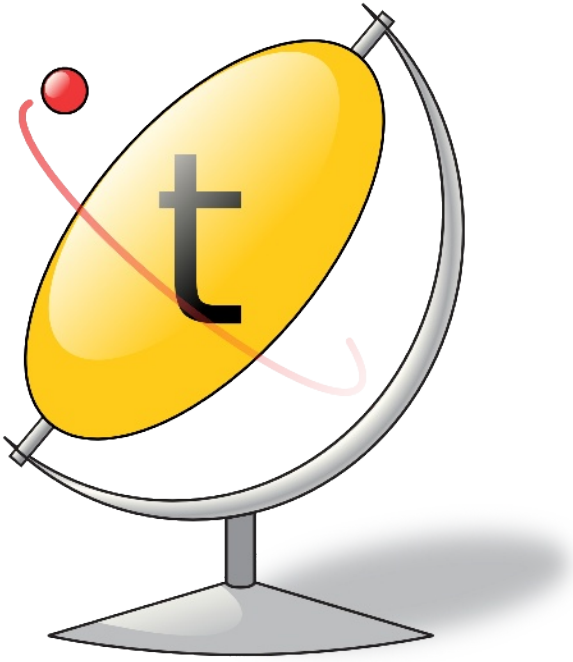
Textadept

Quick Reference

A fast, minimalist, and remarkably extensible text editor



Fifth Edition

Mitchell

Textadept Quick Reference

Textadept is a fast, minimalist, and remarkably extensible cross-platform text editor for programmers. This quick reference contains a wealth of knowledge on how to script and configure Textadept using the Lua programming language. It groups the editor's rich API into a series of tasks in a convenient and easy-to-use manner.

This book covers how to:

- Leverage Textadept's important files and folders
- Adeptly navigate and manipulate text
- Mark lines and text visually
- Show interactive lists and call tips
- Prompt for user input in various ways
- Spawn asynchronous, interactive child processes
- Configure colors, themes, and other settings
- Define lexers for highlighting source code
- And much more

Mitchell is the author and principal developer of Textadept and commands over 15 years of experience with Lua.

Triple Quasar Books

FIFTH EDITION

Textadept
Quick Reference

Mitchell

Textadept Quick Reference
by Mitchell

Contact the author at books@triplequasar.com.

Editor: Ana Balan
Cover Designer: Mitchell
Interior Designer: Mitchell
Indexer: Mitchell

Printing history:
December 2013: First Edition
May 2015: Second Edition
October 2016: Third Edition
August 2018: Fourth Edition
December 2020: Fifth Edition

ISBN: 978-0-9912379-6-8

Preface to the Fifth Edition

This book is an updated version of the fourth edition of *Textadept Quick Reference*. It includes many of the new features introduced in Textadept 11.0, and covers the backwards-incompatible changes made. In a nutshell, this book covers the following new or notable topics:

- The universal use of table indices that start at 1.
- The artificial separation of buffer and view API functions and fields.
- Key sequence modifier changes.
- Theme changes and enhancements.
- New lexer functions, fields, and style definition syntax.
- The ability to save and restore custom session data.
- Auto-highlight the current word (or currently selected word) and find results.
- The new progressbar dialog.
- New events for when text in the find entry and command entry changes.

For a comprehensive list of changes between Textadept versions, please refer to the editor's *docs/changelog.md* file, which is distributed with the application. The online version is located at *https://orbitalquark.github.io/textadept/changelog.html*.

Contents

Introduction

Textadept is a fast, minimalist, and remarkably extensible cross-platform text editor for programmers. Written in a combination of C and Lua[1] and relentlessly optimized for speed and minimalism for over twelve years, Textadept is an ideal editor for programmers who want endless extensibility without sacrificing speed and disk space, and without succumbing to code bloat and a superabundance of features.

Textadept runs in both graphical and text-based user interface environments. The text-based version of the editor is referred to as the "terminal version," since it executes within a terminal emulator.

Textadept Quick Reference is designed to help the user "get things done" when it comes to scripting and configuring Textadept. Its pragmatic approach assumes the user has a basic working knowledge of both Lua and Textadept. This book is broken up into a number of descriptive sections with conveniently grouped tasks that cover nearly every aspect of Textadept's Application Programming Interface (API). For the most part, the contents of each task are not listed in conceptual order. They are listed in procedural order, an order the user would likely follow when writing Lua scripts. This quick reference serves as a complement to Textadept's comprehensive Manual and extensive API documentation.

While this book aims to be a complete reference, it does omit some of the less useful features of Textadept's API. For example, although many of Textadept's table fields are both readable and writable, this reference sometimes chooses to cover only one of those operations. (Unless a field is marked "Read-only" or "Write-only", it is both readable and writable.) This book also does not cover Lua or its standard libraries. *Lua Quick Reference*,[2] by Mitchell (2020), is a good resource on that subject.

Finally, the topics covered in this book are designed to be used primarily in user-written Lua scripts and in the occasional "one-shot" Lua command. If the user keeps this in mind, he or she can realize Textadept's full potential.

1 *https://www.lua.org*
2 *https://orbitalquark.github.io/lua-quick-reference*

Download

Textadept binary packages for Windows, macOS, and Linux platforms are available from *https://orbitalquark.github.io/textadept*. Each package is self-contained and need not be installed. Textadept's source code is also included in each archive. The user may compile the application manually by following the instructions in the editor's Manual.

Conventions

This book uses the following typographical conventions.

Italic
: Used for filenames and for introducing new terms.

`Constant width`
: Used for environment variables, command line options, and Lua code, including functions, tables, and variables.

`Constant width italic`
: Used for user-specified parameters.

`[]`
: Used for optional function arguments, except in code examples that index Lua tables. Unless otherwise specified, optional arguments default to `nil`.

Terminology

This book uses the following terminology.

Buffer
: An object that contains editable text.

View
: An object that displays a single buffer.

Caret
: Either the visual that represents the text insertion point or the end point of a text selection.

Anchor
: The start point of a text selection or search.

Word
: A contiguous sequence of characters from a set of word characters. What constitutes a word character varies between programming languages and can be configured in Textadept.

Virtual Space
: The empty space past the ends of lines.

Lexer
: A Lua module that highlights the syntax of source code written in a particular programming language. Textadept refers to a programming language by its lexer's name.

Style
: A collection of display settings specific to source code comments, strings, keywords, and other ranges of text.

Language Module
: A Lua module automatically loaded by Textadept when editing source code in a particular programming language. The module's name matches the language's lexer name. Not all languages have language modules.

Environment Variables

Textadept utilizes the following environment variables.

`HOME` or `USERHOME`
: The user's home directory. Textadept's user data and preferences exist in a *.textadept/* sub-directory, denoted as *~/.textadept/* throughout this book.

  On Windows, this directory is typically *C:\Users\user name*. On macOS, it is */Users/username/*. On Linux and BSD it is often */home/username/*.

`LANG`
: The user's default locale. Textadept will display localized text and messages in it if possible.

Important Files and Directories

Textadept allows the user to configure and customize the editor using several important files and directories contained within his or her *~/.textadept/* directory.

~/.textadept/init.lua
: The file executed on startup that allows the user to configure Textadept and customize what the application does when it starts. For example, the user can set a color theme, specify default buffer and view settings, change the settings of existing modules, load custom modules, configure key bindings and snippets, extend menus, enhance support for file types and programming languages, and run arbitrary Lua code. Example 1 shows a sample *~/.textadept/init.lua* file.

Example 1. Sample ~/.textadept/init.lua

```
-- Adjust the default theme's font name and size.
if not CURSES then
  view:set_theme('light', {
    font = 'DejaVu Sans Mono', size = 12
  })
end

-- Always use 4 spaces for each level of indentation.
buffer.use_tabs, buffer.tab_width = false, 4

-- Disable code folding and hide the fold margin.
lexer.folding = false
view.margin_width_n[3] = 0

-- Wrap long lines into view and hide the horizontal
-- scroll bar.
view.wrap_mode = view.WRAP_WHITESPACE
view.h_scroll_bar = false

-- Disable character auto-pairing with typeover, strip
-- trailing whitespace on save, and auto-highlight all
-- instances of the current word.
textadept.editing.auto_pairs = nil
textadept.editing.typeover_chars = nil
textadept.editing.strip_trailing_spaces = true
textadept.editing.highlight_words =
  textadept.editing.HIGHLIGHT_CURRENT
```

```
-- Load a user module from ~/.textadept/modules/ and
-- bind a key to one of its functions.
local ctags = require('ctags')
keys.f12 = ctags.goto_tag

-- Remap the quit command from Ctrl+Q to Ctrl+Alt+Q.
keys['ctrl+alt+q'], keys['ctrl+q'] = quit, nil

-- Define some global snippets.
snippets.date = '%<os.date()>'
snippets.ta = '/home/mitchell/code/textadept/'
snippets.accessor = "\z
\tfunction %1(name)(self) return self._%1 end\
\tfunction set_%1(self, %2(value))\
\t\tself._%1 = %2\
\tend\
"

-- Add menu option for resetting Textadept's Lua state.
local tools = textadept.menu.menubar[_L['Tools']]
table.insert(tools, {''}) -- menu separator
table.insert(tools, {'Reset L_ua State', reset})

-- Recognize .luadoc files as Lua code, change .html
-- files to be recognized as XML files, and recognize
-- a shebang like "#!/usr/bin/zsh" as shell code.
textadept.file_types.extensions.luadoc = 'lua'
textadept.file_types.extensions.html = 'xml'
textadept.file_types.patterns['^#!.+/zsh'] = 'bash'

-- Change the run commands for Lua and Python.
textadept.run.run_commands.lua = 'lua5.1 "%f"'
textadept.run.run_commands.python = 'python3 "%f"'

-- In the terminal version, disable suspend and make
-- Ctrl+Z perform undo.
if CURSES then
  events.connect(events.SUSPEND, function()
    buffer:undo()
    return true -- do not propagate
  end, 1)
end

-- Add an additional extension to ignore in all file
-- directory filters.
table.insert(lfs.default_filter, '!.ext')
```

```
-- Change the color of Java functions from orange to
-- black, ensure 4-space indentation for Python, and
-- load an extra module for the Lua language.
events.connect(events.LEXER_LOADED, function(name)
  if name == 'java' then
    local black = view.style_fore[view.STYLE_DEFAULT]
    local style_fun = buffer:style_of_name('function')
    view.style_fore[style_fun] = black
  elseif name == 'python' then
    buffer.use_tabs = false
    buffer.tab_width = 4
  elseif name == 'lua' then
    require('lua.repl')
  end
end)
```

~/.textadept/locale.conf
: Defines Textadept's localized messages. The user may override or manually set Textadept's locale by copying a locale file from the editor's *core/locales/* directory to *~/.textadept/locale.conf*.

~/.textadept/modules/
: Contains user modules. When Textadept looks for modules to load via Lua's `require()` function, it looks in this directory first. The user can override one of Textadept's own modules by creating a new module of the same name in *~/.textadept/modules/*. For example, the user may create a *~/.textadept/modules/textadept/keys.lua* file with a completely different set of default key bindings. Textadept will load that file on startup instead of its own.

~/.textadept/themes/
: Contains user themes. When Textadept looks for a color theme to load, it looks in this directory first. The user can override one of Textadept's own themes by creating a new theme of the same name in *~/.textadept/themes/*. Example 2 shows a sample *~/.textadept/themes/light.lua* file. The section "Create or Modify a Color Theme" on page 99 describes themes in more detail.

Example 2. Sample ~/.textadept/themes/light.lua

```
-- Load the default light theme.
dofile(_HOME .. '/themes/light.lua')
```

```
-- Change the default theme's keywords to be bold.
local colors, styles = lexer.colors, lexer.styles
styles.keyword = {fore = colors.dark_blue, bold = true}
```

~/.textadept/lexers/

Contains user lexers. When Textadept looks for lexers to highlight source code with, it looks in this directory first. The user can override one of Textadept's own lexers by creating a new lexer of the same name in *~/.textadept/lexers/*. Example 3 shows a simple lexer for a C-like language. The section "Define a Lexer" on page 125 describes lexers in more detail.

Example 3. Sample ~/.textadept/lexers/c_like.lua

```
-- Basic definitions.
local lexer = lexer
local token, word_match = lexer.token, lexer.word_match
local P, S = lpeg.P, lpeg.S

-- Create the lexer object.
local lex = lexer.new('c_like')

-- Whitespace.
local ws = token(lexer.WHITESPACE, lexer.space^1)
lex:add_rule('whitespace', ws)

-- Keywords.
local keyword = token(lexer.KEYWORD, word_match[[
  break continue do else for if return while
]])
lex:add_rule('keyword', keyword)

-- Types.
local type = token(lexer.TYPE, word_match[[
  bool char double float int long struct void
]])
lex:add_rule('type', type)

-- Null.
local null = token('null', P('NULL'))
lex:add_rule('null', null)
lex:add_style('null', lexer.STYLE_CONSTANT)

-- Identifiers.
local identifier = token(lexer.IDENTIFIER, lexer.word)
lex:add_rule('identifier', identifier)
```

```
-- Strings.
local sq_str = lexer.range("'", true)
local dq_str = lexer.range('"', true)
local string = token(lexer.STRING, sq_str + dq_str)
lex:add_rule('string', string)

-- Comments.
local line_comment = lexer.to_eol('//')
local block_comment = lexer.range('/*', '*/')
local comment = line_comment + block_comment
lex:add_rule('comment', token(lexer.COMMENT, comment))

-- Numbers.
local number = token(lexer.NUMBER, lexer.number)
lex:add_rule('number', number)

-- Preprocessor.
local pp = lexer.starts_line(lexer.to_eol('#', true))
lex:add_rule('preproc', token(lexer.PREPROCESSOR, pp)

-- Operators.
local op = S('+-/*^<>=;.,()[]{}')
lex:add_rule('operator', token(lexer.OPERATOR, op))

-- Specify how the lexer folds code.
lex:add_fold_point(lexer.OPERATOR, '{', '}')
lex:add_fold_point(lexer.COMMENT, '/*', '*/')
lex:add_fold_point(
  lexer.COMMENT, lexer.fold_consecutive_lines('//'))

return lex
```

Command Line Options

Textadept processes command line options sequentially, so order matters. The application accepts the following command line options.

filename
: Opens file *filename* for editing.

dirname
: Sets Textadept's current working directory to *dirname*, which would typically be a project directory. Any subsequent relative filenames are considered relative to

`buffer.annotation_lines[line]` (Read-only)
: The number of annotation text lines for line number *line*.

Mark Text with Indicators

Textadept supplies 32 *indicators* to mark text with. Each indicator has an assigned indicator style from the list in Table 6. The editor displays indicators along with any existing styles text may have. Indicators move along with text. Example 11 shows how to create and interact with clickable hyperlinks.

Table 6. Indicator styles

Indicator Style	Visual or Description
`view.INDIC_SQUIGGLEPIXMAP`	A squiggly underline.
`view.INDIC_PLAIN`	An underline.
`view.INDIC_DASH`	A dashed underline.
`view.INDIC_DOTS`	A dotted underline.
`view.INDIC_COMPOSITIONTHICK`	A thick underline.
`view.INDIC_STRIKEOUT`	~~A strikeout line.~~
`view.INDIC_BOX`	A bounding box.
`view.INDIC_DOTBOX`	A dotted bounding box.
`view.INDIC_STRAIGHTBOX`	A translucent box.
`view.INDIC_ROUNDBOX`	A translucent box with rounded corners.
`view.INDIC_FULLBOX`	A translucent box that extends to the top of its line.
`view.INDIC_GRADIENT`	A box with a vertical gradient from solid to transparent.
`view.INDIC_GRADIENTCENTER`	A box with a centered gradient from solid to transparent.
`view.INDIC_TT`	An underline of small 'T' shapes.
`view.INDIC_DIAGONAL`	An underline of diagonal hatches.
`view.INDIC_POINT`	A triangle below the start of text.

Indicator Style	Visual or Description
view.INDIC_POINTCHARACTER	A triangle under the first character.
view.INDIC_TEXTFORE	Changes text's foreground color.
view.INDIC_HIDDEN	Plain text with no decorations.

Example 11. Create and interact with hyperlinks

```
-- Define hyperlink indicator.
local INDIC_LINK = _SCINTILLA.next_indic_number()
events.connect(events.VIEW_NEW, function()
  view.indic_hover_style[INDIC_LINK] =
    view.INDIC_TEXTFORE
  view.indic_hover_fore[INDIC_LINK] = 0xFF0000 -- blue
end)

-- Search the buffer and mark hyperlinks.
function mark_hyperlinks()
  local text = buffer:get_text()
  buffer.indicator_current = INDIC_LINK
  buffer:indicator_clear_range(1, buffer.length)
  for s, e in text:gmatch('()https?://%S+()') do
    buffer:indicator_fill_range(s, e - s)
  end
end

-- Open clicked hyperlinks in a web browser.
local browser_cmd = 'firefox "%s"'
events.connect(events.INDICATOR_CLICK, function(pos)
  local indicators = buffer:indicator_all_on_for(pos)
  if indicators & INDIC_LINK > 0 then
    local s = buffer:indicator_start(INDIC_LINK, pos)
    local e = buffer:indicator_end(INDIC_LINK, pos)
    local url = buffer:text_range(s, e + 1)
    os.spawn(browser_cmd:format(url))
  end
end)
```

`_SCINTILLA.next_indic_number()`
: Returns a unique indicator number, up to 32.

`view.indic_style[`*`indicator`*`] = `*`style`*
: Assigns indicator style *`style`* to indicator number *`indicator`*. Table 6 lists all available indicator styles. The section "Assign Indicator Colors" on page 104 describes how to change the color and alpha values of *`indicator`*.

This assignment also resets `view.indic_hover_style[`*`indicator`*`]` to *`style`*.

The terminal version requires *`style`* to be `buffer.INDIC_STRAIGHTBOX`, but cannot draw it translucently.

TIP

The user should either assign indicator styles in his or her *~/.textadept/init.lua* file or within an `events.VIEW_NEW` handler, so subsequent views can recognize them.

`view.indic_under[`*`indicator`*`] = `*`bool`*

Draw indicator number *`indicator`* behind text instead of in front of it. The default value is `false`.

`view.indic_hover_style[`*`indicator`*`] = `*`style`*

Assigns indicator style *`style`* to indicator number *`indicator`* when either the mouse is hovering over that indicator or when the caret is within that indicator.

`buffer.indicator_current = `*`indicator`*

Designates indicator number *`indicator`* as the indicator used by `buffer:indicator_fill_range()` and `buffer:indicator_clear_range()`.

`buffer:indicator_fill_range(`*`pos`*`, `*`length`*`)`
`buffer:indicator_clear_range(`*`pos`*`, `*`length`*`)`

Fills or clears the range of text from position *`pos`* to *`pos`* + *`length`* with indicator number `buffer.indicator_current`.

Highlight Words

Textadept can automatically highlight all instances of the current word or currently selected word using indicators. The user's *~/.textadept/init.lua* file may configure this setting.

`textadept.editing.highlight_words = `*`mode`*

Specifies highlight mode *`mode`* as the mode for automatic word highlighting. The default value is `textadept.editing.HIGHLIGHT_NONE`. `textadept.editing.HIGHLIGHT_SELECTED` automatically highlights all instances of the currently selected word, and `textadept.editing.HIGHLIGHT_CURRENT` highlights all instances of the word under the caret.

Query Indicator Information

The user can fetch indicator locations and retrieve the indicators present at particular positions.

`buffer:indicator_start(`*`indicator`*`, `*`pos`*`)`
`buffer:indicator_end(`*`indicator`*`, `*`pos`*`)`
: Returns the previous or next boundary position, starting from position *`pos`*, of indicator number *`indicator`*. Returns `1` in both instances if *`indicator`* was not found.

`buffer:indicator_all_on_for(`*`pos`*`)`
: Returns a bit-mask that represents the indicators present at position *`pos`*. The mask is a 32-bit value whose bits correspond to Textadept's 32 indicators.

Handle Indicator Events

Textadept emits the following indicator click events that the user can connect to.

`events.emit(events.INDICATOR_CLICK, `*`position`*`, `*`modifiers`*`)`
: Emitted when clicking the mouse on text that has an indicator present. *`position`* is the clicked text's position and *`modifiers`* is a bit-mask of any modifier keys held down (`view.MOD_CTRL` for Ctrl/Command, `view.MOD_SHIFT` for Shift, `view.MOD_ALT` for Alt, and `view.MOD_META` for Ctrl on macOS).

`events.emit(events.INDICATOR_RELEASE, `*`position`*`)`
: Emitted when releasing the mouse after clicking on text that has an indicator present. *`position`* is the clicked text's position.

Show an Interactive List

Textadept has the ability to display two types of interactive lists that update as the user types: an *autocompletion list* and a *user list*. An autocompletion list is a list of completions shown for the current word. A user list is a more general list of options presented to the user. Both types of lists have similar behavior and may display images alongside text. All of the above is described in the following sections.

Concept Index

D

E

F

T

U

V

W

X

Z