maintain delimiter balance. If the to-be-deleted item is not a delimiter, the task can be delegated to the “ordinary” deletion operation.

Examples:

```
(delete-delimiter-pair-or-item cursor :forward) in
  |()
⇒ |

(delete-delimiter-pair-or-item cursor :forward) in
  (|)
⇒ |

(delete-delimiter-pair-or-item cursor :forward :if-not-empty if-not-
empty) in
  (foo|)
⇒ (foo|)  when if-not-empty is nil
⇒ (foo)|  when if-not-empty is :move-past
⇒ (fo|)   when if-not-empty is :delete-inside

(delete-delimiter-pair-or-item cursor :forward :if-not-empty if-not-
empty) in
  |(foo)
⇒ |(foo)  when if-not-empty is nil
⇒ (|foo)  when if-not-empty is :move-past
⇒ |(oo)   when if-not-empty is :delete-inside
```

`surround-with-delimiter-pair` [text.editing] [Generic Function]

cursor unit direction opening `&key` closing count

Surround the item sequence between *cursor* and the location that would result from moving *cursor count* times by *unit* in *direction* with the delimiters *opening* and *closing*.

If supplied, *closing* is a character that should be used as the closing delimiter of the pair. If *closing* is not supplied, the result of evaluating `(closing-delimiter` *`opening`*`)` is used.

If supplied, *count* controls how many units as indicated by *unit* should be surrounded by the inserted delimiters. If *count* is not supplied, a single unit is used.

The return value of this function is unspecified.

Examples:

```
(surround-with-delimiter-pair cursor [unit-word], page 6, :forward #\") in
  |foo bar
⇒ "|foo" bar

(surround-with-delimiter-pair cursor [unit-word], page 6, :forward #\" :count 2)
  |foo bar
⇒ "|foo bar"

(surround-with-delimiter-pair cursor [unit-word], page 6, :backward #\( :count 2)
  foo bar| baz
⇒ (foo bar|) baz
```

4 Search

This chapter describes functions for ordinary search as well as incremental search.

The *ordinary search* operation accepts a query sequence and moves the [term-point], page 3, cursors from their current locations either forward or backward to the nearest occurrence of the query sequence in the buffer, if any.

The *incremental search* operation, on the other hand, maintains a mutable current query sequence which the client can extend or truncate, as well as a set of current matches. Extending or truncating the query shrinks or grows the set of matches as fewer or more subsequences in the buffer match the current query sequence. The [term-point], page 3, cursors are typically moved to the locations of certain matches, for example the nearest match following the location of a given cursor, during the incremental search. The association between cursors and matches can be changed so that point cursors can “jump” from one match to the next or previous match.

The search functionality is provided as a separate module which uses the `text.editing.search` package.

4.1 Search Concepts

The high-level overview for an interactive, incremental search operation in a buffer is something like this:

1. The client performs the operation to start an incremental search in a given buffer which creates a [term-buffer-search-state], page 24, and [term-site-search-state], page 24, for that buffer. The query sequence and set of [term-match], page 24, are initially empty.
2. The client repeatedly updates the search state:
    1. Based on user commands, the client uses operations to extend or truncate the query sequence or to move point cursors between matches or to change parameters of the search operation like case sensitivity.
    2. The set of matches is updated or recomputed. The associations between point cursors and matches are updated. Point cursors are moved to new locations.
    3. The client displays the updated search state, in particular the current set of matches and point cursor locations to the user.
3. Based on user commands, the client finishes or aborts the incremental search to either leave all point cursors where the previous operations positioned them or reset all point cursors to their locations prior to the incremental search. Alternatively, the client may finish the incremental search by converting the final set of matches to sites.

Buffer Search State

During search operations, a *buffer search state* is associated with the buffer in which the operation is performed. This search state consists of parameters, state and results of the search operation such as:

- The buffer region in which the search operation is performed.
- The current query sequence of which occurrences should be found.
- Parameters like the case sensitivity of the search operation.
- The current set of [term-match], page 24.

Match

A *match* consists of a start cursor and an end cursor which delimit a sequence of buffer items that matches the query string. A match has an associated previous match and an associated next match either or both of which can be the match itself (if the set of current matches has a single element).

Site Search State

During search operations, a *site search state* which consists of a start location, that is the location of the point cursor prior to the start of the search, and a current match is associated with each site.

4.2 Search Dictionary

4.2.1 Search Conditions

The conditions of the following types are signaled by functions in the search module:

`already-in-incremental-search-error` `[text.editing.search]` [Class]

An error of this type is signaled when an operation that starts an incremental search is performed when an incremental search is already associated with the buffer.

`not-in-incremental-search-error` `[text.editing.search]` [Class]

An error of this type is signaled when an operation that works only in the context of an incremental search is performed when no incremental search is associated with the buffer.

`no-next-match-error` `[text.editing.search]` [Class]

An error of this type is signaled when the [Generic-Function text.editing.search|next-match], page 29, operation is performed on a cursor for which there is no following match and `wrap-around` is false or there are no matches at all.

`no-previous-match-error` `[text.editing.search]` [Class]

An error of this type is signaled when the [Generic-Function text.editing.search|previous-match], page 29, operation is performed on a cursor for which there is no preceding match and `wrap-around` is false or there are no matches at all.

4.2.2 Search State Protocol

The search state protocol specifies generic functions that operate on a [term-buffer-search-state], page 24, while an incremental search is being performed.

The search state protocol extends the Section 3.1 [Detach Protocol], page 8, that is, search states can (and must) be detached. When detached, a search state detaches all its matches.

`start` `[text.editing.search]` [Generic Function]
: search-state Return a cursor that represents the buffer location at which the incremental search represented by *search-state* started.

`query` `[text.editing.search]` [Generic Function]
: search-state Return the current query sequence for *search-state*.

  The returned sequence must not be modified. [Generic-Function text.editing.search|extend-query], page 28, and [Generic-Function text.editing.search|truncate-query], page 28, must be used instead.

`case-mode` `[text.editing.search]` [Generic Function]
: search-state Return the case mode, which is either `:ignore` or `:match`, of *search-state*.

`(setf case-mode)` `[text.editing.search]` [Generic Function]
: new-value search-state Set the case mode of *search-state* to *new-value* which must be either `:ignore` or `:match`.

  Calling this function causes *search-state* to be rebuilt via [Generic-Function text.editing.search|rebuild-state], page 25.

`match-count` `[text.editing.search]` [Generic Function]
: search-state Return the number of [term-match], page 24, contained in *search-state*.

`map-matches` `[text.editing.search]` [Generic Function]
: function search-state Call *function* with each [term-match], page 24, in *search-state*.

`matches` `[text.editing.search]` [Generic Function]
: search-state Return a sequence of the [term-match], page 24, in *search-state*.

`add-match` `[text.editing.search]` [Generic Function]
: search-state buffer start end Add a match in *buffer* between the cursors *start* and *end* to *search-state*.

  Return the newly created match object.

`remove-match` `[text.editing.search]` [Generic Function]
: search-state buffer match Remove *match* from *search-state*.

  The default method on this generic function Section 3.1 [Detach Protocol], page 8, *match* and, if a site search state refers to *match*, replaces that reference with another match.

`initial-matches` `[text.editing.search]` [Generic Function]
: search-state Compute the initial matches for *search-state* based on the [Generic-Function text.editing.search|start], page 24, cursor and the [Generic-Function text.editing.search|query], page 25, sequence.

  Call [Generic-Function text.editing.search|add-match], page 25, for each computed match.

`rebuild-state` `[text.editing.search]` [Generic Function]
: search-state Rebuild *search-state* from scratch, that is remove the current matches and use [Generic-Function text.editing.search|initial-matches], page 25, to compute new matches.

`finish` `[text.editing.search]` [Generic Function]
search-state Finish the incremental search represented by *search-state* leaving all [term-point], page 3, cursors at their current locations.

`abort` `[text.editing.search]` [Generic Function]
search-state Abort the incremental search represented by *search-state* moving all [term-point], page 3, cursors the locations at which they were positioned before the incremental search started.

`description` `[text.editing.search]` [Generic Function]
search-state `&key` comment

4.2.3 Match Protocol

The match protocol extends the Section 3.1 [Detach Protocol], page 8, that is matches can (and must) be detached.

`next` `[text.editing.search]` [Generic Function]
match Return the next match after *match* or `nil` if there is no next match.

`previous` `[text.editing.search]` [Generic Function]
match Return the previous match before *match* or `nil` if there is no previous match.

`start [text.editing.search]` *match* [Generic Function]
Return a cursor which marks the start of *match*.

`end` `[text.editing.search]` [Generic Function]
match Return a cursor which marks the end of *match*.

`item-matches-p` `[text.editing.search]` [Generic Function]
state match query-item Indicate whether *match* extended with *query-item* matches the buffer sub-sequence corresponding to *match*.

4.2.4 Site Search State Protocol

The site search state protocol extends the Section 3.1 [Detach Protocol], page 8, that is site search states can (and must) be detached.

`start [text.editing.search]` *site-search-state* [Generic Function]
Return a cursor which indicates the location at which the [term-point], page 3, cursor of the site associated with *site-search-state* was positioned before the start of the incremental search.

`match` `[text.editing.search]` [Generic Function]
site-search-state Return the [term-match], page 24, associated with *site-search-state* or `nil`.

`site-search-state` `[text.editing.search]` [Class]
Instances of this class store a start cursor and a [term-match], page 24, that should be associated with a [term-site], page 3, in the context of an incremental search.

4.2.5 Buffer Search State Protocol

The sole purpose of the buffer search state protocol is retrieving the [term-buffer-search-state], page 24, associated with a given buffer:

`search-state` `[text.editing.search]` [Generic Function]

buffer Return the [term-buffer-search-state], page 24, associated with *buffer* or `nil` if there is none.

`search-state-mixin` `[text.editing.search]` [Class]

This class is intended to be mixed into buffer classes that implement the buffer search state protocol.

Methods on [Generic-Function text.editing.search|add-match], page 25, and [Generic-Function text.editing.search|remove-match], page 25, specialized to this class take care of associating sites (or the site) of the buffer with the nearest match(es) as matches are added and removed.

4.2.6 Ordinary Search Operations

`search` `[text.editing.search]` [Generic Function]

target query direction Search for occurrences of *query* in the underlying buffer of *target* from each site in *direction*.

target can be a buffer, a site or a cursor. In any of those cases, the search will include all sites of the underlying buffer.

query is the sequence of items to search for.

direction can be either `:forward` or `:backward` and controls in which direction point cursors should move towards the nearest occurrence of *query* in the buffer.

TODO case mode etc.

4.2.7 Incremental Search Operations

`incremental-search` `[ext.editing.search]` [Generic Function]

target direction Start an incremental search in the underlying buffer of *target*.

target can be a buffer, a site or a cursor. In any of those cases, the incremental search will include all sites of the underlying buffer.

Create a [term-buffer-search-state], page 24, and associate it with the underlying buffer of *target*. The search state starts out with an empty query sequence and an empty set of [term-match], page 24. For each [term-site], page 3, in the buffer, create a [term-site-search-state], page 24, that is initially not associated with any match (as there are no matches initially). Return the created buffer search state.

If there already is an incremental search associated with *target*, signal an error of type [Class text.editing.search|already-in-incremental-search-error], page 24.

`finish-incremental-search` `[text.editing.search]` [Generic Function]

target Finish the incremental search associated with *target*

Keep all point cursors at the locations to which they were moved due to search operations.

target can be a buffer, a site or a cursor. In any of those cases, the incremental search will include all sites of the underlying buffer.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

`abort-incremental-search` `[text.editing.search]` [Generic Function]

target Abort the incremental search associated with *target*.

In particular, move all involved point cursors back to the locations at which they resided before the incremental search started.

target can be a buffer, a site or a cursor. In any of those cases, the incremental search will include all sites of the underlying buffer.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

`convert-matches-to-sites` `[text.editing.search]` [Generic Function]

target Finish the search involving *target*, and add a [term-site], page 3, at the location of each [term-match], page 24, (except for the match that is associated with the primary site).

target can be a buffer, a site or a cursor. In any of those cases, the operation will affect the incremental search state associated with the underlying buffer.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

> **warning:** Performing this operation on an incremental search state that already involves more than one site is currently not supported. A suitable behavior for that situation may be specified in the future.

`extend-query` `[text.editing.search]` [Generic Function]

target item Add *item* at the end of the query sequence of the incremental search associated with *target*.

target can be a buffer, a site or a cursor. In any of those cases, the operation will affect the incremental search state associated with the underlying buffer.

Extending the query sequence can lead to matches being modified or removed from the current set of matches. Point cursors can also move to different locations as a result.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

`truncate-query` `[text.editing.search]` [Generic Function]

target `&key` count Remove *count* items from the end of the query sequence of the incremental search associated with *target*.

target can be a buffer, a site or a cursor. In any of those cases, the operation will affect the incremental search state associated with the underlying buffer.

count is a positive integer no greater than the length of the query sequence.

Truncating the query sequence can lead to matches being modified and new matches being added to the current set of matches. Point cursors can also move to different locations as a result.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

`next-match` `[text.editing.search]` [Generic Function]

target `&key` wrap-around In the search involving *target*, move all point cursors to the respective next match.

target can be a buffer, a site or a cursor. In any of those cases, the incremental search will include all sites of the underlying buffer.

wrap-around controls the behavior in case there is no next match when a point cursor should be moved to the next match. If there is no next match and *wrap-around* is false or there are no matches at all, signal an error of type [Class text.editing.search|no-next-match-error], page 24.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

`previous-match` `[text.editing.search]` [Generic Function]

target `&key` wrap-around In the search involving *target*, move all point cursors to the previous match.

target can be a buffer, a site or a cursor. In any of those cases, the incremental search will include all sites of the underlying buffer.

wrap-around controls the behavior in case there is no previous match when a point cursor should be moved to the previous match. If there is no previous match and *wrap-around* is false or there are no matches at all, signal error of type [Class text.editing.search|no-previous-match-error], page 24.

If there is no incremental search associated with *target*, signal an error of type [Class text.editing.search|not-in-incremental-search-error], page 24.

5 Expressions

5.1 Expressions Concepts

This module adds support for operating on *expressions* which are basically nodes in a (concrete) syntax tree constructed *by clients of this library* from the text of the buffer. To this end, the module defines two [term-unit], page 3: [unit-expression], page 5, and [unit-toplevel-expression], page 6, the semantics of which depend on the implementations of Section 5.2.2 [Expression Node Protocol], page 31, Section 5.2.3 [Expression Tree Protocol], page 32, that clients of this library must provide, for example by parsing the source code of a buffer and constructing a (concrete) syntax tree. The expression-based units work with the usual operations for Section 3.6.1 [Motion Operations], page 14, Section 3.6.3 [Deletion Operations], page 15, and so on. In addition, this module provides Section 5.2.4 [Expression Operations], page 33, such as splitting and joining that work with the expression-based units.

The concept of an expression must appear very vague at this point and this vagueness is in part intrinsic since the exact nature of expressions for a given buffer is defined, as mentioned above, by the client. However, we can still make the concept as concrete as possible by defining that an expression is a node in a tree and has the following properties

- A *start location* which is a buffer location expressed as a line number and a column number.
- An *end location* which is a buffer location that is expressed as a line number and a column number and follows the start location.
- A possibly empty sequence of child expressions such that:
  - For each child expression the start and end locations are within the buffer delimited by the start and end location of the parent expression.
  - for a child c_2 that follows a child c_1 in the sequence of children, the start location of c_2 must be equal to or greater than the end location of c_1.

The following figure shows an example buffer text and a possible expression tree for that text:

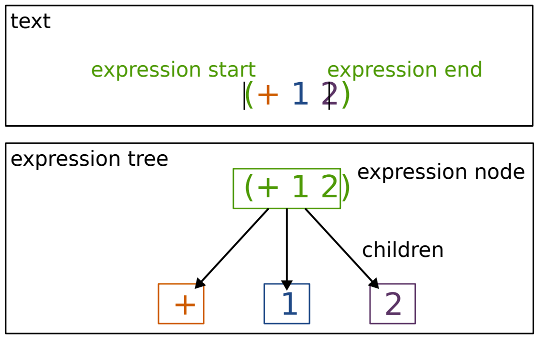

Figure 5.1: Example buffer text and a possible expression tree for that text. Corresponding parts are indicated by matching colors. Child nodes should be considered from left to right.

5.2 Expressions Dictionary

5.2.1 Expression Conditions

`cursor-not-inside-expression-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression which [term-contains], page 32, cursor is attempted and no such expression exists.

`no-expression-after-cursor-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression after the cursor is attempted and no such expression exists.

`no-expression-before-cursor-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression before the cursor is attempted and no such expression exists.

`expression-at-toplevel-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires a non-toplevel expression before or after the cursor is attempted and no such expression exists.

`expression-does-not-have-children-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression with children is attempted on an expression that does not have any children.

`no-expression-after-expression-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression after some designated expression is attempted and no such expression exists.

`no-expression-before-expression-error` [text.editing.expression] [Class]

This error is signaled when an operation that requires an expression before some designated expression is attempted and no such expression exists.

5.2.2 Expression Node Protocol

The purpose of the expression node protocol is to allow this library to inspect nodes of an expression tree that is constructed and managed by a client of this library. Accordingly, this library does not define methods on the following generic functions. Instead, clients are expected to define methods that are suitable for the expression representation used by the respective client.

`range` [text.editing.expression] [Generic Function]

expression Return the source range, that is the start and end buffer positions, of *expression* as four values

1. The start line number of *expression*
2. The start column number of *expression*
3. The end line number of *expression*
4. The end column number of *expression*

`children` `[text.editing.expression]` [Generic Function]

expression Return a possibly empty sequence of child expressions of *expression*. The elements of the returned sequence are ordered according to their respective start locations and do not overlap.

5.2.3 Expression Tree Protocol

The purpose of the expression tree protocol is to allow this library to query an expression tree that is constructed and managed by a client of this library. Accordingly, this library does not define all required methods on the following generic functions. Instead, clients are expected to define methods (or at least one method) that are suitable for the expression tree representation used by the respective client. For this protocol, it is sufficient for each client to define a method that is suitable for the respective expression tree presentation on the generic function `map-expressions-containing-cursor-using-buffer`.

For the description of the protocol functions in this section, we need the following definition: Let r_s, the *start relation*, and r_e, the *end relation*, be either $<$ or $\leq$ respectively. An expression with start location s and end location e *contains* a cursor c with respect to r_s and r_e iff $r_s(s, c) \wedge r_e(c, e)$.

`map-expressions-containing-cursor` `[text.editing.expression]` [Generic Function]

function cursor syntax-tree `&key` start-relation end-relation

Call *function* for each [term-expression], page 30, in the buffer of *cursor* that [term-contains], page 32, *cursor* with respect to *start-relation* and *end-relation*. The return value of this function is unspecified. For more information about the parameters and behavior, see [Generic-Function text.editing.expression|map-expressions-containing-cursor-using-buffer], page 32.

The default method on this generic function calls [Generic-Function text.editing.expression|map-expressions-containing-cursor-using-buffer], page 32, with the buffer that *cursor* is associated with.

`map-expressions-containing-cursor-using-buffer` [Generic Function]
`[text.editing.expression]`

function buffer cursor syntax-tree `&key` start-relation end-relation

Call *function* for each [term-expression], page 30, in *buffer* that [term-contains], page 32, *cursor* with respect to *start-relation* and *end-relation*. If multiple expressions contain *cursor*, the call for a given expression precedes the calls for the ancestors of that expression in the expression tree, that is innermost or leaf expressions are processed first, outermost or toplevel expressions are processed last.

syntax-tree selects the syntax tree in which expressions should be considered. For example, the buffer of *cursor* may have an associated concrete syntax tree and also an abstract syntax tree.

Both *start-relation* and *end-relation* which must be either the symbol `<` or the symbol `<=` select the respective relation.

The return value of this function is unspecified.

`expressions-containing-cursor` `[text.editing.expression]` [Generic Function]

cursor syntax-tree `&key` start-relation end-relation count

Return a possibly empty sequence of [term-expression], page 30, in the buffer of *cursor* that [term-contains], page 32, *cursor* with respect to *start-relation* and *end-relation*. If *count* is supplied, its value must be a non-negative integer. In that case, the number of elements in the returned sequence is limited to that number. For more information about the behavior and the other parameters, see [Generic-Function text.editing.expression|map-expressions-containing-cursor-using-buffer], page 32.

The default method on this generic function calls [Generic-Function text.editing.expression|map-expressions-containing-cursor], page 32, and collects the provided expressions, stopping when *count* expressions have been collected if applicable.

`innermost-expression-containing-cursor` `[text.editing.expression]` [Function]
cursor syntax-tree `&key` start-relation end-relation

Return the innermost [term-expression], page 30, in the buffer of *cursor* that [term-contains], page 32, *cursor* with respect to *start-relation* and *end-relation* or `nil` if there is no such expression. For more information about the behavior and the other parameters, see [Generic-Function text.editing.expression|map-expressions-containing-cursor-using-buffer], page 32.

`outermost-expression-containing-cursor` `[text.editing.expresssion]` [Function]
cursor syntax-tree `&key` start-relation end-relation

Return the outermost [term-expression], page 30, in the buffer of *cursor* that [term-contains], page 32, *cursor* with respect to *start-relation* and *end-relation* or `nil` if there is no such expression. For more information about the behavior and the other parameters, see [Generic-Function text.editing.expression|map-expressions-containing-cursor-using-buffer], page 32.

5.2.4 Expression Operations

The operations described in this section are designed to, in conjunctions with the Section 3.6.11 [Operations on Delimiter Pairs], page 18, enable structural editing of buffer contents.

`raise` `[text.editing.expression]` [Generic Function]
cursor unit direction

Raise the innermost [term-expression], page 30, *e* which [term-contains], page 32, follows or precedes *cursor* by deleting the buffer items that make up the parent of *e* and the siblings of *e* but preserving the buffer items that make up *e*.

unit must be [unit-expression], page 5, at the moment.

direction which must be either `:forward` or `:backward` controls whether the expression *e* should follow or precede *cursor*.

The return value of this function is unspecified.

After the operation, *cursor* resides in the same relative location with respect to the preserved buffer item as before the operation.

When there is no such expression, signal [Class text.editing.expression|no-expression-after-cursor-error], page 31, or [Class text.editing.expression|no-expression-before-cursor-error], page 31, depending on *direction*.

Examples:

```
(raise cursor [unit-expression], page 5, :forward) in
  1 2 (3 4 |5 6) 7 8
⇒ 1 2 |5 7 8

(raise cursor [unit-expression], page 5, :backward) in
  1 2 (3 4 |5 6) 7 8
⇒ 1 2 |5 7 8
```

`splice` [text.editing.expression] [Generic Function]
cursor unit direction

Splice [term-expression], page 30, e_1 to e_n that follow or precede *cursor* by replacing the buffer items that make up the expression which [term-contains], page 32, *cursor* with the buffer items that make up e_1 to e_n.

unit must be [unit-expression], page 5, at the moment.

If *direction* is `:forward`, expressions that follow *cursor* are preserved. If *direction* is `:backward`, expressions that precede *cursor* are preserved. If *direction* is `nil`, expressions that follow *and* precede *cursor* are preserved.

The return value of this function is unspecified.

After the operation, *cursor* resides in the same relative location with respect to the preserved buffer items as before the operation.

When there is no such expression, signal [Class text.editing.expression|cursor-not-inside-expression-error], page 31.

Examples:

```
(splice cursor [unit-expression], page 5, nil) in
  1 2 (3 4 |5 6) 7 8
⇒ 1 2 3 4 |5 6 7 8

(splice cursor [unit-expression], page 5, :forward) in
  1 2 (3 4 |5 6) 7 8
⇒ 1 2 |5 6 7 8

(splice cursor [unit-expression], page 5, :backward) in
  1 2 (3 4 |5 6) 7 8
⇒ 1 2 3 4| 7 8
```

`split` [text.editing.expression] [Generic Function]
cursor unit

Split the innermost [term-expression], page 30, or toplevel expression *e* that [term-contains], page 32, *cursor* by inserting the a copy of the buffer items that make up the closing delimiter of *e* before *cursor* and a copy of the buffer items that make up the opening delimiter of *e* after *cursor*.

If *unit* is [unit-expression], page 5, *e* is the innermost expression that contains *cursor*. If *unit* is [unit-toplevel-expression], page 6, *e* is the toplevel expression that contains *cursor*. Other values of *unit* are not supported at the moment.

The return value of this function is unspecified.

After the operation, *cursor* resides between the buffer items that make up the two new expressions.

When there is no such expression, signal [Class text.editing.expression | cursor-not-inside-expression-error], page 31.

Examples:

```
(split cursor [unit-expression], page 5) in
  (1 (2 |3) 4)
⇒ (1 (2 )|(3) 4)

(split cursor [unit-toplevel-expression], page 6) in
  (1 (2 |3) 4)
⇒ (1 (2 ))|((3) 4)
```

`join` `[text.editing.expression]` [Generic Function]
cursor unit

Join the innermost [term-expression], page 30, e_1 and e_2 that precede and follow *cursor* respectively by deleting the buffer items that make up the closing delimiter of e_1 and the buffer items that make up the opening delimiter of e_2.

unit must be [unit-expression], page 5, at the moment.

The return value of this function is unspecified.

After the operation, *cursor* resides between the buffer items that make up the two child expressions of the joined expressions that were the last and first child of e_1 and e_2 respectively.

When there is no expression either following or preceding *cursor*, signal [Class text.editing.expression | no-expression-after-cursor-error], page 31, or [Class text.editing.expression | no-expression-before-cursor-error], page 31, respectively.

Example:

```
(join cursor [unit-expression], page 5) in
  (1 2) |(3 4)
⇒ (1 2 |3 4)
```

`eject` `[text.editing.expression]` [Generic Function]
cursor unit direction

Assuming an expression e (or possibly other unit as specified by *unit*) contains *cursor*, depending on *direction*, move the last child of e after the end of e or move the first child of e before the start of e.

unit must be [unit-expression], page 5, at the moment.

direction must be either `:forward` or `:backward`. If *direction* is `:forward`, move the last child of e after the end of e. If *direction* is `:backward`, move the first child of e before the start of e

After the operation, *cursor* is still contained in e.

If *cursor* is not contained in any expression, signal [Class text.editing.expression | cursor-not-inside-expression-error], page 31. If the expression e does not have any children, signal [Class text.editing.expression|expression-does-not-have-children-error], page 31.

Examples:

```
(eject cursor [unit-expression], page 5, :forward) in
  (1 2 3 4|)
⇒ (1 2 3|) 4

(eject cursor [unit-expression], page 5, :backward) in
  (1 2 |3 4)
⇒ 1 (2 |3 4)
```

absorb [text.editing.expression] [Generic Function]
cursor unit direction

Assuming an expression *e* contains *cursor* and a "target" expression *t* follows or precedes *e*, depending on *direction*, move *t* into *e*.

- If *e* is a toplevel expression, try to find the target expression before or after *e* in the sequence of toplevel expressions.
- If *e* is not toplevel expression, try to find the target expression before or after *e* in the children of *p*, the parent of *e*. If there is no suitable target expression among the children of *p*, repeat the process with *p* instead of *e*. In other words, look for a suitable target expression before or after each ancestor of *e* proceeding from the innermost expression (which is *e*) to the toplevel ancestor of *e*.

unit must be [unit-expression], page 5, at the moment.

direction must be either `:forward` or `:backward`. If *direction* is `:forward`, move a "target" expression that follows *e* into *e* as the last child. If *direction* is `:backward`, move a "target" expression that precedes *e* into *e* as the first child.

After the operation, *cursor* is at the same location relative to the unmodified boundary of *e* as before.

If *cursor* is not contained in any expression, signal [Class text.editing.expression|cursor-not-inside-expression-error], page 31. When there is no expression either following or preceding (an ancestor of) *e*, signal [Class text.editing.expression|no-expression-after-expression-error], page 31, or [Class text.editing.expression|no-expression-before-expression-error], page 31, respectively.

Examples:

```
(absorb cursor [unit-expression], page 5, :forward) in
  (1 2 | 3) 4
⇒ (1 2 | 3 4)

(absorb cursor [unit-expression], page 5, :backward) in
  1 (|2 3 4)
⇒ (1 |2 3 4)
```

delete-semi-line-or-expressions [text.editing.expression] [Generic Function]
cursor direction

For *cursor* located on line *l*, delete to either the end of *l* or to the end of some expression which starts on *l* so that delimiters are kept balanced.

direction controls whether to delete from *cursor* towards the end of the line or towards the beginning of the line. At the moment, *direction* has to be `:forward`.

Examples:

```
(delete-semi-line-or-expressions cursor :forward) in
  (1 |2 (3 4) 5)
⇒ (1 |)

(delete-semi-line-or-expressions cursor :forward) in
  (1 |2 (3
         4)
        5)
⇒ (1 |
        5)

(delete-semi-line-or-expressions cursor :forward) in
  1 |2 (3
        4)
  5
⇒ 1 |
  5
```

6 Equivalent Emacs Commands

6.1 Motion

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`edit:move`	`edit:item`	`:forward`	`forward-char` (*C-f*)
`edit:move`	`edit:item`	`:backward`	`backward-char` (*C-b*)
`edit:move`	`edit:line`	`:forward`	`next-line` (*C-n*)
`edit:move`	`edit:line`	`:backward`	`previous-line` (*C-p*)
`edit:move`	`edit:semi-line`	`:forward`	`end-of-line`
`edit:move`	`edit:semi-line`	`:backward`	`beginning-of-line`
`edit:move`	`edit:line-boundary`	`:forward`	`end-of-line` (*C-a*)
`edit:move`	`edit:line-boundary`	`:backward`	`beginning-of-line` (*C-e*)
`edit:move`	`edit:buffer`	`:forward`	`end-of-buffer`
`edit:move`	`edit:buffer`	`:backward`	`beginning-of-buffer`
`edit:move`	`edit:semi-buffer`	`:forward`	`end-of-buffer`
`edit:move`	`edit:semi-buffer`	`:backward`	`beginning-of-buffer`
`edit:move`	`edit:buffer-boundary`	`:forward`	`end-of-buffer` (*M->*)
`edit:move`	`edit:buffer-boundary`	`:backward`	`beginning-of-buffer` (*M-<*)
`edit:move`	`edit:word`	`:forward`	`forward-word` (*M-f*)
`edit:move`	`edit:word`	`:backward`	`backward-word` (*M-b*)
`edit:move`	`edit:sentence`	`:forward`	`forward-sentence` (*M-e*)
`edit:move`	`edit:sentence`	`:backward`	`backward-sentence` (*M-a*)
`edit:move`	`edit:paragraph`	`:forward`	`forward-paragraph` (*M-}*)
`edit:move`	`edit:paragraph`	`:backward`	`backward-paragraph` (*M-{*)
`edit:move`	`edit::page`	`:forward`	`forward-page` (*C-x]*)
`edit:move`	`edit::page`	`:backward`	`backward-page` (*C-x [*)
`edit:move`	`text.editing.expression:expression`	`:forward`	`forward-sexp` (*C-M-f*)
`edit:move`	`text.editing.expression:expression`	`:backward`	`backward-sexp` (*C-M-b*)
`edit:move`	`text.editing.expression:toplevel-expression`	`:forward`	`end-of-defun` (*C-M-e*)
`edit:move`	`text.editing.expression:toplevel-expression`	`:backward`	`beginning-of-defun` (*C-M-a*)

6.2 Deletion

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`edit:delete`	`edit:region`	`:forward`	`kill-region beg end`
`edit:delete`	`edit:region`	`:backward`	`kill-region beg end`
`edit:delete`	`edit:item`	`:forward`	`delete-char 1` (*C-d*)
`edit:delete`	`edit:item`	`:backward`	`delete-backward-char 1` (*<backspace>*)
`edit:delete`	`edit:line`	`:forward`	`kill-line` (*C-k*)
`edit:delete`	`edit:line`	`:backward`	`kill-line` (*with 0 prefix (C-0 C-k)*)
`edit:delete`	`edit:semi-line`	`:forward`	`kill-line` (*C-k*)

`edit:delete`	`edit:semi-line`	`:backward`	`kill-line 0` (*with 0 prefix (C-0 C-k)*)
`edit:delete`	`edit:buffer`	`:forward`	`erase-buffer`
`edit:delete`	`edit:buffer`	`:backward`	`erase-buffer`
`edit:delete`	`edit:word`	`:forward`	`kill-word 1` (*M-d*)
`edit:delete`	`edit:word`	`:backward`	`backward-kill-word 1` (*M-<backspace>*)
`edit:delete`	`edit:sentence`	`:forward`	`kill-sentence` (*M-k*)
`edit:delete`	`edit:sentence`	`:backward`	`backward-kill-sentence` (*C-x DEL*)
`edit:delete`	`edit:paragraph`	`:forward`	`kill-paragraph 1`
`edit:delete`	`edit:paragraph`	`:backward`	`backward-kill-paragraph 1`
`edit:delete`	`text.editing.expression:expression`	`:forward`	`kill-sexp 1` (*C-M-k*)
`edit:delete`	`text.editing.expression:expression`	`:backward`	`backward-kill-sexp 1` (*C-M-<backspace>*)

Operation	**Equivalent Emacs Command**
`edit:delete-indentation`	`delete-indentation`

Operation	**Equivalent Emacs Command**
`edit:delete-trailing-whitespace`	`delete-trailing-whitespace`

6.3 Marking

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`edit:mark-object`	`edit:buffer`	`:forward`	`mark-whole-buffer` (*C-x h*)
`edit:mark-object`	`edit:semi-buffer`	`:forward`	`mark-end-of-buffer`
`edit:mark-object`	`edit:semi-buffer`	`:backward`	`mark-beginning-of-buffer`
`edit:mark-object`	`edit:word`	`:forward`	`mark-word` (*M-@*)
`edit:mark-object`	`edit:sentence`	`:forward`	`mark-end-of-sentence`
`edit:mark-object`	`edit:sentence`	`:backward`	`mark-beginning-of-sentence`
`edit:mark-object`	`edit:paragraph`	`:forward`	`mark-paragraph` (*M-h*)
`edit:mark-object`	`edit::page`	`:forward`	`mark-page` (*C-x C-p*)
`edit:mark-object`	`text.editing.expression:expression`	`:forward`	`mark-sexp` (*C-M-@*)
`edit:mark-object`	`text.editing.expression:toplevel-expression`	`:forward`	`mark-defun` (*C-M-h*)

6.4 Transformation

Operation	**Unit**	**Direction**	**Case**	**Equivalent Emacs Command**
`edit:change-case`	`edit:region`	`:forward`	`:up`	`upcase-region beg end` (*C-x C-u*)
`edit:change-case`	`edit:region`	`:forward`	`:down`	`downcase-region beg end` (*C-x C-l*)
`edit:change-case`	`edit:region`	`:forward`	`:capital`	`capitalize-region beg end`

`edit:change-case`	`edit:item`	`:forward`	`:up`	`upcase-char 1` (*Emacs does not move point*)
`edit:change-case`	`edit:item`	`:forward`	`:down`	`downcase-char 1` (*Emacs does not move point*)
`edit:change-case`	`edit:item`	`:forward`	`:capital`	`capitalize-char 1` (*Emacs does not move point*)
`edit:change-case`	`edit:word`	`:forward`	`:up`	`upcase-word 1` (*M-u*)
`edit:change-case`	`edit:word`	`:forward`	`:down`	`downcase-word 1` (*M-l*)
`edit:change-case`	`edit:word`	`:forward`	`:capital`	`capitalize-word 1` (*M-c*)

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`edit:transpose`	`edit:item`	`:forward`	`transpose-chars 1` (*C-t*)
`edit:transpose`	`edit:line`	`:forward`	`transpose-lines` (*C-x C-t*)
`edit:transpose`	`edit:word`	`:forward`	`transpose-words 1` (*M-t*)
`edit:transpose`	`edit:sentence`	`:forward`	`transpose-sentences 1`
`edit:transpose`	`edit:paragraph`	`:forward`	`transpose-paragraphs 1`
`edit:transpose`	`text.editing.expression:expression`	`:forward`	`transpose-sexps` (*C-M-t*)

6.5 Structure Editing

Operation	**Unit**	**Equivalent Emacs Command**
"*not implemented*"	"*not implemented*"	`paredit-wrap-sexp argument open close`

Operation	**Unit**	**Equivalent Emacs Command**
`text.editing.expression:raise`	`text.editing.expression:expression`	`paredit-raise-sexp` (*M-r*)

Operation	**Unit**	**Equivalent Emacs Command**
`text.editing.expression:splice`	`text.editing.expression:expression`	`paredit-splice-sexp` (*M-s*)

Operation	**Unit**	**Equivalent Emacs Command**
`text.editing.expression:split`	`text.editing.expression:expression`	`paredit-split-sexp` (*M-S*)
`text.editing.expression:split`	`text.editing.expression:toplevel-expression`	?

Operation	**Unit**	**Equivalent Emacs Command**
`text.editing.expression:join`	`text.editing.expression:expression`	`paredit-join-sexp` (*M-J*)

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`text.editing.expression:eject`	`text.editing.expression:expression`	`:forward`	`paredit-forward-barf-sexp` (*C-<left>*)
`text.editing.expression:eject`	`text.editing.expression:expression`	`:backward`	`paredit-backward-barf-sexp` (*C-M-<right>*)

Operation	**Unit**	**Direction**	**Equivalent Emacs Command**
`text.editing.expression:absorb`	`text.editing.expression:expression`	`forward`	`paredit-forward-slurp-sexp` (*`C-<right>`*) (*Emacs moves point differently*)
`text.editing.expression:absorb`	`text.editing.expression:expression`	`backward`	`paredit-backward-slurp-sexp` (*`C-M-<left>`*) (*Emacs moves point differently*)

Concept index

Function and macro and variable and type index