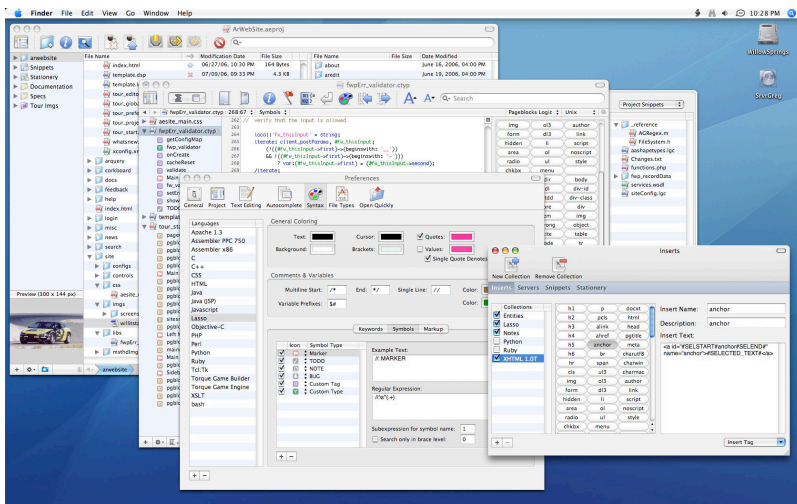




Version 1.0 User Guide

for Public Preview 5



Araelium Group



Araelium Edit User Guide is Copyright 2006-2007, Araelium Group. All rights reserved.

Araelium Edit is Copyright 2006-2007, Araelium Group. All rights reserved.

Araelium™ and Araelium Edit™ are trademarks of Araelium Group

There's probably trademarks in here that are trademarks of Apple Computer, Inc.

Araelium Group • 960 N. Tustin St. #247 • Orange, CA 92867 • www.araelium.com

General Contact: info@araelium.com

Product Assistance: areditbeta@araelium.com

Contents

Getting Started	6
This is a Beta Release	6
Contact Us	6
Installation	6
Be Aware That...	6
The Jump Start Tour	7
Document Window	7
Document Editor	7
Document Browser	8
View Splitters	8
Inserts	8
Snippets	8
Document and Symbols Selectors	8
Companion File Switcher	9
Project Window	9
Project Files and Groups	9
Project File Browser	10
Image Preview	10
File Transfer	10
Global Resource Browser	10
Servers, Stationary, Snippets, and Inserts	10
Preferences	11
General Preferences	11
Text Editing Preferences	11
Code Completion	11
Syntax Preferences	11
File Type Mapping Preferences	11
Open by Path Preferences	11
Help and Language References	12
Application Documentation	12
Language Reference Materials	12

Working with Documents	13
Creating and Opening Documents	13
Document Window Work Spaces	14
The Document Window File Shelf	15
The Document Window Navigation Bar	16
Editor Split Views	18
Document Window Inserts Panel	18
Document Window Snippets Drawer	19
Text Transformations	20
Sort Lines	20
Hard Wrap Lines	21
Prefix / Postfix Lines	21
Remove Duplicate Lines	21
Add/Remove Line Numbers	22
Working with Inserts and Snippets	23
Interface Differences Between Inserts and Snippets	23
Inserts	23
Snippets	24
Working with Inserts	25
Working with Snippets	27
Substitutions for Inserts, Snippets, and Stationery	28
Substitution Keywords	28
Searching with Find and Find All	29
Using Find	29
Search From Toolbar	30
Using Find All	30
Find All and Selective Replace	31
Using Regular Expressions	32
Syntax elements	32
Characters	32
Character Types	33
Quantifier	33

Character class	34
Extended groups	35
Back Reference	36
Subexp Call (“Tanaka Akira special”)	36
Captured Group	37
Extensions Original to Oniguruma	37
PERL 5.8.0 Features Not Present	38
Specifying Subexpressions in Araelium Edit	38
Customizing Syntax Definitions	39
General Coloring	40
Comments and Variables	40
Keywords	41
Symbols	41
Multiple Language Coloring	43
Working with Projects	44
Project Window Panels	44
Real, Virtual, and Smart Folders	45
Starting a New Project	45
Using the New Project Assistant	46
Creating a New Source Code Folder	46
Adding Source Code files to the Project	47
Project Tips	47
Use the Project Shelf to Organize Reference Materials	48
File Transfers in Projects	49
Command Line Tool aredit	50
Installation	50
Using aredit	50
Using aredit with svn	50
Licenses	51
END USER AGREEMENT	51
Additional Licenses	53

Getting Started

This is a Beta Release

Please be aware and keep in mind that this release of Araelium Edit is a beta. There are known bugs, incomplete features, and some awkward user interface areas. Please read the release notes from the application Help menu for recent fixes, changes, and current known issues.

Contact Us

We have an active list of fixes and enhancements to make, but we encourage you to send us all your comments about things that smell like bugs, things you find awkward, things you like, and things you'd like to see added.

Please join the talk list: <http://www.araelium.com/talk/>

Send Beta Feedback to: areditbeta@araelium.com

Installation

To install Araelium Edit, copy the application from the downloaded disk archive to the the Applications folder of your hard drive.

Araelium Edit will create a folder in ~/Library/Application Support. This is where user defined application-wide Inserts, application-wide Snippets, Stationary, and Servers will be located. Any additions or changes you make to Syntax and File Types preferences will also be stored here.

Be Aware That...

Araelium Edit is designed to be a development environment for multiple languages, and there can be conflicts if multiple languages try to claim a file extension or a keyboard shortcut for the Araelium Edit inserts feature.

By default when Araelium Edit is first installed, the inserts feature enables all languages at once. This can, and likely will cause conflicts where inserts from different languages may use the same keyboard command. The first thing you should do when getting started is open Window ▸ Inserts and uncheck languages and insert sets you don't plan to use. In a future release we'll have a more intelligent way to deal with this.

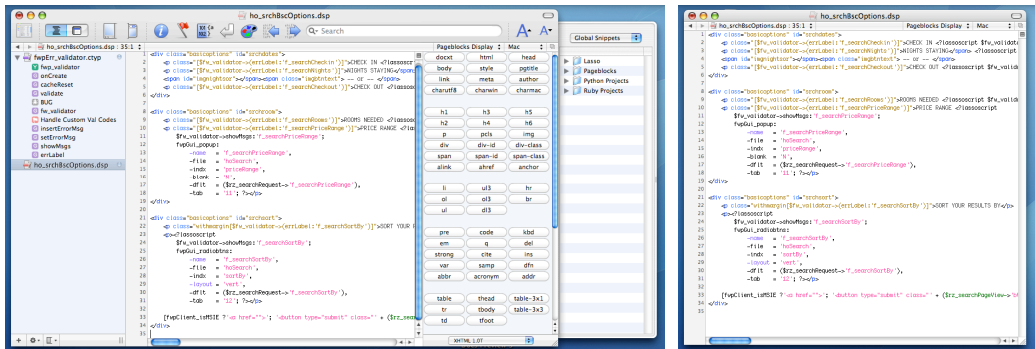
The Jump Start Tour

This section provides an overview of the application's major windows, controls, and features. It also identifies terminology which may be unique to Araelium Edit, or may be the first time you have encountered it.

Document Window

From minimalist to instant access to multiple tools, the document window can conform to your preferred style.

Document Windows in Contrasting Layouts



Document Editor

- multiple documents per window
- syntax coloring for up to four languages per document
- block indent control¹
- parenthesis balancing and highlighting
- language keyword and custom keyword autocomplete²
- lots more of course

Document Browser

- show or hide a list of open documents
- reveal symbols³ for any open document (symbols are the names of functions, classes, methods, HTML divs with id attributes, arbitrary markers, and pretty much any custom string you want to catch for display in the symbols list)
- click a symbol to jump to that point in any document whether it is the front document or not
- view either the open documents shelf or the project shelf

View Splitters

- split files horizontally
- split files vertically and use synchronized scrolling
- multiple splits allowed
- each split can have a different document

Inserts

- insert text with insertion, selection, and substitution keywords
- 3 methods: click button, drag and drop button, or type the button title followed by your preferred hot key such as ^ or \ or tab
- option-click a button to edit it
- tooltips provides description aids
- create and edit custom insert sets selectable from popup menu

Snippets

- insert text files of any kind such as prewritten code, reference material, templates, etc.
- uses substitution keywords compatible with BareBones® BBEdit™ glossary placeholders
- select from global snippets or project-specific snippets
- option-click a file to edit it rather than insert it

Document and Symbols Selectors

- document popup lists documents opened in the current window
- sub-menus for each document lists the symbols in that document

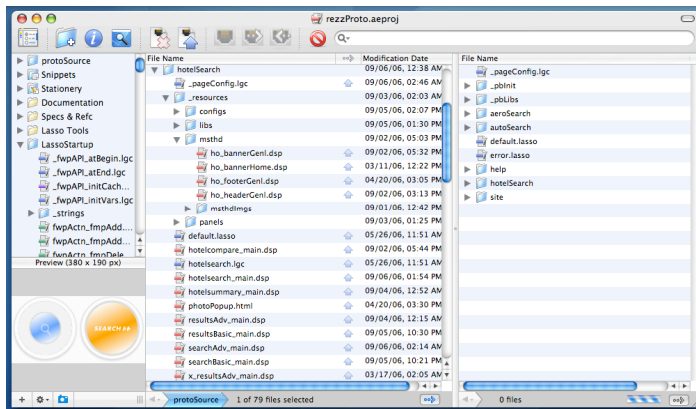
- the symbols selector popup shows all symbols of the currently viewed document in their order of appearance in the document
- the names of symbols within any given document are automatically added to the autocomplete list

Companion File Switcher

- open myfile.x and have Araeliu Edit automatically find and open myfile.y for you (automatically works with any extension)

Project Window

Project Window Showing the Shelf, Project Files, and Server Files



Project Files and Groups

- use real folders (blue) to organize files as actual disk file organization
- use virtual folders (yellow) to organize files from any hard drive location into arbitrary collections
- view files by smart folders
- view project snippets
- view project stationary

Project File Browser

- straightforward file list
- mark files not to be uploaded during server syncing
- quickly identify missing files

Image Preview

- show/hide image previewer
- use File Types preferences to set files which display in the preview panel

File Transfer

- SFTP and FTP
- remote server file browser
- multiple upload management choices (selection, sync, etc)
- multiple simultaneous upload sessions
- project window and source files still accessible during uploads
- uploading virtual folders creates matching real folders on server

Global Resource Browser

Servers, Stationary, Snippets, and Inserts

- define multiple servers global to the application (projects can select from any global server at any time)
- organize global stationary
- organize global snippets
- create and edit Insert sets

Preferences

General Preferences

- customize general window behavior and other application-wide settings

Text Editing Preferences

- font, tab control, line endings, encodings, and others

Code Completion

- set time delay for autocomplete popup
- set Inserts hot key
- switch on/off auto complete of () pairs

Syntax Preferences

- multiple syntax coloring options
- keyword lists
- uses regular expressions to identify symbols along with user selected icon
- define custom “markers” to create arbitrary identifiers amongst symbols

File Type Mapping Preferences

- define up to four language settings for each file type
- define your preferred editor for a given file type, allows you to use Araelium Edit's project manager as the launch pad for any file and application within the project

Open by Path Preferences

- define paths for Araelium Edit to automatically search as you key in path names to open files

Help and Language References

Application Documentation

- PDF documentation (this document) available from the Help menu
- Apple Helpbook available from the Help Menu

Language Reference Materials

- The Help menu will automatically add content from the ~/Library/Application Support/Help folder. Any folder in the Help folder will add a hierarchical menu item to the Help menu. All files will be added as menu items.
- Help menu includes links to a number of language references on the web
- Add your own .webloc and .pdf files, or any other file types to the Help folder as real files or as aliases.

¹ Block indenting is the ability to select multiple lines of text and have them indented less or more in a single step together.

² Autocomplete is the behavior in which the editor attempts to complete the typing of a keyword for you based on what you have typed so far. If the application determines there is only a single keyword that applies to what has been typed, autocomplete will type ahead of the user. If multiple keywords are possible based on what has been typed so far, a small popup selector will list possible words. The user can continue typing, or select a word from the list.

³ A symbol is a generic term used to encompass a language structure such as a function, procedure, class, method, comment, etc. Language editors like Araelium Edit will try to identify these structures in a source code file to provide the developer an outline of the file, as well as some functional "links" to use to jump to various location in the file. In Araelium Edit the listing of these structures can include unique icons, or symbols, to help identify the structure type as well as its name. Araelium Edit refers to the list of these names and icons as symbols.

Working with Documents

Araelium Edit supports multiple text documents per window, and includes several language tools for making code editing more efficient. This section explains the options for working with documents, and the language tools within the document window.

Creating and Opening Documents

Araelium Edit supports multiple text documents per window. Therefore, there are several File menu items that allow the creating and opening of a document within a new window, or the current front window.

New in Window will add a blank document to the current front window.

New File will create a blank document in a new window.

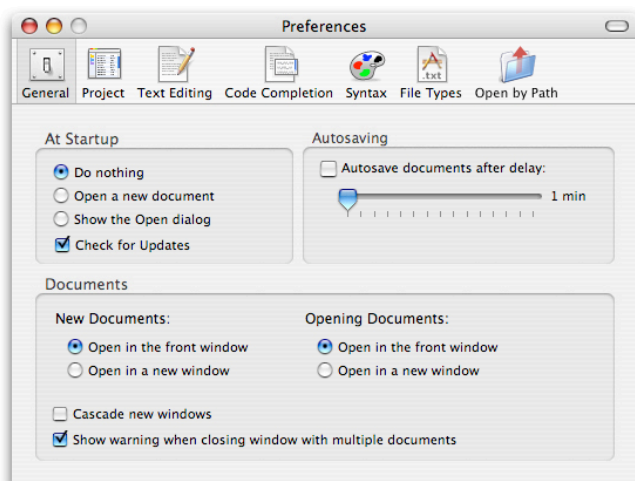
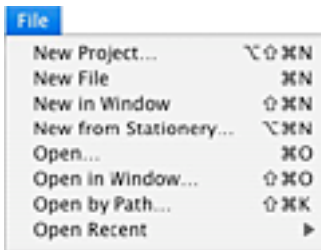
New from Stationery will create a new document populated with the text of a pre-defined stationery document selected from a list presented in a dialog.

Open in Window opens an existing document into the front window.

Open File opens an existing document into a new window.

Open by Path accepts a typed file name to be opened if the file is found in the project or in any of the paths entered in the Open by Path preferences.

Menu Items and Preferences for Documents

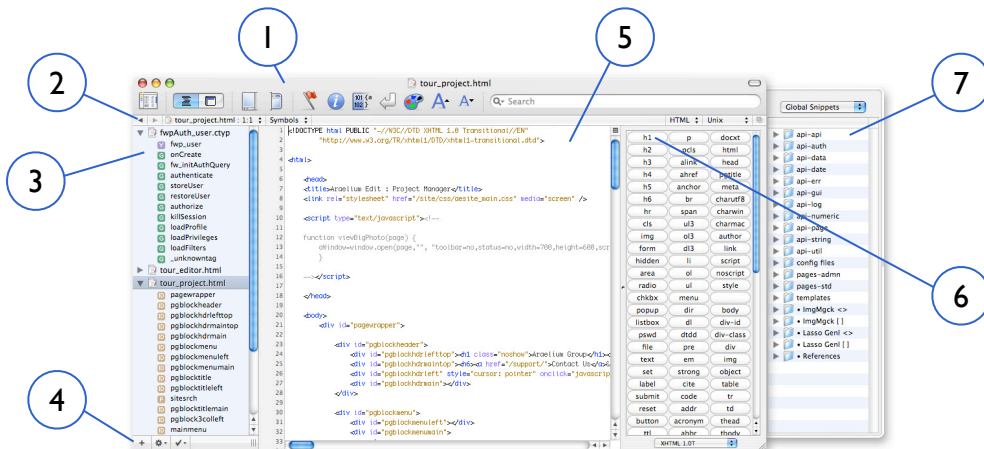


In the General preferences, document creation options at application startup can be defined, as well as the default behavior of whether new documents and opened documents (from the project, or from Finder) are added to the front window or a new window.

Document Window Work Spaces

The document window includes several tool spaces around the editor. Each of these can be displayed or hidden as needed or desired. The major tool spaces are the standard OS X document toolbar, the file shelf on left side of window

Document window editor and tool spaces



- ① Toolbar—use View ▸ Customize Toolbar... to change the currently visible toolbar controls.
- ② Navigation Bar—contains several controls for selecting files, positions within files, and changing file configuration.
- ③ File Shelf—shows currently opened files and their symbols, or the project file shelf if the file is from a project.
- ④ File Shelf Controls—create new documents, and configure what is shown in the file shelf through these controls.
- ⑤ Document Editor—the main text editing field for the document.
- ⑥ Inserts Browser—view and select inserts from this browser, or edit an insert by option-clicking an insert button.
- ⑦ Snippets Browser—view, select, and open either global snippets or project-specific snippets from this list.

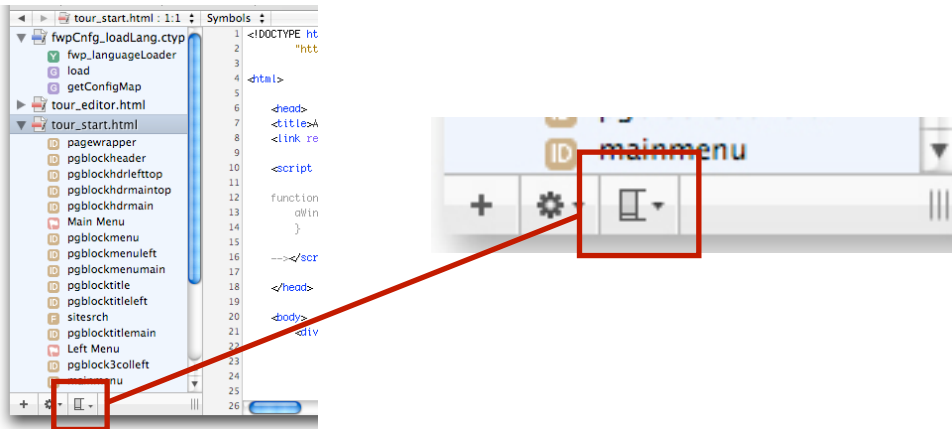
(command-1), the Inserts panel on the right side (command-2), and the Snippets drawer (command-3).

The Document Window File Shelf

The file shelf at the left of the document window is used to display either the contents of the project window file shelf (if the front document was opened from a project window), the documents currently open in the window, or the symbols list of the current front document.

When the open documents browser is used, each file may have a reveal button to display extracted symbols from that document. Symbols are listed in their order of appearance in the document.

File ShelfView Selector



File Shelf Usage Tips

General

- Use command-1 to show/hide the file shelf.
- To switch views in the file shelf, select from the shelf view selector at the bottom of the window.

Open-File Browser

- Click a file name to display that document in the editor.
- Click a file's reveal button to expose the extracted symbols from that document.
- Click a symbol of any file to jump to that position within that file.

Project-File Browser

- Double-click a text file from the project file list to open that document in the current window.

Symbols Browser

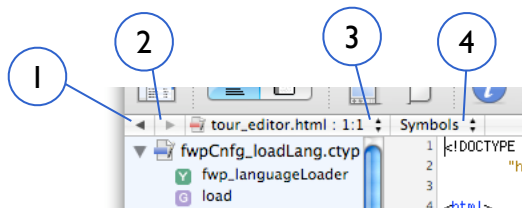
- Click a symbol to jump to that position within the file.

The Document Window Navigation Bar

Just below the window toolbar (or the title bar if the toolbar is not displayed), is the document window navigation bar. The left side of the navigation bar has controls for selecting files and jumping to a specific location within the front file.

The file selector list includes sub-menus to display the symbols of each file. The first sub-menu shows the unique symbol types found in the file. Each of the

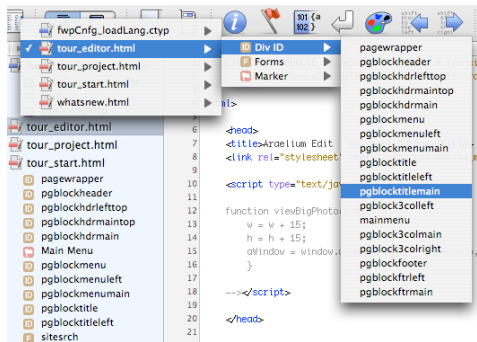
Document Navigation Bar (Left)



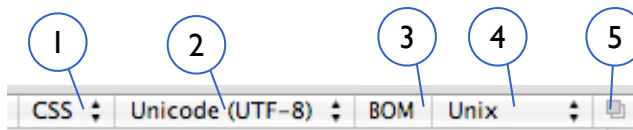
- ① Click the ◀ button to cycle backwards through the history of files you have viewed (that are currently opened).
 - ② Click the ▶ button to cycle forwards through the history of files you have viewed (that are currently opened).
 - ③ The third item is the open file selector used to switch among the currently open files.
 - ④ The Symbols popup shows the symbols extracted from the current front document. If the window is split and has different documents in each split, the Symbols list will change based on the view that has the active cursor.
-

symbol sub-menus contains a list of each instance of that type of symbol in the document. This specific organization was created so that while working in one document, it would be easy to find a specific symbol in another document without have to switch documents, and without having to alter the display of the file shelf. Selecting a specific item from the symbol instances list will bring that document to the front and jump to that symbol location.

Navigation Bar File Selector with Symbol Sub-menus



Document Navigation Bar (Right)



- ① Language Set menu. Identifies which language set to apply to the document.
- ② Encodings menu. Sets character encoding for the document.
- ③ Byte Order Mark switch. Adds/removes BOM from the document.
- ④ Line Endings menu. Sets/displays line endings for the document
- ⑤ Companion switch. Switches between documents of the same name, but different extensions.

The right side of the navigation bar includes selectors for setting the configuration of the file including the language set, encoding, and line endings.

The Language Set menu which file type set to use for syntax coloring. This list is not necessarily a list of languages. Araelium Edit can colorize text with up

to four individual language syntax rules per file. So, selecting just one language is not sufficient. Groups of languages which are combined in a single file are based on named file sets defined in the File Types preferences. This popup menu is a list of those sets.

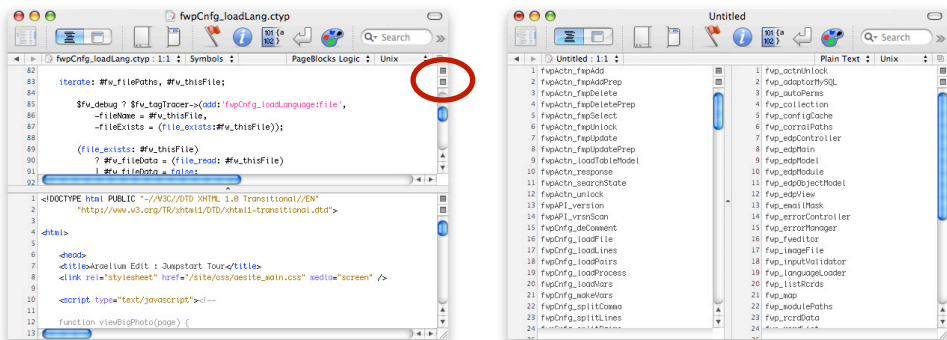
The second menu sets the character encoding for the file. If the encoding is one of the Unicode options, then a BOM switch will appear next to the encodings menu. This is a simple button which is inactive (dimmed) if there no BOM, and active if there is a BOM. Click the button to change the setting.

The last popup in the navigation bar sets the line endings for the document. When pasting text, the line endings of the text being pasted will be converted to the setting indicated for the document.

The last little button in the navigation bar is the file companion switch. When a document is opened, Araeliu Edit will search the same disk directory the current file is from to locate files of the same name but with different extensions. If any are found the companion switch will be active. Click the switch will cycle through each file, automatically opening it in the editor.

Editor Split Views

The document editor field can be split horizontally or vertically. In either case, multiple splits is allowed. Each split can display a separate document. Synchronized scrolling in either horizontal or vertical splits can be enabled with the View ▸ Synchronized Scrolling menu item. Creating and closing splits is done with the buttons at the top of the vertical scroll bar.



Document Window Inserts Panel

The inserts browser panel (shown as ⑥ in the diagram on page 14) includes clickable and draggable buttons to insert text into the document. A menu at the bottom of the panel allows selection from multiple inserts collections.

To place an insert into the currently viewed document:

- type the insert name and hot key (user adjustable in Preferences)
- place the cursor where the insert is to be placed, and click the insert button
- click and drag and insert button to a location to place the text
- NOTE: the insert does not have to be from the currently displayed collection. Collection displays are for convenience, but do not have any impact on which inserts can be used at any given time.

Using and editing the inserts is discussed in more detail in the chapter *Working with Inserts and Snippets*.

Document Window Snippets Drawer

The snippets browser drawer (shown as ⑦ in the diagram on page 14) includes files which can be double-clicked or dragged into the document. The popup menu at the top of the drawer enables selection of global snippets (available to all projects and files) and project snippets (available only to the current project).

To place a snippet into the currently viewed document:

- move the text editing cursor to the location where the snippet is to be inserted
- reveal the snippets drawer (command-3), and navigate through the file list to the snippet to be inserted
- double-click a snippet file
- or, drag a snippet file into the position where the contents should be inserted

To edit a snippet file:

- hold the option key down and double-click the file

Using and editing the snippets is discussed in more detail in the chapter *Working with Inserts and Snippets*.

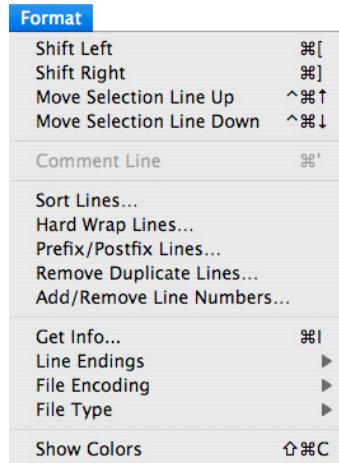
Text Transformations

The Format menu contains a number of tools for common text transformations in the current front document of the document window.

Sort Lines

The Sort Lines transformation will sort all lines, or the selected lines, of a document into either ascending or descending alphabetical order. Character sorting is based on the Mac OS X International Preference Pane Language tab setting for lists.

The default setting is to use the entire line for sorting, with an option to write a regular expression which will select only a portion of a line to use for sorting.

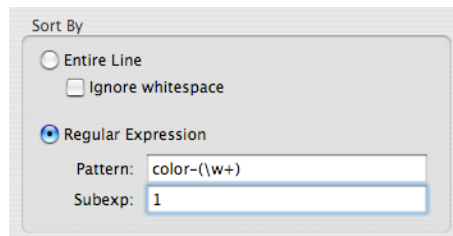


Using Regular Expressions for Sorting

Let's assume we have a list of text like the following in which we want the lines sorted based on the color value.

```
shape-triangle, color-black, size-small
shape-circle, color-yellow, size-small
shape-square, color-red, size-small
shape-trapezoid, color-blue, size-small
```

We can use the regular expression option to accomplish this by entering an expression in the Sort Lines sheet. The subexpression field accepts an integer value that identifies which () subexpression to actually use for the sorting comparison. The example expression would result in the list being sorted as follows:



```
shape-triangle, color-black, size-small
shape-trapezoid, color-blue, size-small
shape-square, color-red, size-small
shape-circle, color-yellow, size-small
```

Hard Wrap Lines

The Hard Wrap Lines transformation will take a body of text and inject line breaks, according to the document line endings setting, at fixed intervals of line lengths.

Prefix / Postfix Lines

The Prefix/Postfix Lines transformation can add or remove a specified string to the beginning and/or ending of each line of the document or the selected lines of a document.

Remove Duplicate Lines

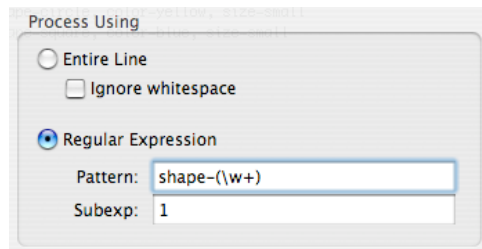
The Process Duplicate Lines transformation will remove duplicate lines from a document or the selected lines of a document. The default setting is to use the entire line for comparison to determine duplication, with an option to write a regular expression which will select only a portion of a line to use for determining whether a line is a duplicate.

Using Regular Expressions for Removing Duplicates

Let's assume we have a list of text like the following in which we want lines to be considered duplicates where the shape value is repeated. In these lines, we want the second occurrence of square to be removed.

```
shape-triangle, color-black, size-small  
shape-circle, color-yellow, size-small  
shape-square, color-red, size-small  
shape-trapezoid, color-blue, size-small  
shape-square, color-blue, size-small
```

We can use the regular expression option to accomplish this by entering an expression in the Remove Duplicate Lines sheet. The subexpression field accepts an integer value that identifies which () subexpression to actually use for the sorting comparison. The example expression would result in the list being sorted as follows:



The screenshot shows the 'Process Using' dialog box for the Remove Duplicate Lines transformation. It has two main sections: 'Entire Line' and 'Regular Expression'. The 'Regular Expression' section is selected with a radio button. Below it, there are two input fields: 'Pattern:' and 'Subexp:'. The 'Pattern:' field contains the text 'shape-(\w+)' and the 'Subexp:' field contains the number '1'. There is also an 'Ignore whitespace' checkbox which is unchecked.

```
shape-triangle, color-black, size-small  
shape-circle, color-yellow, size-small  
shape-square, color-red, size-small  
shape-trapezoid, color-blue, size-small
```

Add/Remove Line Numbers

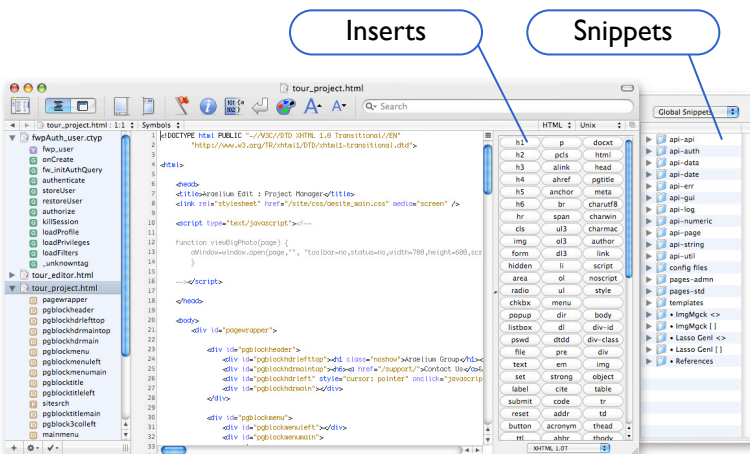
The Add/Remove Line Numbers transformation will insert a prefix to each line of the document, or selected lines of a document, with each line prefix automatically incremented. Set the Increment By field to a value that each line number will increase by.

With the Remove mode selected, the transformation will look for what appears to be a line number at the beginning of a line and remove it.

Working with Inserts and Snippets

Araelium Edit includes two mechanisms for inserting text into a document to accelerate code creation. Ultimately they're similar in capability, but vary in user interface, and thus each lends itself to particular uses or user preference.

Document window with inserts and snippets



Interface Differences Between Inserts and Snippets

Inserts

Inserts are generally for language commands and short language constructs that are used over and over even within a single file. For HTML, this might be something like inserting the text `<p class=""></p>` and having the cursor placed between the quotes ready for you to type. In Araelium Edit that can be done simply by typing something like `pcls\` where the `\` is a hot key used to identify the preceding string as an insert. The hot key is user definable, and of course, the insert name and text is user definable as well. Inserts are case sensitive so something like `p\` could insert a basic `p` tag pair, and `P\` could insert the tag pair with a class attribute. Typing the insert name and hot key is an interface feature that is unique to inserts. Snippets cannot be inserted this way.

Inserts are presented in a grid of buttons. Each button is labeled with the text that is typed in conjunction with the hot key to insert the expanded text. Inserts are defined in collections where each collection contains 150 buttons. Each grid, as seen in the window pictured above, is a single collection. A popup menu at the bottom of the button grid is used to select a collection for display.

Collections may be activated or deactivated using the inserts editor window. If a collection is deactivated, it will not appear in the collection popup list, and inserts which are part of the deactivated collection cannot be used. This is how the same insert name might be used for different languages without causing conflicts. For example `def\` can be used to insert a new Python function or a new Ruby class method. The Ruby and Python collections would be alternately activated/deactivated depending on which language is in use. Insert collections are application-wide, there are no project-specific inserts.

Snippets

Snippets are generally for much larger text constructs. Where inserts are saved to plist files that don't really lend themselves to being edited directly, snippets are individual standard text files. The interface for snippets is essentially a file list. Double clicking a file inserts the content of that file into the document. With the presentation of snippets being a file list, the visibility of the whole file name inherently offers one difference over the "shorthand" labeling of inserts.

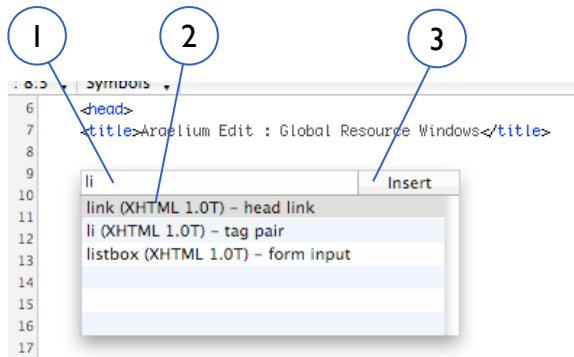
Snippets are presented in a document drawer. Snippets can be stored for application wide (a.k.a global) access, or stored to the project snippets folder to be project-specific. A selector at the top of the drawer allows the display of either.

Working with Inserts

To place an insert into the currently viewed document:

- type the insert name and hot key (Tab by default)
- place the cursor where the insert is to be placed, and click the insert button
- click and drag and insert button to a location to place the text
- place the cursor where the insert is to be placed, and press shift-Tab. A small window appears with a list of all inserts. As you type, the list of inserts is reduced to possible options much like autocomplete. This method is useful for adjoining inserts, for using an insert from a currently inactive insert collection, or simply for when you can't recall the exact name of similarly named inserts.
- NOTE: the insert does not have to be from the currently displayed collection. Collection displays are for convenience, but do not have any impact on which inserts can be used at any given time (even if they're not displayed at all).

Inserts Selector



- ① Press shift-Tab to reveal the inserts selector list, and type in the input field
 - ② Click inside the list and use the arrow keys to scroll through the list, and press Enter to select an item.
 - ③ Press Return or Enter, or click the Insert button to insert the text into the document.
-

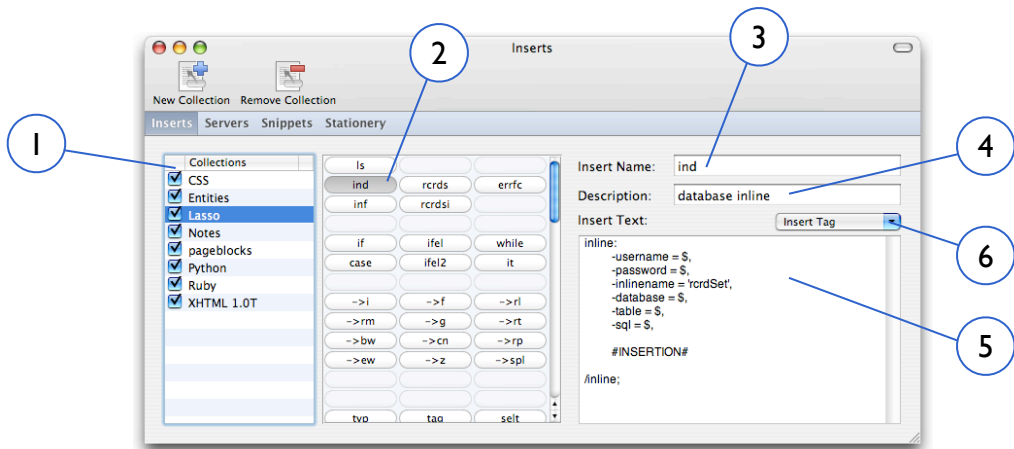
To edit or create an insert definition:

- option-click an insert button, or select Window ▸ Inserts to open the inserts editor window
- the insert name is the text that will be displayed in the button, and is the text that is to be typed for use with the hot key
- the description is displayed as a help tip when the mouse pointer hovers over the insert button

To reorganize insert button order:

- insert buttons can be moved in the inserts editor by dragging the button to be relocated above an existing button position. Existing buttons in that column will be pushed down one position.

Inserts Editor



- ① Collections List—use the checkbox to activate and deactivate a collection.
- ② Insert Arrangement—click a button to edit it, and drag a button to move its position.
- ③ Insert Name—the string entered here is what will be typed to insert expanded text into the document (most characters allowed, no spaces).
- ④ Description—enter a short string to appear as the help tip when the mouse is hovered over the button in the document window.
- ⑤ Insert Text—the text to be inserted when the button is used.
- ⑥ Substitutions Popup—select from this popup to insert substitution codes into the Insert Text field.

To create an insert collection:

- use the insert editor, click the [+] button below the collection list

To change the insert hot key:

- edit the setting in the Autocomplete application preference panel

Working with Snippets

To place a snippet into the currently viewed document:

- reveal the snippets drawer (command-3), and navigate through the file list to the snippet to be inserted
- double-click a snippet file

To open a snippet file for editing its content:

- reveal the snippets drawer (command-3), and navigate through the file list to the snippet to be edited, and option-click the snippet file
- use the global snippets browser using the Window ▸ Snippets menu item to open application-wide snippet files
- use the project window Snippets item in the file shelf to open project-specific snippet files.

To create a new snippet:

- start a new document, edit is as needed
- to save to the global snippets, use File ▸ Save To ▸ Global Snippets
- to save to the project snippets, use File ▸ Save To ▸ Location... to locate the project package, and save the file in the project package's Snippets folder.
- Note: global snippets are stored in ~/Library/Application Support/Araeliu Edit/Snippets
- Note: project snippets are stored inside the project file bundle
- you can create folders in the global and project Snippets folder as needed

Substitutions for Inserts, Snippets, and Stationery

Inserts, Snippets, and Stationery all support the use of text substitutions. These are keywords which are replaced with dynamic values at the time the text is rendered in a document. For example, if an insert or snippet includes the text `#DATE#`, then when that insert or snippet is added to the document, the `#DATE#` text will be replaced with the actual current date.

Substitution Keywords

- `#SELECTED_TEXT#` – substitutes with the highlighted text established prior to the insertion of the insert or snippet text, and does not retain the highlighting after the substitution. Can be used multiple times to repeatedly insert the highlighted text.
- `#SELECTION#` – substitutes with the highlighted text established prior to the insertion of the insert or snippet text, and retains the highlighting of that text after the substitution. Can be used multiple times to repeatedly insert the highlighted text.
- `#SELECT#` – a synonym for `#SELECTION#` for BBEdit compatibility
- `#SELSTART#` – marks the start of a selection range that will be highlighted upon insertion of the text into the document
- `#SELEND#` – marks the end of a selection range that will be highlighted upon insertion of the text into the document
- `#INSERTION#` – places the text cursor at this point in the text
- `#BASENAME#` – substitutes with the file name less the rightmost period delimited segment
- `#FILE#` – substitutes with the complete file name and extension
- `#FILE_EXTENSION#` – substitutes with the rightmost period delimited segment of the file name
- `#DATE#` – substitutes with the value of the Short date as defined in the International pane of the System Preferences.
- `#TIME#` – substitutes with the value of the Medium time as defined in the International pane of the System Preferences.
- `#NAME#` – substitutes with the current user's full user name
- `#INDENT#` – forces the inserted text block to be inserted at the indent level of the current cursor position of the document. Used only for snippets, as indentation level is automatically applied to all inserts.

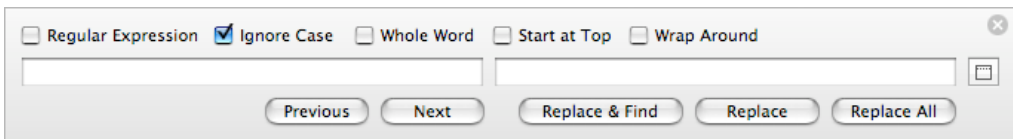
Searching with Find and Find All

The Araelium Edit application has two modes for finding sequential occurrences of text within a document, or finding all occurrences of text within one or more documents.

Using Find

The Find panel is used for search and replace operations within a single document. To use for searching only, leave the Replace field blank, and do not use any of the Replace buttons.

Find panel used for searching in the front document



Standard search order is to search from the current position of the text cursor, and proceed to end of the document and stop. Use the checkbox options to alter the searching behavior. Start at Top will begin the search process from the top of the document regardless of where the cursor position currently is. Wrap Around will start at the current position of the text cursor, proceed to the end of the document, then start at the top of the document and proceed up to the current position of the cursor.

Search is case sensitive unless the Ignore Case checkbox is selected.

Search will find strings as partial words unless Whole Word is selected.

Regular Expression will interpret both the Search for and Replace with fields as regular expressions. See the *Using Regular Expressions* section for information on the regular expression engine used in the Araelium Edit application.

The Find panel opens by default with the search and replace fields as single line text entry boxes. Click the field expansion button at the right side of the fields to expand them both to multiple line text entry boxes.

Search From Toolbar

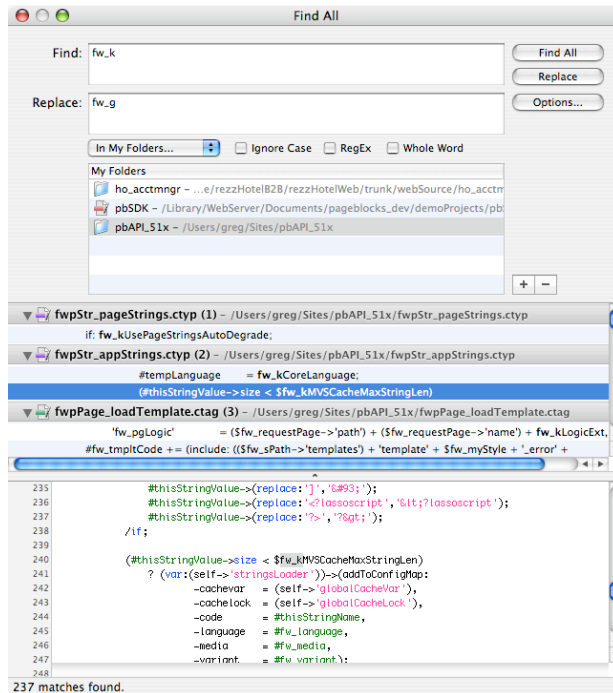
Option-Command-F will place the cursor in the toolbar search field. Using this field is the same as using search from the Find window and will use the current setting of the Find window's Start at Top, Ignore Case, and Whole Word settings.

Using Find All

The Find All window is used to find and find/replace all occurrences of a text string in the front document, in all open documents, in the entire project, or in arbitrary user-identified folders. All sources are searched, and all found occurrences are listed in a results panel of the Find All window.

The Find All window has three sections. At the top is the controls for defining the search options. The center panel is for listing all found occurrences. The bottom panel (which may be hidden) is a document viewer for previewing a found document.

Find All window used for searching multiple documents



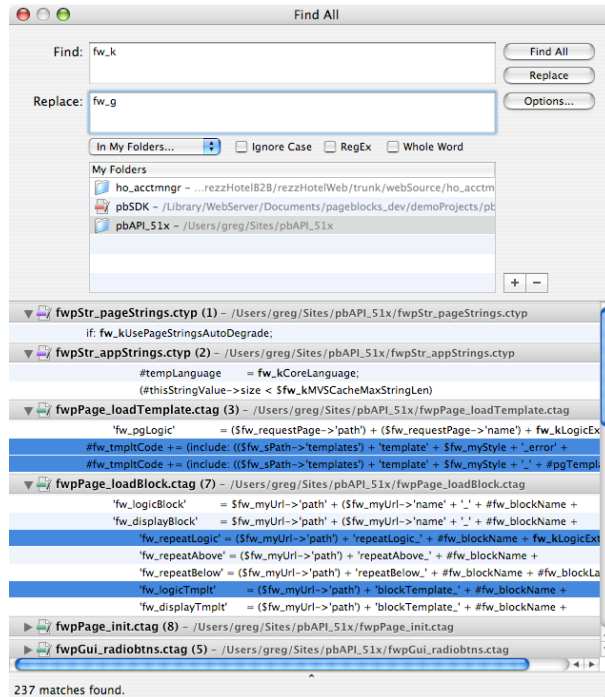
Each found occurrence is listed with the document name, the number of occurrences found, and the pathname of the document. Each document listed can be revealed to list exact found occurrences.

Single-click any specific found occurrence to load the file into the document viewer panel at the bottom of the window (which may require sliding the divider bar from the window bottom). Double-click any specific found occurrence to open the file in a document window.

Find All and Selective Replace

Typically, a replace option will replace text in all files of the found set. However, it is also possible to select specific occurrences to apply the Replace action.

Find All window showing a Replace for only selected occurrences



Using Regular Expressions

The Araeliium Edit application uses the OgreKit wrapper for OniGuruma regular expressions engine. This provides the native Ruby regular expressions engine, and a largely PERL compatible engine. Currently, Araeliium Edit enables all the PERL compatibility features, but we'll be looking into allowing the Ruby native mode as a preference in a future release.

The following section is directly from the OniGuruma documentation with a few minor edits. As for general tutorials, look for some online how-to guides as they'll be able to provide much better instruction than this manual could. One we tend to find quick to use and easy to understand is <http://www.regular-expressions.info/tutorial.html>.

Syntax elements

<code>\</code>	escape
<code> </code>	alternation
<code>(...)</code>	group
<code>[...]</code>	character class

Characters

<code>\t</code>	horizontal tab	(0x09)
<code>\v</code>	vertical tab	(0x0B)
<code>\n</code>	newline	(0x0A)
<code>\r</code>	return	(0x0D)
<code>\b</code>	back space*	(0x08)
<code>\f</code>	form feed	(0x0C)
<code>\a</code>	bell	(0x07)
<code>\e</code>	escape	(0x1B)
<code>\nnn</code>	octal char	(encoded byte value)
<code>\xHH</code>	hexadecimal char	(encoded byte value)
<code>\x{7HHHHHHH}</code>	wide hexadecimal char	(character code point value)
<code>\cx</code>	control char	(character code point value)
<code>\C-x</code>	control char	(character code point value)
<code>\M-x</code>	meta (x 0x80)	(character code point value)
<code>\M-\C-x</code>	meta control char	(character code point value)

* `\b` is effective in character class [...] only

Character Types

```

.      any character (except newline)
\w     word character
      Not Unicode:
          alphanumeric, "-" and multibyte char.
      Unicode:
          General_Category --
          (Letter|Mark|Number|Connector_Punctuation)

\W     non word char
\s     whitespace char
      Not Unicode:
          \t, \n, \v, \f, \r, \x20
      Unicode:
          0009, 000A, 000B, 000C, 000D, 0085(NEL),
          General_Category -- Line_Separator
                           -- Paragraph_Separator
                           -- Space_Separator

\S     non whitespace char
\d     decimal digit char
      Unicode: General_Category -- Decimal_Number

\D     non decimal digit char
\h     hexadecimal digit char  [0-9a-fA-F]
\H     non hexadecimal digit char

```

Quantifier

Greedy

```

?      1 or 0 times
*      0 or more times
+      1 or more times
{n,m}  at least n but not more than m times
{n,}   at least n times
{n,}   at least 0 but not more than n times ({0,n})
{n}    n times

```

Reluctant

```

??     1 or 0 times
*?     0 or more times
+?     1 or more times
{n,m}? at least n but not more than m times
{n,}?  at least n times
{n,}?  at least 0 but not more than n times (== {0,n}?)

```

Possessive

(greedy and does not backtrack after repeated)

<code>?+</code>	1 or 0 times
<code>*+</code>	0 or more times
<code>++</code>	1 or more times

Anchors

<code>^</code>	beginning of the line
<code>\$</code>	end of the line
<code>\b</code>	word boundary
<code>\B</code>	not word boundary
<code>\A</code>	beginning of string
<code>\Z</code>	end of string, or before newline at the end
<code>\z</code>	end of string
<code>\G</code>	matching start position (*)

Character class

<code>^...</code>	negative class (lowest precedence operator)
<code>x-y</code>	range from x to y
<code>[...]</code>	set (character class in character class)
<code>..&&..</code>	intersection (low precedence at the next of ^)

Example:

```
[a-w&&[^c-g]z] ==> ([a-w] AND ([^c-g] OR z)) ==> [abh-w]
```

If you want to use `[`, `-`, or `]` as a normal character in a character class, you should escape these characters by `\`.

POSIX bracket `[[:xxxxx:]]`, negate `[[:^xxxxx:]]`

Not Unicode Case:

<code>alnum</code>	alphabet or digit char
<code>alpha</code>	alphabet
<code>ascii</code>	code value: [0 - 127]
<code>blank</code>	<code>\t</code> , <code>\x20</code>
<code>cntrl</code>	
<code>digit</code>	0-9
<code>graph</code>	include all of multibyte encoded characters
<code>lower</code>	
<code>print</code>	include all of multibyte encoded characters
<code>punct</code>	
<code>space</code>	<code>\t</code> , <code>\n</code> , <code>\v</code> , <code>\f</code> , <code>\r</code> , <code>\x20</code>

upper
xdigit 0-9, a-f, A-F

Unicode Case:

alnum	Letter Mark Decimal_Number
alpha	Letter Mark
ascii	0000 - 007F
blank	Space_Separator 0009
cntrl	Control Format Unassigned Private_Use Surrogate
digit	Decimal_Number
graph	[[^space:]] && ^Control && ^Unassigned && ^Surrogate
lower	Lowercase_Letter
print	[[:graph:]] [[:space:]]
punct	Connector_Punctuation Dash_Punctuation Close_Punctuation Final_Punctuation Initial_Punctuation Other_Punctuation Open_Punctuation
space	Space_Separator Line_Separator Paragraph_Separator 0009 000A 000B 000C 000D 0085
upper	Uppercase_Letter
xdigit	0030 - 0039 0041 - 0046 0061 - 0066 (0-9, a-f, A-F)

Extended groups

(?#...)	comment
(?imx-imx)	option on/off i: ignore case m: multi-line (dot(.) match newline) x: extended form
(?imx-imx:subexp)	option on/off for subexp
(?:subexp)	not captured group
(subexp)	captured group
(?=subexp)	look-ahead
(?!subexp)	negative look-ahead
(?<=subexp)	look-behind
(?<!subexp)	negative look-behind
(?>subexp)	atomic group (don't backtrack in subexp)
(?<name>subexp)	define named group

Subexp of look-behind must be fixed character length. But different character length is allowed in top level alternatives only. Example:

(?<=albc) is OK.
(?<=aaa(?:blcd)) is not allowed.

In negative-look-behind, captured group isn't allowed, but shy group (?:) is allowed.

All characters of the <name> must be a word character. And first character must not be a digit or upper case. Not only a name but a number is assigned like a captured group. Assigning the same name as two or more subexps is allowed. In this case, a subexp call can not be performed although the back reference is possible.

Back Reference

<code>\n</code>	back reference by group number (n >= 1)
<code>\k<name></code>	back reference by group name

In the back reference by the multiplex definition name, a subexp with a large number is referred to preferentially. (When not matched, a group of the small number is referred to.)

Back reference by group number is forbidden if named group is defined in the pattern and `ONIG_OPTION_CAPTURE_GROUP` is not set.

Subexp Call (“Tanaka Akira special”)

<code>\g<name></code>	call by group name
<code>\g<n></code>	call by group number (n >= 1)

Left-most recursive call is not allowed. Example

<code>(?<name>a \g<name>b)</code>	=> error
<code>(?<name>a b\g<name>c)</code>	=> OK

If the option status of called group is different from calling position then the group's option is effective. Example:

`(?-i:\g<name>)(?i:(?<name>a)){0}` match to "A"

Captured Group

Behavior of the no-named group (...) changes with the following conditions. (But named group is not changed.)

Case 1. /.../ (named group is not used, no option)

(...) is treated as a captured group.

Case 2. /.../g (named group is not used, 'g' option)

(...) is treated as a no-captured group (?:...).

Case 3. /...(?<name>...)/ (named group is used, no option)

(...) is treated as a no-captured group (?:...).

numbered-backref/call is not allowed.

Case 4. /...(?<name>...)/G (named group is used, "G" option)

(...) is treated as a captured group.

numbered-backref/call is allowed.

where:

g: ONIG_OPTION_DONT_CAPTURE_GROUP

G: ONIG_OPTION_CAPTURE_GROUP

The Araelium Edit application uses ONIG_OPTION_CAPTURE_GROUP enabled.

Extensions Original to Oniguruma

hexadecimal digit char type	\h, \H
named group	(?<name>...)
named backref	\k<name>
subexp call	\g<name>, \g<group-num>

PERL 5.8.0 Features Not Present

```
[word:]  
\N{name}  
\l,\u,\L,\U, \X, \C  
{code}  
{code}  
(condition)yes-pat | no-pat
```

Specifying Subexpressions in Araelium Edit

Whenever a regular expression option is available and a sub-expr field is provided, that indicates the ability to identify a specific sub-expression for the results to be used. For example, in the Syntax Preferences panel, the Symbols tab allows defining a regular expression to extract a symbol. A sub-expression field is also given. If we wanted to we wanted to extract the foo from text that looked like `define_tag: 'foo'` or like `define_tag('foo')` we would use the following regular expression:

```
define_tag:*(\s*\(\)\s*(\w+)\s*)
```

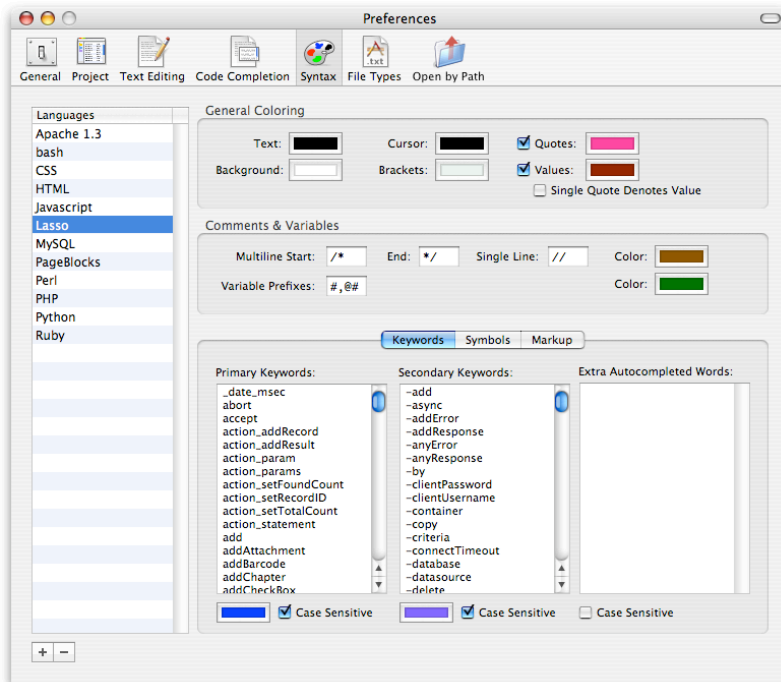
and specify a sub-expression value of 2 which would extract only the `(\w+)` part.

Customizing Syntax Definitions

Using the application Syntax preferences panel, language syntax definitions can be edited and created. Araelium Edit includes some language definitions to start with, but these can be easily modified or added to.

Beta Note: beta releases will have somewhat haphazard language syntax definitions. These will be refined and expanded with each release. It may be possible to get updates from the website between beta releases. Check the support section periodically.

Any additions or changes you make to built-in Syntax and File Types preferences will generate a copy of the default file modified with your changes and stored in `~/Library/Application Support`. To revert to a default Syntax definition, remove the file for that syntax definition from the `~/Library/Application Support/Languages/` folder.



General Coloring

- Text — the color of non-special text within the document.
- Background — the color of the editor background
- Selection — the highlight color of selected text (leave as white to implement the system setting color)
- Brackets — the color of the highlight used to shade text between () {} and [] pairs when the cursor is adjacent to one of the bracket characters.
- Quotes — colors all text between double quotes " " and single quotes ' '
- Values — colors numeric values
- Single Quote Denotes Value — treats all text in single quotes as a value, and thereby colors single quoted text differently from double quoted text

Quoting configurations:

Following are some hints on how to apply the quoting settings:

- If the language uses double quote delimiters for parameters and values (like HTML does), then Quotes should be checked, and a color assigned.
 - `class="doubleQuoteText"`
- If the language makes use of unquoted numeric values (integers, decimals, negative numbers, and numbers followed by %), then Values should be checked and a color assigned.
 - `var percentage = 16.5`
- If Single Quote Denotes Value is left unchecked then all text in single quotes will be colored in the Quotes color assignment.
 - `var percentage = 16.5`
 - `var units = 'widgets';`
- If Single Quote Denotes Value is checked then all text in single quotes will be colored in the Values color assignment.
 - `var percentage = 16.5`
 - `var units = 'widgets';`

Comments and Variables

Multiline and single line comments share a common color. Each field accepts a string used to delimit comments such as /* and */ for the multiline comments, and // or # for single line comments

Variable Prefixes is used to color words which begin with a symbol such as \$myVar, #myVar, or @myVar. Multiple symbols can be entered with each symbol

separated by a comma. Each symbol can be single or multiple-characters. For example, in Ruby, to have both `@myVar` and `@@myVar` colored, enter `@,@@` into the preferences field. Likewise, in Lasso to have variables and referenced variables both colored, you might enter `#,@#` so that both `#myVar` and `@#myVar` will be colored.

Keywords

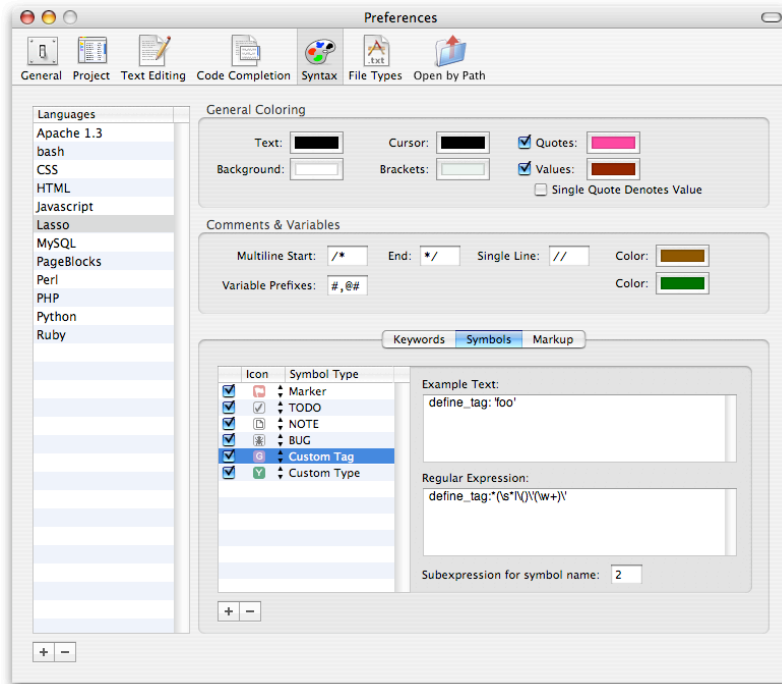
There are three keyword lists. Two of the lists can have individual colors, and the third list of words will not be colored, but will be included in autocomplete choices.

Keywords must be single words comprised of letters, numerals, hyphen, and underscore.

If case sensitive is checked then keywords are colored only if the case as well as the text are a match. Case sensitive being checked also ensures that autocomplete will insert text using the case of the keyword list, otherwise keywords are inserted in all lower case. If you prefer a specific camel case configuration for an otherwise non-case-sensitive language, then modify the keyword list to reflect the case design you prefer, and select the case sensitive checkbox.

Symbols

The Symbols panel allows the definition of named language constructs such as classes, methods, functions, procedures, and variables, as well as any arbitrary pattern of text. The purpose of defining these is to associate an instance of that pattern with a particular icon symbol which is displayed in the symbols list and popup of the document editor. These provide convenient reference points inside documents for having the editor jump to that instance point.



Each Symbol definition includes a name defined in the Symbol Type column, an associated symbol icon, an example of a pattern being identified, and the regular expression for finding the pattern. The checkbox item in the list denotes whether the symbol is displayed in the symbols list and popup menus.

Why would you not want them displayed? Most symbols are going to be defined to capture a single keyword. Those keywords are automatically added to the autocomplete options for that file. This makes it very handy for capturing the function names and variable names within a given document so those keywords can be autocompleted. By leaving the checkbox unchecked, high frequency words like variables won't overwhelm the symbols lists, but the words can still be captured for autocomplete.

Marker Symbols

The example pattern is just for reference in most cases, except for a special case called a Marker. A special Symbol Type named Marker functions a little differently than other definitions. The document editor includes a feature called Insert Marker (menu item and toolbar button). When invoked, the text from the Marker symbol Example Text field is inserted into the document. The associated regular expression would then recognize that pattern, and display the symbol in the symbols list (which is intended to be the red flag icon). This provides a

mechanism for inserting arbitrary reference points in the document that show up in the symbols lists.

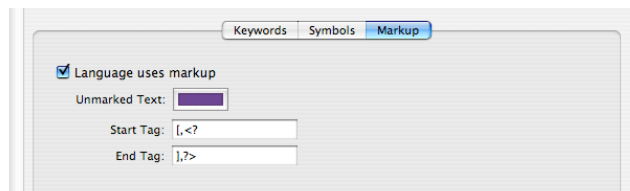
Usually the marker is going to be some type of comment syntax with something unique to identify it from all other comments. For example, a language that uses `//` for single line comments might define example marker text as `//: NewMarker` where the word `NewMarker` is going to show up in the symbols list (until it is edited, in which case that text will appear). So long as other comments don't start with `//:`, then these comments will create handy reference points in the document to jump to.

Tip: if the marker text for a given marker instance is a single hyphen, then the symbols lists in the document window file shelf and popup menu will create a menu divider line. So, to create a divider above a marker, you might enter two lines like this:

```
//: -  
//: NewMarker
```

Markup

Languages like HTML, PHP, JSP, or Lasso that are or can be used as a markup syntax will usually have some starting and ending character(s) to designate markup codes like `<form>` or `[$firstName]`. These characters are defined in the Markup section of the syntax preferences. If a language, like Lasso, allows for more than one set of delimiters, separate them with commas.



Multiple Language Coloring

Applying unique coloring to multiple languages within a file is accomplished in the Preferences File Types pane. For a given File Type (which is a collection of one or more applicable file extensions), up to four language syntaxes may be defined. For example a file type named Plain HTML might be defined to have HTML, JavaScript, and CSS syntaxes applied to `html`, `htm` file extensions.

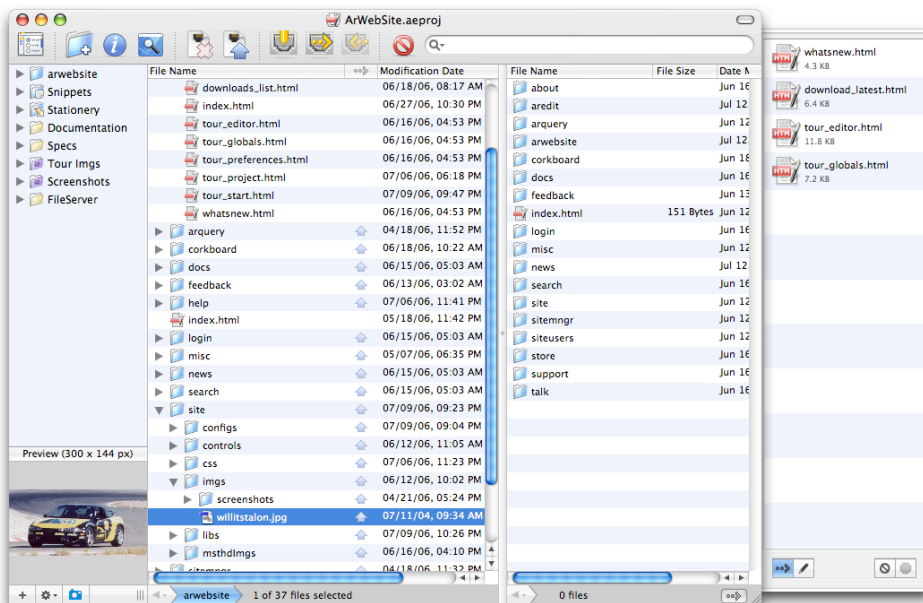
Working with Projects

Araelium Edit projects can bring together in one place all the files associated with the development project. Source code, master graphics files, requirements documents, reference specifications, documentation, project management, and even memos or emails.

Any document that may be an active component of the project, or a useful reference can be organized into the project by including it directly into the project folder, or by reference to the file wherever it is located on your system including server volumes.

Project Window Panels

The project window provides the organization and navigation space for project files and server connections. The far left panel, the project shelf, is the root level of the project contents. The project shelf can display files, real folders (blue), virtual folders (yellow), and smart folders. Two folders are always included: Snippets and Stationery.



Real, Virtual, and Smart Folders

Real folders are folders that exist on the hard drive. Contents of real folders are the actual contents of that disk drive folder. Including a blue folder in a project (at any level), is creating a direct link to that folder and its files. Changes made to the organization of files within blue folders is reflected on the hard drive.

Virtual folders are collections of files within an imaginary folder. Virtual folders do not have a real counterpart on the hard disk, they are a container maintained within the project data structure itself. Files in a virtual folder are located by reference. The file may exist anywhere on local hard drives or server volumes, and files from multiple locations can be grouped within any one virtual folder. Files within virtual folders can be moved around inside any other virtual folder in the project without affecting the file's actual hard drive location.

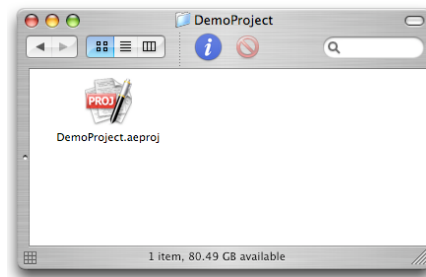
Renaming a file in the project whether it is in a blue folder or yellow folder will rename that file on the hard drive. So, yellow folders are virtual, but the file listing itself is not a virtualization of the file's information. Opening a file from a yellow folder, opens the actual file, and of course, any changes made are being made in the real file, not a copy maintained specifically for the project.

When virtual folders contain files from a server volume, the file references are maintained even when the server is disconnected. Files from a disconnected server volume can't be opened, but once the server is connected again, the file references are reconnected as well. Opening a server volume file will not generate a login attempt (*at least not yet*).

Smart folders are also virtual folders, but they automatically locate files from within the scope of the project to create a list of files that share a common file name pattern as defined by regular expression. (*Currently the user has to define the expression, but future releases will include some built in definitions that will be easy to select.*)

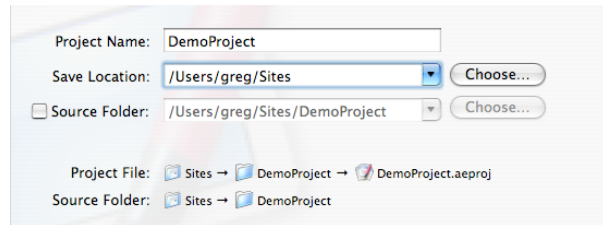
Starting a New Project

By default, new projects consist of a folder with the project package inside. There are some scenarios where this does not have to be the case, but for now, let's assume this is how all projects should be organized. To create this setup, use the default behavior of the New Project Assistant.



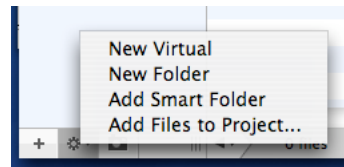
Using the New Project Assistant

Creating a new project requires giving the project a name, and identifying a location for the project save location. Leave the Source Folder checkbox unchecked, and click the Create button. This will create a folder with a project package like show above.

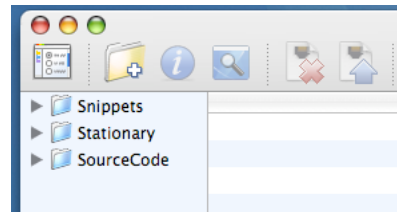


Creating a New Source Code Folder

For most projects, you'll want a primary source code folder. This folder needs to be a blue folder. The best way to do this is to select the New Folder menu item from the project's actions menu at the bottom of the file shelf. To rename the folder, highlight it and press Return. The name can be whatever you want.



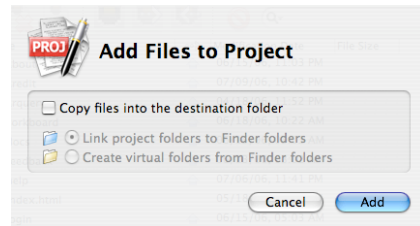
Use this folder to save source code files for the project. One reason this folder is important, is that while files can be saved directly in the project's root folder, those files will not appear anywhere in the project window. So, to have files which are saved to the project automatically appear in the project's window, a blue folder is needed.



There are scenarios where this behavior of files from the project root not appearing in the project window is desirable, but it can be confusing until you experiment more with virtual folders. This behavior allows the project root folder to contain any combination of files which you may not want organized into real folder, but are instead referenced by project yellow folders to give them a logical grouping.

Adding Source Code files to the Project

Creating a new folder as discussed in the above section is likely the normal course of action for a new project. However, it is also possible to reference existing source code files, or copy in a template set of source code files. Referencing existing files allows the project to link to an existing file or folder without copying it or moving it into the project root folder.



Project Tips

To add a real folder to the project shelf:

- control-click in the project shelf, or use the action menu (gear icon) at the bottom of the project shelf to select New Folder

To import files/folders into the project shelf:

- control-click in the project shelf, or use the action menu (gear icon) at the bottom of the project shelf to select Add Files To Project
- select a file or a folder from the Open dialog that is presented
- in File Import Options dialog (shown below), select how you want the files added.
 - Copy files into the destination folder — this will make a duplicate of the selected file or folder (and its contents) into the project. If unchecked, then all files and folders are imported as references to the existing file locations.
 - Link project folders to Finder folders — this will create a blue folder in the project at any point that a folder is encountered on the hard drive while traversing the subdirectories during an import. The files and folders are not copied unless the checkbox is checked, but the file listings in the project are still a direct link to the actual hard drive organization. If Copy files is checked, then the file structure selected is duplicated within the project's source directory (as defined in the project preferences).
 - Create virtual folders from Finder folders — this will create a yellow folder in the project at any point that a folder is encountered on the hard drive while traversing the subdirectories during an import. The files are not copied unless the checkbox is checked, but the original file structure of

the selection is recreated in the project. The creation of yellow folders means that after the import, the files in the project listings can be moved around without impacting the original file structure on disk. If Copy files is checked, then the file structure selected is duplicated within the project's source directory (as defined in the project preferences).

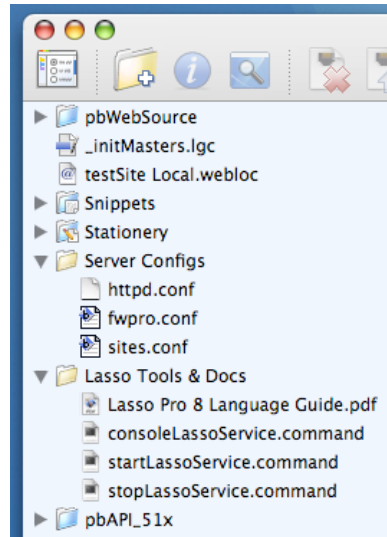
To import files/folders into a project folder:

- follow the same steps outlined above for importing into the project shelf, except open the folder you want to import into, and then choose Add Files to Project.

Use the Project Shelf to Organize Reference Materials

One of the advantages of the Araeliu Edit projects is the ability to centralize diverse resource materials into the project window for quick access.

Items you may want to consider adding to the project shelf include yellow folders for language reference documentation, application documentation, application specifications. You might also want to include .webloc web browser files as convenient launch points for application testing, reference sites, etc. You might also want to include references to frequently accessed source code files such as master config files, or web server config files.



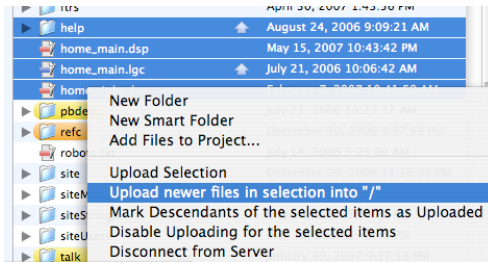
File Transfers in Projects

Araelium Edit has built-in file transfer functionality.

Currently FTP is available, and SFTP, WebDAV, and .Mac are being worked on for the 1.0 release as well. SFTP is functional, but connecting may require closing and re-opening the project.

Define servers for connection with the Servers browser available from Window ▸ Servers menu item. Select a server to connection via the project preferences. When a server is connected to, the project window will add a second file list panel. The server file list can be displayed to the right of the project file list, or below the project file list (see application Project preferences, and the View ▸ Split Horizontally / Vertically menu item).

To upload files, select the files to be transferred and click the upload toolbar button or select the Project ▸ Upload Selection menu item. To upload only the recently modified items of a project, select one or more files/folders and using a right-click, select the Upload Newer menu item. This will upload only those files marked with the blue arrow in the upload status column of the project window.



Yellow folders can be uploaded. A real file structure will be created on the server that mirrors the yellow folder file organization.

Multiple upload sessions can be operational in parallel. Select a folder, click the upload button. Navigate to another folder, and upload it. The two sessions will start and continue until completed.

The transfer history drawer displays current transfer activity and retains the history of past transfers. The view can be switched to show FTP transcripts.

When a selection from the project list is uploaded, it is uploaded to the mirrored location on the target server no matter what the target server view currently is. Unlike other FTP transfers that require opening the local and server file lists to same to location to achieve that, Araelium Edit makes the correlation automatically.

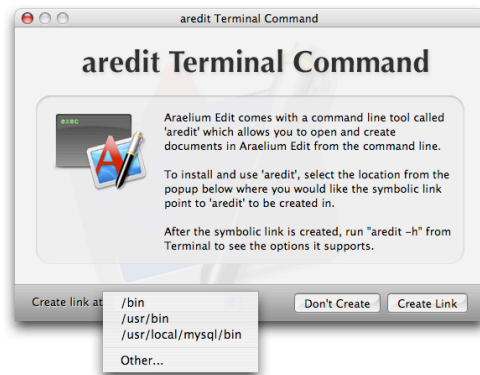
Command Line Tool aredit

Use aredit to open documents from the command line, and allow Araelium Edit to be used for svn logs and more.

Installation

To install the aredit command line tool, select the aredit Terminal Command... menu item from the shebang (!) menu. This will open a window to define where to install the command.

With the Create link at popup menu at the bottom of the window, define where the command line tool is to be installed, and click the Create Link button.



Using aredit

In the Terminal application, type `aredit -h` to see this list of available options to use when opening a file with aredit:

- `-a, --async` = use this option to have a shell script/command continue without waiting for Araelium Edit to close the file
- `-w, --wait` = use this option to have the shell script/command wait for Araelium Edit to close the file before continuing.
- `-n, --no-reactivation` = use in conjunction with a `-w` to have the calling app wait for Araelium Edit to close the file, but then don't bring the calling app to the front (leaves Araelium Edit as the front application).
- `-h, --help` = shows information about these options.

Using aredit with svn

To use Araelium Edit as the editor for making log entries in svn, set svn's `--editor` switch to use: `aredit -w`

Licenses

END USER AGREEMENT

IMPORTANT READ CAREFULLY: This Araelium Group End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Araelium Group for the Araelium Group software product identified above, which includes "online" or electronic documentation and may include computer software and associated media and printed materials ("SOFTWARE PRODUCT" or "SOFTWARE"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, you are not authorized to use the SOFTWARE PRODUCT.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. GRANT OF LICENSE

This EULA grants you the following rights upon your purchase of the SOFTWARE PRODUCT license:

- * Primary Installation and Use. You may install and use copies of the SOFTWARE PRODUCT on a personal computer considered to be your primary computer for use primarily by you.
- * Secondary Installation and Use. You may install and use copies of the SOFTWARE PRODUCT on a personal computer considered to be your secondary computer (for example, a laptop, or home vs. office computer) as long as access to the SOFTWARE PRODUCT will primarily be by you.
- * Backup copy. You may make copies of the SOFTWARE PRODUCT solely for backup or archival purposes.

2. PREVIEW AND BETA VERSIONS

Preview and Beta versions are provided for a time-expired period for the evaluation and initial product testing use. Evaluation versions are not licensed for extended use.

3. EVALUATION VERSIONS

Evaluation versions are provided for one-time 30-day evaluation and initial product testing use. Evaluation versions are not licensed for extended use.

4. OTHER RIGHTS AND LIMITATIONS

You agree to use the SOFTWARE PRODUCT solely for your own personal use and you will not translate, decompile, reverse engineer, disassemble, modify, copy, alter, merge

into other software, reproduce, rent, lease, lend, distribute, or re-market the SOFTWARE PRODUCT or any part thereof.

5. COPYRIGHT

All title and copyrights in and to the SOFTWARE PRODUCT and any copies thereof are owned by Araelium Group. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content.

Araelium Group reserves the right and the sole discretion to (A) make improvements, corrections, adaptations, conversions and/or any other change to the SOFTWARE PRODUCT; (B) change, limit, terminate or cease to provide, at any time, temporarily or permanently, without prior notice, for any reason or no reason, to all users or any number thereof of the SOFTWARE PRODUCT and its availability and functionality, (C) establish new operating and usage policies for the SOFTWARE PRODUCT or change them at any time temporarily or permanently without prior notice, for any reason or no reason.

6. NO WARRANTIES AND NO LIABILITY FOR DAMAGES

YOU EXPRESSLY AGREE THAT DOWNLOADING THE SOFTWARE AND ANY USE OF THE SOFTWARE IS AT YOUR OWN RISK. NO WARRANTY, REPRESENTATION, CONDITION, UNDERTAKING OR TERM - EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE - INCLUDING BUT NOT LIMITED TO THE CONDITION, QUALITY, DURABILITY, PERFORMANCE, ACCURACY, STABILITY, RELIABILITY, NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE OR USE OF THE SOFTWARE IS GIVEN OR ASSUMED BY ARAELIUM GROUP. ALL SUCH WARRANTIES, REPRESENTATIONS, CONDITIONS, UNDERTAKINGS AND TERMS ARE HEREBY EXCLUDED. ARAELIUM GROUP MAKES NO WARRANTY THAT THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT IT WILL BE UNINTERRUPTED, TIMELY, SECURE, OR ERROR FREE; IN NO EVENT SHALL ARAELIUM GROUP BE LIABLE TO ANY PARTY FOR ANY DAMAGES INCLUDING WITHOUT LIMITATION, ANY DIRECT, INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR INFORMATION, LOSS OF PROFITS AND SAVINGS AND THE LIKE), OR ANY OTHER DAMAGES ARISING - IN ANY WAY, SHAPE OR FORM - OUT OF THE AVAILABILITY, USE, RELIANCE ON, INABILITY TO UTILIZE OR IMPROPER USE OF THE SOFTWARE EVEN IF ARAELIUM GROUP SHALL HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND REGARDLESS OF THE FORM OF ACTION, WHETHER IN CONTRACT, TORT, OR OTHERWISE. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, THE ABOVE EXCLUSIONS OF INCIDENTAL AND CONSEQUENTIAL DAMAGES MAY NOT APPLY TO YOU.

Additional Licenses

CodeCog

Copyright 2004-2006, Sal Sorrentino. All rights reserved.

QuickLite

Copyright 2004-2006, Tito Ciuro. All rights reserved.

RBSplitView

Copyright 2004-2006, Rainer Brockerhoff. All rights reserved.

UKKQueue

Copyright 2003-06 by M. Uli Kusterer. All rights reserved.

PSMTabBarController, John Pannell

Copyright 2005-2006, John Pannell Positive Spin Media. All rights reserved.

OAGradientTableView

Copyright 2003-2005 Omni Development, Inc. All rights reserved.

OgreKit

Copyright 2003, Isao Sonobe. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Oniguruma

Copyright 2002-2005, K. Kosako. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF

LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Connection Framework

Copyright 2006, Greg Hulands. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

AquaticPrime

Copyright 2005, Lucas Newman. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.