

Contents

1	Overview	1
2	shu-base	2
2.1	List of functions and variables	2
3	shu-batch-mode	15
3.1	List of functions and variables	15
4	shu-bde-cpp	17
4.1	List of functions by alias name	17
4.2	List of functions and variables	17
5	shu-bde	18
5.1	List of functions by alias name	18
5.2	List of functions and variables	19
6	shu-capture-doc	23
6.1	List of functions and variables	23
7	shu-cpp-general	38
7.1	Selected highlights	38
7.1.1	Dealing with long string constants	38
7.2	List of functions by alias name	38
7.3	List of functions and variables	50
8	shu-cpp-match	75
8.1	List of functions and variables	79
9	shu-cpp-misc	86
9.1	List of functions by alias name	86
9.2	List of functions and variables	87
10	shu-cpp-project	91
10.0.1	Toggle back and forth between files	91
10.0.2	Creating a project file	91
10.0.3	File names within the project	91
10.0.4	Visiting related files	92

10.0.5	Visiting files based on error messages	92
10.1	List of functions by alias name	92
10.2	List of functions and variables	96
11	shu-cpp-token	113
11.1	List of functions by alias name	113
11.2	List of functions and variables	113
12	shu-keyring	122
12.1	List of functions by alias name	123
12.2	List of functions and variables	124
13	shu-match	129
13.1	List of functions by alias name	129
13.2	List of functions and variables	129
14	shu-misc	140
14.1	List of functions by alias name	140
14.2	List of functions and variables	149
15	shu-nvplist	172
15.1	List of functions and variables	172
16	shu-org-extensions	174
16.1	List of functions by alias name	174
16.2	List of functions and variables	174
17	shu-xref	176
17.1	List of functions and variables	176

1 Overview

This manual contains detailed information on all of the function, macro, variable, and constant definitions in this repository.

Version 1.0 was merged with the master branch on 31 December 2018.

Version 1.1 was merged with the master branch on 5 January 2019.

Version 1.2 was merged with the master branch on 6 January 2019.

Version 1.3 was merged with the master branch on 12 January 2019.

Version 1.4 was merged with the master branch on 2 February 2019.

Version 1.5 was merged with the master branch on 18 August 2019.

Version 1.6 was merged with the master branch on 16 November 2019.

This is Version 1.6.108 of the Shu elisp repository.

What this document lacks are detailed scenarios and work flows. The reader might well say that this is an interesting collection of parts, and then go on to ask how to use it. How does one use all of these things in a coherent manner?

I hope to address that in the near future.

One thing I will mention is that the function *shu-capture-all-latex* created the entire LaTeX users manual (shu-manual.tex) and the function *shu-capture-all-md* created the entire markdown version of the users manual (shu-manual.md).

2 shu-base

Collection of miscellaneous functions and constants used by other packages in this repository. Most of the elisp files in this repository depend on this file.

2.1 List of functions and variables

List of functions and variable definitions in this package.

shu-add-to-alist *added-item new-item alist* [Macro]
testfn

Add an item to an alist. The car of *new-item* is a key to be added to the alist *alist*. If the key does not already exist in *alist*, *new-item* is added to *alist*. *added-item* is either the item that was added or the item that was previously there. If (eq *added-item* NEW-ITEM), then *new-item* was added to the list. If (not (eq *added-item* NEW-ITEM)), then the key already existed in the list and *added-item* is the item that was already on the list with a matching key. equal is the function used to determine equality unless *testfn* is supplied, in which case *testfn* is used.

This version of this macro is for emacs 26.1 and later. emacs 26.1 was released on May 28, 2018.

shu-add-to-alist *added-item new-item alist* [Macro]

Add an item to an alist. The car of *new-item* is a key to be added to the alist *alist*. If the key does not already exist in *alist*, *new-item* is added to *alist*. *added-item* is either the item that was added or the item that was previously there. If (eq *added-item* NEW-ITEM), then *new-item* was added to the list. If (not (eq *added-item* NEW-ITEM)), then the key already existed in the list and *added-item* is the item that was already on the list with a matching key. equal is the function used to determine equality.

This version of this macro is for versions of emacs older than 26.1, which was released on May 28, 2018

shu-add-to-alist-list *key value alist* [Function]
alist is an alist in which each element has a car that is the key and a cdr

that is a list of values associated with that key. If the given *key* does not already exist in *alist*, it is added to *alist* with a list of length one containing *value*. If the *key* exists in *alist*, *value* is pushed onto the list of values associated with *key*. The return value is the modified *alist*.

shu-all-commits [Constant]

A list of all commits by version starting with version 1.2

shu-all-whitespace-chars [Constant]

List of all whitespace characters. Since the syntax table considers new-line to be (among other things) a comment terminator, the usual `s-` won't work for whitespace that includes newlines.

shu-all-whitespace-regexp [Constant]

Regular expression to search for whitespace. Since the syntax table considers newline to be (among other things) a comment terminator, the usual

`s-` won't work for whitespace that includes newlines. Note that this regular expression is a character alternative enclosed in left and right brackets. `skip-chars-forward` does not expect character alternatives to be enclosed in square brackets and will include the left and right brackets in the class of characters to be skipped.

shu-all-whitespace-regexp-scf [Constant]

This is the regular expression contained in `shu-all-whitespace-regexp` but with the enclosing left and right square brackets removed. `skip-chars-forward` does not expect character alternatives to be enclosed in square brackets and thus will include the brackets as characters to be skipped.

shu-bool-to-string *arg* [Function]

Convert a boolean value to its string representation and return the string.

shu-comment-start-pattern [Constant]

The regular expression that defines the delimiter used to start a comment.

shu-conditional-load-library-files *path-to-libs* [Function]

Conditionally Load all of the library files listed in in the list *libs* using the path *path-to-libs*. A file in the list is loaded only if FILE-READABLE-P returns true for that file. Return true if all readable files were loaded. Return false if any readable file failed to load.

shu-cpp-author [Custom]

The string to place in the doxygen author tag.

shu-cpp-comment-end [Custom]

Standard end point (right hand margin) for C style comments.

shu-cpp-comment-start [Custom]

Column in which a standard comment starts. Any comment that starts to the left of this point is assumed to be a block comment.

shu-cpp-default-allocator-name [Custom]

The name of the class member variable that holds the pointer to the allocator used by the class.

shu-cpp-default-allocator-type [Custom]

The class name of the standard abstract allocator.

shu-cpp-default-global-namespace [Custom]

The string that defines the default global C++ namespace, if any. If this has a value, then C++ classes are declared with a two level namespace with the global namespace encompassing the local one

shu-cpp-default-member [Custom]

The prefix used to indicate that a variable in a class is a member variable of that class.

shu-cpp-default-namespace [Custom]

The string that defines the default C++ namespace, if any.

shu-cpp-file-directory-name [Constant]

A regular expression to match the name of a C or C++ file in the file system including directory names. This is the same regular expression as *shu-cpp-file-name-list* with a “/” included.

shu-cpp-file-name [Constant]

A regular expression to match the name of a C or C++ file in the file system.

shu-cpp-file-name-list [Constant]

List of all characters that can be present in a C++ file name.

shu-cpp-include-user-brackets [Custom]

Set non-nil if user written include files are to be delimited by angle brackets instead of quotes. In many C and C++ environments, system include files such as `stdio.h` are delimited by angle brackets, for example:

```
#include <stdio.h>
```

while user written include files are delimited by quotes, for example:

```
#include "myclass.h"
```

If this variable is non-nil, then user written include files are delimited by angle brackets and an include of `"myclass.h"` would be written as

```
#include <myclass.h>
```

shu-cpp-indent-length [Custom]

Size of the standard indent for names within class declarations, etc.

shu-cpp-line-end [Custom]

Standard end point (right hand margin) for a line of code.

shu-cpp-modern [Custom]

Set to true if using the features of C++ 11/14 such as auto and explicitly deleted copy / move constructors.

shu-cpp-name [Constant]

A regular expression to match a variable name in a C or C++ program.

shu-cpp-name-list [Constant]

List of all characters that can be present in a C++ variable name.

shu-cpp-std-namespace [Custom]
 The name of the namespace for the C++ standard library. Some users of the BDE open source Bloomberg libraries may prefer “bsl” instead.

shu-cpp-use-bde-library [Custom]
 Set non-nil if the BDE library is to be used for generated C++ code.

shu-current-line [Function]
 Return the line number of the current line relative to the start of the buffer.

shu-current-year [Function]
 Return an integer that represents the four digit current year.

shu-date [Constant]
 Date of the most recent merge with the master branch.

shu-delete-last-char-if *input test-char* [Function]
 Return the string *input* with the last character removed if the last character is equal to the string *test-char*. If the last character is not equal to the string *test-char*, return the input string unmodified.

shu-downcase-first-letter *string* [Function]
 Return the given string with the first character of the string converted to lower case

shu-end-of-dq-string [Function]
 Return the point that represents the end of the current quoted string in the buffer. The quote character is the double quote character (“). Escaped quotes are skipped. If the current string is not terminated by a double quote character, nil is returned. If the current string is terminated by a double quote character, the position following the quote is returned and point is set to that position.

shu-end-of-string *string-term* [Function]
 Return the point that terminates the current quoted string in the buffer. The single argument *string-term* is a string containing the character that started the string (single or double quote). Return nil if the current string is not terminated in the buffer.

This function actually returns the position after the terminating quote and also moves point to that position. This cannot be changed because other functions depend on this frankly strange behavior.

shu-fixed-format-num *num width* [Function]

Return a printable representation of *num* in a string right justified and pad filled to length *width*. The number is formatted as comma separated as defined by shu-group-number.

shu-format-num *num width &optional pad-char* [Function]

Return a printable representation of *num* in a string right justified and pad filled to length *width*. The number is padded with blanks unless an optional *pad-char* is supplied, which is then used instead of blanks.

pad-char must be a character, not a string. If you want the string representation of the number to be right justified and zero filled, specify a pad character of ?0. Do not use a pad character of “0”

shu-get-directory-prefix [Function]

Get a directory based prefix, which is the last name in the current path. If the current directory is “foo/blah/humbug”, the value returned from this function is “humbug”

shu-global-buffer-name [Constant]

The name of the buffer into which shu-global-operation places its output.

shu-goto-line *line-number* [Function]

Move point to line *line-number*.

shu-group-number *num &optional size char* [Function]

Format *num* as string grouped to *size* with *char*. Default *size* is 3. Default *char* is ‘,’. e.g., 1234567 is formatted as 1,234,567. Argument to be formatted may be either a string or a number.

shu-internal-dev-url [Custom]

A string that identifies the internal development URL of an organization (if any).

shu-internal-group-name [Custom]

A string that identifies the group name of which an individual is a member (if any).

shu-invert-alist-list *alist* **&optional** *compare-fn* [Function]

alist is an alist in which the car of each item is the key and the cdr of each item is a list of things associated with the key. This function inverts the alist. The car of each item in the new list is a member of one of the value lists in *alist*. The cdr of each item in the new list is a list of all of those keys that map to the value that is now the key in the new list.

If *compare-fn* is non-nil, then the lists in the car of each item in the new list are sorted using *compare-fn*.

As an example, the following input:

```
A -> (X Y Z)
B -> (Q X Z)
C -> (P X)
```

results in the following output being returned:

```
P -> (C)
Q -> (B)
X -> (A B C)
Y -> (A)
Z -> (A B)
```

shu-invert-alist-to-hash *alist* [Function]

alist is an alist in which the cdr of each item in the list and the car of each item is an associated list of values. The function constructs two items to return.

The first is a hash table that is an inversion of the alist. If the input *alist* contains:

```
A ->(G W)
B ->(X Y)
```

then the returned hash table would be:

G->A
M->A
X->B
Y->B

But if the values contain duplicates, it is impossible to construct the complete hash table. If *alist* contains

A ->(X G W)
B ->(X Y)

then the returned hash table would have to contain

G->A
M->A
X->A
X->B
Y->B

but you cannot have duplicate keys in a hash table. In this latter case, this function constructs an alist that contains the duplicated keys. The key of the alist is the value of what would be a duplicate key in the hash table. The value of the alist is a list of all of the values to which the key maps.

In the case illustrated above, the returned hash table would contain

G->A
M->A
X->A
Y->B

and the returned alist would be

X->(A B)

The return value of this function is a cons cell whose car is the hash table and whose cdr is the alist. If the cdr of the return value is nil, then the entire hash table could be constructed.

shu-kill-new *string* [Function]

Effectively do a kill-new with *string* but use kill-ring-save from a temporary buffer. This seems to do a better job of putting *string* in a place from which other programs running on Linux and Windows can do a paste.

shu-last-commit [Constant]

The git SHA-1 of the most recent commit. This cannot be the SHA-1 hash of the last commit because that is not known until after the commit happens. Just before the merge with master, a commit is done. Its SHA-1 hash is copied into this constant. Then one more commit is done to push this change out. If you want to find this version in git, look for the commit after the one defined here.

shu-library-files [Constant]

A list of all of the library files that comprise the Shu elisp package in the order in which they should be loaded.

shu-line-and-column-at *arg* [Function]

Return the line number and column number of the point passed in as an argument.

shu-line-number-at-pos **&optional** *pos absolute* [Function]

line-number-at-pos in simple.el takes two arguments as of emacs 26. This allows the two argument version to run on older versions of emacs. If *absolute* is specified, widen the buffer, then call the one argument version of line-number-at-pos, which is supported in emacs 24 and 25, and perhaps others.

shu-load-library-files *path-to-libs* [Function]

Load all of the library files listed in *shu-library-files* using the path *path-to-libs*. Return true if all of the files were successfully loaded.

shu-minimum-leading-space *arg* **&optional** [Function]

Find the amount of white space in front of point on the same line and return either that count or *arg*, whichever is smaller. Used by functions that wish to safely delete *arg* characters of white space from the current position without deleting any characters that are not white space. An

optional second argument, *white-space*, is a string that defines what is meant by white space. The default definition is *shu-all-whitespace-regexp*.

shu-non-cpp-name [Constant]

A regular expression to match a character not valid in a variable name in a C or C++ program.

shu-not-all-whitespace-regexp [Constant]

Regular expression to search for non-whitespace. Since the syntax table considers newline to be (among other things) a comment terminator, the usual

s- won't work for whitespace that includes newlines. Note that this regular expression is a character alternative enclosed in left and right brackets. *skip-chars-forward* does not expect character alternatives to be enclosed in square brackets and will include the left and right brackets in the class of characters to be skipped.

shu-point-at-sexp *sos* [Function]

Return the point of the sexp that matches the point at *sos*.

shu-point-in-string &optional *pos* [Function]

Return the start position of the string text if point is sitting between a pair of non-escaped quotes (double quotes). The left-hand quote (opening quote) must be on the same line as point. The string must be on a single line. If point is sitting on a quote, then it is not inside a string. In order to be inside a string, point must lie between two non-escaped quotes. The optional argument *pos*, if specified, is used in place of the position of point.

shu-random-letter [Function]

Return a randomly selected lower case letter as a single character (not as a string).

shu-random-range *x y* [Function]

Return a random number that lies within the closed interval $[x, Y]$. If $y < x$, then the closed interval is $[y, X]$. If y is equal to x , then the returned value is x .

shu-remove-trailing-all-whitespace *input-string* [Function]

Return a copy of *input-string* with all trailing whitespace removed. All control characters are considered whitespace.

shu-replace-string *original replacement &optional literal* [Function]

Go to the top of the current buffer and use SEARCH-FORWARD to find all instances of *original*, replacing each instance with *replacement*. The optional arguments *fixedcase* and *literal* are passed through to the REPLACE-MATCH function. Return the count of the number of instances that were replaced, if any.

shu-split-new-lines *data &optional separator* [Function]

Split a string into a list of strings. If the optional *separator* is present, it is used as a regular expression that represents the line separator and it is not retained in each split line. If *separator* is not present, the separator is the newline character. The separator expressions are removed from the input string and a list of separated strings is returned.

If the input line is an empty string, a list containing one empty string is returned. If the line contains a trailing new line character, that trailing new line character is discarded without generating a new, empty line in the output.

In the following examples *separator* has been set to a comma.

For example, the input string “Hello,There,” will return exactly the same output list as the input string “Hello,There”.

As another example, the input string “Hi,How are you?,” returns a list of two strings, which are “Hi” and “How are you?”.

shu-starts-with *regexp* [Function]

If the first non-whitespace on the current line matches *regexp*, return the position of the beginning of the matched *regexp*. If the first non-whitespace does not match *regexp*, return nil.

shu-string-starts-ends *string start-string end-string* [Command]

Return true if the given *string* starts with *start-string* and ends with *end-string*. If *end-string* is omitted, *start-string* is used instead. An

empty *start-string* matches anything. An empty *end-string* matches anything.

shu-swap *x y* [Macro]
Swap the contents of *x* and *y*. *x* gets the value of *y*. *y* gets the value of *x*.

shu-the-column-at *arg* [Function]
Return the column number of the point passed in as an argument.

shu-the-line-at *arg* [Function]
Return the line number of the point passed in as an argument. The line number is relative to the start of the buffer, whether or not narrowing is in effect.

shu-trace-buffer [Constant]
The name of the buffer into which some functions write debug trace information.

shu-trim *string* [Function]
Trim leading and trailing whitespace from *string*. Return the modified string. String remains unmodified if it had no leading or trailing whitespace.

shu-trim-file [Custom]
If true, whitespace is trimmed from the end of lines and blank lines at the end of a file are deleted when a file is saved.

shu-trim-file-hook [Function]
Run as a before-save-hook to trim trailing whitespace from the end of lines and to trim blank lines from the end of a file if *shu-trim-file* is true.

shu-trim-leading *string* [Function]
Trim leading whitespace from *string*. Return the modified string. String remains unmodified if it had no leading whitespace.

shu-trim-trailing *string* [Function]
Trim trailing whitespace from *string*. Return the modified string. String

remains unmodified if it had no trailing whitespace.

shu-unit-test-buffer [Constant]

The name of the buffer into which unit tests place their output and debug trace.

shu-upcase-first-letter *string* [Function]

Return the given string with the first character of the string converted to upper case

shu-version [Constant]

The version number of the Shu elisp package.

3 shu-batch-mode

A startup script and other functions for running the Shu elisp package in batch mode. Some of the functions in this package have been enabled to run as stand alone shell scripts. This allows these functions to be used by non-emacs users and in automated build work flows.

The function `shu-batch-init` loads all of the Shu elisp libraries. Other functions in this file invoke various functions in the Shu elisp libraries and are set up such that they can be invoked from the `-eval` comment line option of emacs running in batch.

3.1 List of functions and variables

List of functions and variable definitions in this package.

`shu-batch-add-alexandria` [Function]

Call the *shu-add-alexandria* function in batch mode. One required argument is the value for the custom variable *shu-internal-dev-url*.

Invoke as:

```
emacs --batch -l ~/emacs/shu-batch-mode.elc -f shu-batch-add-alexandria <dev-url>
```

where “<dev-url>” is the organization’s internal web site that hosts its code and tools. See the description of the custom variable *shu-internal-dev-url* for more information.

`shu-batch-copy-trace` [Function]

Copy the contents of the *shu-trace-buffer* to stdout in batch mode.

`shu-batch-fail` [Command]

A test function to terminate emacs via ERROR.

`shu-batch-hello` [Command]

A test function for batch mode.

`shu-batch-init` [Function]

Load all of the .elc files from the Shu elisp package. This is used to allow Shu elisp functions to be used in batch mode. This function

searches for the functions in the path specified by the environment variable “SHU Emacs_LOAD_PATH”. If that environment variable does not exist, then it searches in the local “ /emacs” directory.

shu-batch-rmv-using [Function]
Call rmv-using in batch mode.

shu-batch-test-args [Function]
A script to use in batch mode to demonstrate how to fetch command line arguments.

When run from a batch script as the function that is the target of the “-f” option. For example

```
emacs --batch -l shu-batch-mode.elc -f shu-batch-test-args hello world how are you
```

produces the following output:

```
There are 5 arguments
arg: 0: ‘‘hello’’
arg: 1: ‘‘world’’
arg: 2: ‘‘how’’
arg: 3: ‘‘are’’
arg: 4: ‘‘you’’
```

shu-generate-comdb2-code [Function]
Generate the C++ code for a comdb2 row class

shu-generate-component [Function]
Fetch the arguments from environment variables and then call *shu-internal-gen-bde-component* to generate a set of three BDE component files.

shu-local-class-list [Constant]
List of standard namespaces and their associated classes

shu-old-generate-component [Function]
Fetch the arguments from environment variables and then call *shu-internal-gen-bde-component* to generate a set of three BDE component files.

4 shu-bde-cpp

A collection of useful functions for generating C++ skeleton code files and classes for code written in Bloomberg, L.P. BDE style.

4.1 List of functions by alias name

A list of aliases and associated function names.

gen-bb-component *class-name* [Command]
(Function: shu-gen-bb-component)
Generate the three files for a new component: .cpp, .h, and .t.cpp

4.2 List of functions and variables

List of functions and variable definitions in this package.

shu-bb-cpp-set-alias [Function]
Set the common alias names for the functions in shu-bb-cpp. These are generally the same as the function names with the leading shu- prefix removed.

shu-gen-bb-component *class-name* [Command]
(Alias: gen-bb-component)
Generate the three files for a new component: .cpp, .h, and .t.cpp

shu-generate-bb-cfile *author namespace class-name* [Function]
Generate a skeleton cpp file

shu-generate-bb-hfile *author namespace class-name* [Function]
Generate a skeleton header file

shu-generate-bb-tfile *author namespace class-name* [Command]
Generate a skeleton t.cpp file

5 shu-bde

A collection of useful functions for generating C++ skeleton code files and classes for code written in Bloomberg, L.P. BDE style.

5.1 List of functions by alias name

A list of aliases and associated function names.

bde-add-guard [Command]

(Function: shu-bde-add-guard)

Add the BDE include guards around an existing `#include` directive. If the line before the `#include` directive contains a valid guard, then we do not add a guard and position point to the line following the `#include`. This makes it possible to run `bde-all-guard` on a file that contains some guarded `#includes` and some unguarded `#includes`. Only the unguarded ones will have the guard added.

bde-all-guard [Command]

(Function: shu-bde-all-guard)

Add the BDE include guards around all of the `#include` directives in a file or narrowed region.

bde-decl *class-name* [Command]

(Function: shu-bde-decl)

Generate a skeleton BDE class declaration at point.

bde-gen *class-name* [Command]

(Function: shu-bde-gen)

Generate a skeleton BDE class code generation at point.

bde-include *fn* [Command]

(Function: shu-bde-include)

Insert at the current line, the BDE include guard sequence of `#ifndef INCLUDED_GUARD #include <guard.h> #endif`

bde-sdecl *class-name* [Command]

(Function: shu-bde-sdecl)

Generate a skeleton BDE struct definition at point.

bde-sgen *class-name* [Command]
 (Function: shu-bde-sgen)
 Generate a skeleton BDE struct code generation at point.

gen-bde-component *class-name* [Command]
 (Function: shu-gen-bde-component)
 Generate the three files for a new component: .cpp, .h, and .t.cpp

5.2 List of functions and variables

List of functions and variable definitions in this package.

shu-bde-add-guard [Command]
 (Alias: bde-add-guard)
 Add the BDE include guards around an existing `#include` directive. If the line before the `#include` directive contains a valid guard, then we do not add a guard and position point to the line following the `#include`. This makes it possible to run `bde-all-guard` on a file that contains some guarded `#includes` and some unguarded `#includes`. Only the unguarded ones will have the guard added.

shu-bde-all-guard [Command]
 (Alias: bde-all-guard)
 Add the BDE include guards around all of the `#include` directives in a file or narrowed region.

shu-bde-decl *class-name* [Command]
 (Alias: bde-decl)
 Generate a skeleton BDE class declaration at point.

shu-bde-gen *class-name* [Command]
 (Alias: bde-gen)
 Generate a skeleton BDE class code generation at point.

shu-bde-gen-cfile-copyright-hook [Custom]
 Generate the text that is the copyright notice placed in a code file, if any.

shu-bde-gen-file-identifier-hook [Custom]

Generate the text that constitutes a source file identifier, if any.

shu-bde-gen-h-includes-hook [Custom]

Generate the code for the standard includes in a header file.

shu-bde-gen-hfile-copyright-hook [Custom]

Generate the text that is the copyright notice placed in a header file, if any.

shu-bde-gen-tfile-copyright-hook [Custom]

Generate the text that is the copyright notice placed in a unit test file, if any.

shu-bde-include *fn* [Command]

(Alias: bde-include)

Insert at the current line, the BDE include guard sequence of `#ifndef INCLUDED_GUARD #include <guard.h> #endif`

shu-bde-include-guard &optional *fn* [Function]

Return the name of the macro variable to be used in a BDE style include guard. Name of the current buffer file name is used if no file name is passed in as the only optional argument. This is the name of the macro variable that is used in the include guard. If the name of the file is `foo_something.h`, then this function returns `INCLUDED_FOO_SOMETHING`. See also `shu-bde-include-guard-fn`

shu-bde-include-guard-fn &optional *fn* [Function]

Return the file name name of the macro variable to be used in a BDE style include guard. Name of the current buffer file name is used if no file name is passed in as the only optional argument. This is only the file name part of the include guard. If the name of the file is `foo_something.h`, then this function returns `FOO_SOMETHING`. The full name of the macro variable would be `INCLUDED_FOO_SOMETHING`. See also `shu-bde-include-guard`

shu-bde-insert-guard *fn at-top* [Function]

Insert a `#ifndef / #endif` guard around an `#include` directive. *fn* is the name of the included file. *at-top* is true if the `#include` directive is located on the first line of the file so there is no line above it.

shu-bde-sdecl *class-name* [Command]
 (Alias: bde-sdecl)
 Generate a skeleton BDE struct definition at point.

shu-bde-set-alias [Function]
 Set the common alias names for the functions in shu-bde. These are generally the same as the function names with the leading shu- prefix removed.

shu-bde-sgen *class-name* [Command]
 (Alias: bde-sgen)
 Generate a skeleton BDE struct code generation at point.

shu-gen-bde-component *class-name* [Command]
 (Alias: gen-bde-component)
 Generate the three files for a new component: .cpp, .h, and .t.cpp

shu-gen-bde-create-prompt [Function]
 This function creates the prompt for the interactive special form of the function *shu-gen-bde-component*. The prompt includes the namespace in which the new class will be created or the string “NO NAMESPACE” if there is no default namespace set. If the name of the current directory does not match the default namespace, the prompt also includes the directory name to remind the user that the current directory name does not match the namespace.

shu-generate-bde-cfile *author namespace class-name* [Function]
 Generate a skeleton cpp file

shu-generate-bde-hfile *author namespace class-name* [Function]
 Generate a skeleton header file

shu-generate-bde-tfile *author namespace class-name* [Command]
 Generate a skeleton t.cpp file

shu-generate-git-add *filename gitbuf* [Function]
 Do a “git add” of *filename* and show the result of the operation in the buffer *gitbuf*.

`shu-internal-gen-bde-component` *class-name author* [Function]
file-prefix

Generate the three files for a new component: .cpp, .h, and .t.cpp

6 shu-capture-doc

Collection of functions used to capture function and variable definitions along with their associated doc strings in elisp code. It can then write this information into a buffer in either markdown or LaTeX format for subsequent publication.

This mechanism was used to create most of the documentation for the elisp functions in this repository.

6.1 List of functions and variables

List of functions and variable definitions in this package.

shu-capture-a-type-after [Constant]

The a-list key value that identifies the string that is placed after a verbatim code snippet.

shu-capture-a-type-arg [Constant]

The a-list key value that identifies the function that converts an argument name to markup.

shu-capture-a-type-before [Constant]

The a-list key value that identifies the string that is placed before a verbatim code snippet.

shu-capture-a-type-buf [Constant]

The a-list key value that identifies the function that converts a buffer name or other name that begins and ends with asterisks to markup.

shu-capture-a-type-close-quote [Constant]

The a-list key value that identifies the string that is a close quote.

shu-capture-a-type-doc-string [Constant]

The a-list key value that identifies the function that converts a key word, such as “&optional” or “&rest” to markup.

shu-capture-a-type-enclose-doc [Constant]

The a-list key value that identifies the function that converts a key word, such as “&optional” or “&rest” to markup.

<code>shu-capture-a-type-func</code>	[Constant]
The a-list key value that identifies the function that formats a function signature'	
<code>shu-capture-a-type-hdr</code>	[Constant]
The a-list key value that identifies the function that emits section headers	
<code>shu-capture-a-type-keywd</code>	[Constant]
The a-list key value that identifies the function that converts a key word, such as “&optional” or “&rest” to markup.	
<code>shu-capture-a-type-open-quote</code>	[Constant]
The a-list key value that identifies the string that is an open quote.	
<code>shu-capture-alias-list</code>	[Variable]
The alist that holds all of the alias names.	
<code>shu-capture-aliases</code>	[Function]
Undocumented	
<code>shu-capture-all-latex</code>	[Command]
Visit all of the files in <i>shu-capture-file-list</i> , invoking <i>shu-capture-latex</i> on each file to capture its documentation and turn it into LaTeX source.	
<code>shu-capture-all-md</code>	[Command]
Visit all of the files in <i>shu-capture-file-list</i> , invoking <i>shu-capture-md</i> on each file to capture its documentation and turn it into markdown source.	
<code>shu-capture-arg-to-latex</code> <i>arg-name</i>	[Function]
Convert a function argument in a doc-string or argument list to LaTeX.	
<code>shu-capture-arg-to-md</code> <i>arg-name</i>	[Function]
Convert a function argument in a doc-string or argument list to markdown.	
<code>shu-capture-attr-alias</code>	[Constant]
Bit that indicates that a function is identified by its alias name	

<code>shu-capture-attr-const</code>	[Constant]
Bit that indicates that a definition is a defconst	
<code>shu-capture-attr-custom</code>	[Variable]
Bit that indicates that a definition is a defcustom	
<code>shu-capture-attr-inter</code>	[Constant]
Bit that indicates that a function is interactive	
<code>shu-capture-attr-macro</code>	[Constant]
Bit that indicates that a function is a macro	
<code>shu-capture-attr-var</code>	[Variable]
Bit that indicates that a definition is a defvar	
<code>shu-capture-buf-to-latex</code> <i>buf-name</i>	[Function]
Convert a buffer name or other name that starts and ends with asterisks in a doc-string to markdown.	
<code>shu-capture-buf-to-md</code> <i>buf-name</i>	[Function]
Convert a buffer name or other name that starts and ends with asterisks in a doc-string to markdown.	
<code>shu-capture-buffer-name</code>	[Constant]
Name of the buffer into which the converted documentation is written	
<code>shu-capture-code-in-doc</code> <i>before-code after-code</i> <i>section-converter</i>	[Function]

The current buffer is assumed to hold a doc string that is being converted to either markdown or LaTeX. We divide the text into two categories. The first category is plain text that should be scanned for characters to escape, such as pound signs if we are converting to LaTeX. The second category is text that should not be scanned for characters to escape, either because it is to be treated as a verbatim code snippet or because it is a pseudo markdown section heading that will be converted either to a markdown section heading or to a LaTeX section heading.

When we come to the end of plain text (either because we have found a code snippet or because we have found a pseudo markdown section heading), we call the *text-converter* function on the bounds of the plain

text whose end we have just found.

A pseudo markdown section heading is identified as follows. It must start in column 1. It must start with two to four pound signs. It must have some text. It must end at the end of the line with the same number of pound signs with which it started.

A code snippet to be shown in verbatim mode is any one whose first column occurs on or after *shu-capture-doc-code-indent*.

When the *text-converter* function is called. It may expand the size of the text area if it adds characters to the text. It is the responsibility of the *text-converter* function to return the new text end point to this function.

shu-capture-code-in-md [Function]

The current buffer is assumed to hold a doc string that is being converted to markdown. Any line that is indented to column *shu-capture-doc-code-indent* or greater is assumed to be a code snippet and will be surrounded by “” to make it a code snippet in markdown. Return the number of code snippets marked.

shu-capture-commentary [Function]

Search through an elisp file for a package name and a commentary section. Return a cons cell whose car is the package name and whose cdr is the prose found in the commentary section.

shu-capture-convert-args-to-markup *signature* [Function]
keywd-converter

signature contains the function signature (both function name and arguments). *arg-converter* is the function used to convert an argument to markup. *keywd-converter* is the function used to convert an argument list keyword, such as “&optional” or “&rest” to markup.

This function returns a cons cell pointing to two lists. The first list contains the length of each argument name prior to conversion to markup. This is because the amount of space on a line is largely determined by the length of the unconverted argument. “arg” will take much less space on a line than will the same word with markup added. The second list contains each of the argument names converted to the appropriate markup.

Given the following function signature:

```
do-something (with these things \&optional and \&rest others)
```

the length list will contain (4, 5, 6, 9, 3, 5, 6). The converted arguments list for markdown will contain (“{*with*}”, “{*these*}”, “{*things*}”, “{**}&optional{**}”, “{*and*}”, “{**}&rest{**}”, “{*others*}”).

If the function signature contains no arguments, then nil is returned instead of the above described cons cell.

`shu-capture-convert-doc-string` *signature* [Function]
converters

description contains a doc string from a function definition (with leading and trailing quotes removed). *converters* is an a-list of functions and strings as follows:

Key	Value
---	----
shu-capture-a-type-hdr	Function to format a section header
shu-capture-a-type-func	Function to format a function signature
shu-capture-a-type-buf	Function to format a buffer name
shu-capture-a-type-arg	Function to format an argument name
shu-capture-a-type-keywd	Function to format a key word
shu-capture-a-type-doc-string	Function to finish formatting the doc string
shu-capture-a-type-enclose-doc	Function to enclose doc string in begin / end
shu-capture-a-type-before	String that starts a block of verbatim code
shu-capture-a-type-after	String that ends a block of verbatim code
shu-capture-a-type-open-quote	String that is an open quote
shu-capture-a-type-close-quote	String that is a close quote

This function turns escaped quotes into open and close quote strings, turns names with leading and trailing asterisks (e.g., `{**project-buffer**}`) into formatted buffer names, turns upper case names that match any argument names into lower case, formatted argument names. This is an internal function of `shu-capture-doc` and will likely crash if called with an invalid a-list.

`shu-capture-convert-func-latex` *func-def converters* [Function]

Take a function definition and turn it into a string of LaTeX. Return said string.

shu-capture-convert-func-md *func-def converters* [Function]

Take a function definition and turn it into a string of markdown. Return said string.

shu-capture-convert-quotes *open-quote close-quote* [Function]

Go through the current buffer converting any escaped quote to either an open or close quote. If an escaped quote is preceded by whitespace, “(“, “{“, “<“, or “>“, or by a close quote, then we replace it with an open quote. Otherwise we replace it with a close quote.

shu-capture-doc *converters* [Function]

Top level function that captures all definitions and doc strings in a language neutral manner and then uses the supplied *converters* to convert the documentation to either markdown or LaTeX.

shu-capture-doc-code-indent [Constant]

Any line indented by this much in a doc string is assumed to be a sample code snippet.

shu-capture-doc-convert-args *signature converters* [Function]

The current buffer contains a doc string from a function. The argument to this function is the *signature* of the function for which the doc string was written. This function goes through the doc string buffer looking for any word that is all upper case. If the upper case word matches the name of an argument to the function, it is passed to the CONVERTER function for conversion into a markup language, which is probably markdown or LaTeX, and it is then replaced in the doc string buffer.

For example, if the function has the following signature:

```
do-something (hat cat)
```

with the following doc string:

“The Linux HAT is converted to an IBM CAT.”

would be converted to:

“The Linux $\{\text{hat}\}$ is converted to an IBM $\{\text{cat}\}$.”

`shu-capture-doc-convert-args-to-latex` *signature* [Function]
Undocumented

`shu-capture-doc-convert-args-to-md` *signature* [Function]
Undocumented

`shu-capture-enclose-doc-latex` [Function]
Enclose the doc-string with the appropriate begin / end pair for LaTeX.

`shu-capture-enclose-doc-md` [Function]
Enclose the doc-string with the appropriate begin / end pair for markdown.

`shu-capture-file-list` [Constant]
This is a list of all of the files in this repository from which documentation should be extracted.

`shu-capture-finish-doc-string-latex` [Command]
Function that executes last step in the conversion of a doc-string to markdown.

`shu-capture-finish-doc-string-md` [Function]
Function that executes last step in the conversion of a doc-string to markdown.

`shu-capture-func-type-name` *attributes* [Command]
Return the name of the type “Alias,” “Macro,” “Constant,” “Variable,” or “Function” based on the *attributes* passed in.

`shu-capture-get-args-as-alist` *signature* [Function]
signature contains the function signature (both function name and arguments). This function returns the arguments as an a-list in which all of the argument names are the keys. The special argument names “&optional” and “&rest”, if present, are not copied into the a-list.

For example, if *signature* holds the following:

do-something (with these things \&optional and \&rest others)

an a-list is returned with the keys “others,” “and,” “things,” “these,” and “with.”

`shu-capture-get-doc-string` *eof* [Command]

Enter with point positioned immediately after a function declaration. Try to fetch the associated doc string as follows: 1. Look for the first open or close parenthesis. 2. Look for the first quote. If the first parenthesis comes before the first quote, then there is no doc string. In the following function, there is no doc string:

```
(defun foo (name)
  (interactive ‘‘sName?: ‘’))
```

but if we do not notice that the first parenthesis comes before the first quote, then we might think that there is a doc string that contains “sName?: “.

Return the doc string if there is one, nil otherwise.

`shu-capture-get-func-def` *func-def signature* [Macro]
description alias

Extract the information from the func-def

`shu-capture-get-func-def-alias` *func-def alias* [Macro]

Extract the function alias from the func-def

`shu-capture-get-func-def-sig` *func-def signature* [Macro]

Extract the function signature from the func-def

`shu-capture-get-name-and-args` *signature func-name* [Macro]

Extract the function and the string of arguments from a whole signature that includes both the function name and the arguments. If *signature* contains:

```
‘‘do-something (to something)’’
```

The on return *func-name* will hold “do-something” and *args* will contain the string “(to something)”. If there are no arguments, *args* will contain a string

of length zero. If there is no function name, *func-name* will contain a string of length zero

shu-capture-headers-in-doc *section-converter* [Function]

Convert markdown section headers to either markdown or LaTeX. This allows the author of some Commentary at the beginning of a file to add section headers. If the heading level is 2 through 4 and the heading begins in column 1 and the number of pound signs at the end is the same as the number of pound signs at the beginning and the pound signs at the end are at the end of a line, then this is considered to be a heading and is translated to either markdown or LaTeX.

shu-capture-index-buffer [Constant]

Name of the buffer into which the markdown index is written

shu-capture-internal-all *file-list capture-func* [Function]

Visit all of the files in *file-list*, invoking *capture-func* on each file to capture its documentation and turn it into either LaTeX or markdown.

shu-capture-internal-convert-doc-string *signature converters* [Function]

description contains a doc string from a function definition (with leading and trailing quotes removed). *converters* is an a-list of functions and strings as follows:

Key	Value
---	----
shu-capture-a-type-hdr	Function to format a section header
shu-capture-a-type-func	Function to format a function signature
shu-capture-a-type-buf	Function to format a buffer name
shu-capture-a-type-arg	Function to format an argument name
shu-capture-a-type-keywd	Function to format a key word
shu-capture-a-type-doc-string	Function to finish formatting the doc string
shu-capture-a-type-enclose-doc	Function to enclose doc string in begin / end
shu-capture-a-type-before	String that starts a block of verbatim code
shu-capture-a-type-after	String that ends a block of verbatim code
shu-capture-a-type-open-quote	String that is an open quote
shu-capture-a-type-close-quote	String that is a close quote

This function turns escaped quotes into open and close quote strings, turns

names with leading and trailing asterisks (e.g., `{**project-buffer**}`) into formatted buffer names, turns upper case names that match any argument names into lower case, formatted argument names. This is an internal function of `shu-capture-doc` and will likely crash if called with an invalid a-list.

`shu-capture-internal-doc` [Command]

Function that captures documentation for all instances of “defun,” “defsubst,” and “defmacro.”

`shu-capture-keywd-optional` [Constant]

The argument list keyword for an optional argument.

`shu-capture-keywd-rest` [Constant]

The argument list keyword for a multiple optional arguments.

`shu-capture-keywd-to-latex` *keywd-name* [Function]

Convert a function argument key word in a doc-string or argument list to LaTeX.

`shu-capture-keywd-to-md` *arg-name* [Function]

Convert a function argument key word in a doc-string or argument list to markdown.

`shu-capture-latex` [Command]

Capture all of the function and macro definitions in a .el source file and turn them into a LaTeX text that documents the functions and their doc strings.

`shu-capture-latex-arg-end` [Constant]

Define the latex string that is used to terminate an argument name.

`shu-capture-latex-arg-start` [Constant]

Define the latex string that is used to prepended to an argument name.

`shu-capture-latex-buf-end` [Constant]

Define the LaTeX string that is used at the end of a buffer name or any other name that has leading and trailing asterisks

`shu-capture-latex-buf-start` [Constant]

Define the LaTeX string that is used in front of a buffer name or any

other name that has leading and trailing asterisks

<code>shu-capture-latex-close-quote</code>	[Constant]
Define the LaTeX string that is a close quote.	
<code>shu-capture-latex-code-end</code>	[Constant]
Define the LaTeX string that is at the end of a verbatim code snippet.	
<code>shu-capture-latex-code-start</code>	[Constant]
Define the LaTeX string that is at the beginning of a verbatim code snippet.	
<code>shu-capture-latex-converters</code>	[Constant]
This is the association list of functions and strings that is used to take an elisp function and its associated doc string and convert it to LaTeX.	
<code>shu-capture-latex-doc-end</code>	[Constant]
Define the LaTeX string that ends a doc string.	
<code>shu-capture-latex-doc-start</code>	[Constant]
Define the LaTeX string that starts a doc string.	
<code>shu-capture-latex-keywd-end</code>	[Constant]
Define the latex string that is used to terminate an argument name.	
<code>shu-capture-latex-keywd-start</code>	[Constant]
Define the latex string that is used to prepended to an argument name.	
<code>shu-capture-latex-open-quote</code>	[Constant]
Define the LaTeX string that is an open quote.	
<code>shu-capture-latex-section-end</code>	[Constant]
Define the LaTeX tag that is used to identify the start of a section heading.	
<code>shu-capture-latex-section-start</code>	[Constant]
Define the LaTeX tag that is used to identify the start of a section heading.	
<code>shu-capture-make-args-latex</code>	[Function]
<i>func-name markups</i>	

func-name is the name of the function, macro, alias, etc. *func-type* is a string that represents the function type. This will be part of the argument display. *markups* is either nil or is a cons cell that points to two lists. If *markups* is nil, the function has no arguments. If *markups* is non-nil, it is a cons cell that points to two lists. The car of *markups* is a list of the lengths of each argument before any markup was added to the argument. If an argument name is “arg1,” its length is 4 even though the length of the argument name after markup is applied may be longer. The cdr of *markups* is a list of the arguments with markup applied to them.

shu-capture-make-args-md *func-name markups* [Function]
section-converter

func-name is the name of the function, macro, alias, etc. *func-type* is a string that represents the function type. This will be part of the argument display. *markups* is either nil or is a cons cell that points to two lists. If *markups* is nil, the function has no arguments. If *markups* is non-nil, it is a cons cell that points to two lists. The car of *markups* is a list of the lengths of each argument before any markup was added to the argument. If an argument name is “arg1,” its length is 4 even though the length of the argument name after markup is applied may be longer. The cdr of *markups* is a list of the arguments with markup applied to them. *section-converter* is the function that will turn a string into a section heading.

shu-capture-make-latex-section *level hdr* [Function]

Turn *hdr* into a LaTeX section header of level *level*, where 1 is a section, 2 a subsection, etc. Return the LaTeX string.

shu-capture-make-md-section *level hdr* [Function]

Turn *hdr* into a markdown section header of level *level*, where 1 is a section, 2 a subsection, etc. Return the markdown string. If level is one (major heading), write a corresponding entry into the markdown table of contents buffer.

shu-capture-md [Command]

Capture all of the function and macro definitions in a .el source file and

turn them into markdown text that documents the functions and their doc strings.

`shu-capture-md-arg-delimiter` [Constant]
Define the markdown delimiter that is used to surround an argument name.

`shu-capture-md-buf-delimiter` [Constant]
Define the markdown delimiter that is used to surround a buffer name or any other name that has leading and trailing asterisks

`shu-capture-md-code-delimiter` [Constant]
Define the markdown delimiter that is used to surround a code snippet.

`shu-capture-md-converters` [Constant]
This is the association list of functions and strings that is used to take an elisp function and its associated doc string and convert it to markdown.

`shu-capture-md-keywd-delimiter` [Constant]
Define the markdown delimiter that is used to surround an key word such as “&optional” or “&rest”.

`shu-capture-md-quote-delimiter` [Constant]
Define the markdown delimiter that is used for open and close quote.

`shu-capture-md-section-delimiter` [Constant]
Define the markdown delimiter that is used to identify a section. This is separated from the section name by a space.

`shu-capture-pre-code-in-doc` [Constant]
The a-list key value that identifies the function that converts characters in a doc string right before the code snippets are captured.

`shu-capture-pre-code-latex` *min-point max-point* [Function]
Function that prepares a doc string to capture code snippets in LaTeX. Enter with *min-point* and *max-point* defining the region to be changed. *min-point* cannot change because all changes are made after it. But *max-point* will change if replacements add extra characters. Return the new value of *max-point* which takes into account the number of

characters added to the text.

`shu-capture-pre-code-md` *min-point max-point* [Function]
 Function that prepares a doc string to capture code snippets in mark-
 down.

`shu-capture-set-func-def` *func-def signature* [Macro]
description
 Create a func-def to describe the function

`shu-capture-set-func-def-alias` *func-def signature* [Macro]
description alias
 Create a func-def to describe the function

`shu-capture-show-list` *func-list converters buffer* [Function]

func-list is a list of function and macro definitions. *converters* is an
 a-list of functions and strings as follows:

Key	Value
---	-----
<code>shu-capture-a-type-hdr</code>	Function to format a section header
<code>shu-capture-a-type-func</code>	Function to format a function signature
<code>shu-capture-a-type-buf</code>	Function to format a buffer name
<code>shu-capture-a-type-arg</code>	Function to format an argument name
<code>shu-capture-a-type-keywd</code>	Function to format a key word
<code>shu-capture-a-type-doc-string</code>	Function to finish formatting the doc string
<code>shu-capture-a-type-enclose-doc</code>	Function to enclose doc string in begin / end
<code>shu-capture-a-type-before</code>	String that starts a block of verbatim code
<code>shu-capture-a-type-after</code>	String that ends a block of verbatim code
<code>shu-capture-a-type-open-quote</code>	String that is an open quote
<code>shu-capture-a-type-close-quote</code>	String that is a close quote

This function goes through the list and uses the *converters* to turn the set
 of function definitions into either markdown or LaTeX.

`shu-capture-show-list-md` *func-list buffer* [Function]
 Show a list

`shu-capture-toc-buffer` [Constant]

Name of the buffer into which the markdown table of contents is written

shu-capture-vars *func-list* [Function]

Find the name and doc-string for instances of “defvar” or “defconst.”

shu-doc-internal-func-to-md *func-def* [Function]

Take a function definition and turn it into a string of markdown text.

shu-doc-internal-to-md *description* [Function]

description contains a doc string from a function definition (with leading and trailing quotes removed). This function turns escaped quotes into regular (non-escaped) quotes and turns names with leading and trailing asterisks (e.g., {**project-count-buffer**}) into short code blocks surrounded by back ticks. It also turns upper case names into lower case names surrounded by markdown ticks.

shu-doc-sort-compare *lhs rhs* [Function]

Compare two function names in a sort.

7 shu-cpp-general

A collection of useful functions for dealing with C++ code.

7.1 Selected highlights

Here are some useful features of this package.

7.1.1 Dealing with long string constants

If you copy strings of text into string constants in your program, you may end up with some very long lines. *shu-csplit* can automatically split such a line for you. *shu-cunsplit* can undo the split. *shu-creplace* can in one operation, replace a split line with a different string constant.

7.2 List of functions by alias name

A list of aliases and associated function names.

author [Command]

(Function: shu-author)

Insert the doxygen author tag in an existing file.

bininclude [Command]

(Function: shu-bininclude)

If point is sitting on something that resembles a fully qualified class name, first check to see if it is in list of standard class names defined in *shu-cpp-include-names*. If it is found there, that defines the name of the defining include file. If it is not found there, then use the standard BDE algorithm to turn the class name into the name of an include file. The standard BDE algorithm replaces the `::` between namespace and class name with an underscore, makes all letters lower case, and appends “.h” to the end of the name.

Thus “abcdef::MumbleFrotz” becomes “abcdef_mumblefrotz.h”.

An include directive for the file is then created and put into the kill ring for a subsequent yank.

The file name is delimited by double quotes unless *shu-cpp-include-user-brackets* variable is true, in which case the file name is delimited by left

and right angle brackets.

Return true if a class name was found an an include generated. This is for the benefit of unit tests.

cdo (Function: shu-cdo) Insert an empty do statement.	[Command]
celse (Function: shu-celse) Insert an empty else statement.	[Command]
cfor (Function: shu-cfor) Insert an empty for statement.	[Command]
cif (Function: shu-cif) Insert an empty if statement.	[Command]
ck <i>start end</i> (Function: shu-cpp-check-streaming-op) Check a streaming operation. Mark a region that contains a set of streaming operators and invoke this function. It will make sure that you have no unterminated strings and that you are not missing any occurrences of <<.	[Command]
clc (Function: shu-clc) Place a skeleton Doxygen header definition at point.	[Command]
cpp1-class <i>class-name</i> (Function: shu-cpp1-class) Place a skeleton class definition in the current buffer at point.	[Command]
cpp2-class <i>class-name</i> (Function: shu-cpp2-class) Place a skeleton class definition in the current buffer at point.	[Command]

`creplace` *prefix* [Command]
(Function: `shu-creplace`)

This function will replace the C++ string in which point is placed with the C++ string in the kill ring. The C++ string in the kill ring is expected to be a single string with or without quotes. The C++ string in which point is placed may have been split into smaller substrings in order to avoid long lines.

Assume you have the sample string that is shown in *shu-csplit*

```
static const std::string x(“This is a very long line of text that looks “  
                           “as though it will go on forever.””);
```

You wish to replace it with a slightly different line of text, perhaps something that came from the output of a program. Copy the new string into the kill ring. Then put the cursor into any part of any line of the string to be replaced and invoke this function. This function will remove the old string, replace it with the contents of the string in the kill ring, and then split it up into shorter lines as in the following example. The string in the kill ring may have opening and closing quotes or not.

```
static const std::string x(“This is a very long line of text that looks “  
                           “as though it will go on forever and probably “  
                           “already has done so or is threatening to do “  
                           “so.””);
```

This is especially useful if you have a a string constant in a unit test and you have modified the code that creates the string. `gtest` will complain that the expected string did not match the actual string. If the actual string is correct, copy it into the kill ring, go into your unit test, find the old string, place the cursor in the old string, and replace it with the new.

`csplit` *prefix* [Command]
(Function: `shu-csplit`)

Split a C++ string into multiple strings in order to keep the line length below a certain minimum length.. The line length used is defined by the custom variable *shu-cpp-line-end*.

For example, you may copy a very long line of text into a section of code as follows:

```
static const std::string x(“This is a very long line of text that looks as though
```

To be polite to future code readers, you want to split this into multiple lines. This can be a bit cumbersome if the text is very long. This function splits the text at a somewhat arbitrary boundary so that it can be read by others whose text editors do not show code much beyond column 80 or so. This is an example of the above line after `csplit` was invoked:

```
static const std::string x(“This is a very long line of text that looks “
                           “as though it will go on forever.”);
```

This function normally splits lines on a word boundary. If any prefix argument is present, the split will be composed of fixed length lines with no respect to word boundaries.

cunsplit [Command]

(Function: `shu-cunsplit`)

The beginnings of a re-write of *shu-cunsplit*. Needs more testing. Undo the split that was done by `csplit`. Place the cursor anywhere in any of the strings and invoke this function.

cwhile [Command]

(Function: `shu-cwhile`)

Insert an empty while statement.

dbx-malloc [Command]

(Function: `shu-dbx-summarize-malloc`)

Go through the output of a `dbx malloc` dump and generate a summary. `dbx` is the AIX debugger. It has a `malloc` command that goes through the heap and prints one line for every allocated buffer. Here is a sample of some of its output:

ADDRESS	SIZE	HEAP	ALLOCATOR
0x30635678	680	0	YORKTOWN
0x30635928	680	0	YORKTOWN
0x30635bd8	680	0	HEAPCACHE
0x30635bcf	680	0	YORKTOWN

YORKTOWN is the name of the default allocator on AIX. This function goes through the malloc output and gets the number and sizes of all buffers allocated. This tells you how many buffers were allocated, the total number of bytes allocated, and the total number of buffers allocated by size. The output is placed in a separate buffer called `{**shu-aix-malloc**}`.

dcc [Command]
 (Function: shu-dcc)
 Place a skeleton Doxygen header definition at point.

dce [Command]
 (Function: shu-dce)
 Place a skeleton Doxygen header definition at point.

dciterate *type-name var-name-1 var-name-2* [Command]
 (Function: shu-dciterate)

Insert the code to iterate through a pair of data structures of type *type-name*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first prompt reads the type name, second and third prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name::const_iterator,
            type-name::const_iterator>
     its(var-name-1.begin(), var-name-2.begin());
     its.first != var-name-1.end() && its.second != var-name-2.end();
     ++its.first, ++its.second)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

diterate *type-name var-name-1 var-name-2* [Command]
 (Function: shu-diterate)

Insert the code to iterate through a pair of data structures of type *type-name*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first prompt reads the type name, second and third prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name::iterator,  
        type-name::iterator>  
    its(var-name-1.begin(), var-name-2.begin());  
    its.first != var-name-1.end() && its.second != var-name-2.end();  
    ++its.first, ++its.second)  
{  
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

dox-brief [Command]
(Function: `shu-dox-brief`)
Place a skeleton Doxygen header definition at point.

dox-cbt [Command]
(Function: `shu-dox-cbt`)
Convert a section of comments delimited by `//!` into Doxygen brief format.

dox-cvt [Command]
(Function: `shu-dox-cvt`)
Convert a section of comments delimited by `//` into Doxygen format.

dox2-hdr [Command]
(Function: `shu-dox2-hdr`)
Place a skeleton Doxygen header definition at point.

drc [Command]
(Function: `shu-drc`)

Place a skeleton Doxygen header definition at point.

fill-area *start end* [Command]

(Function: shu-cpp-fill-test-area)

For all lines between the marked start and end points, if a recognized data type has been declared on a line, fill it with random test data.

For the benefit of unit tests, this function returns a cons cell whose car is the number of unrecognized data types and whose cdr is the number of values generated.

fill-data [Command]

(Function: shu-cpp-fill-test-data)

If the data type at the beginning of the line is a recognized data type, then fill in a random value for that data type at point. This allows someone writing a test to declare a data type and a name and then call this function. If the author creates a line that looks like this and then invokes this function

```
std::string  abc
```

The line will be transformed into one that looks something like this:

```
std::string  abc('RDATZC');
```

The recognized data types are the ones that are defined by the custom variables shu-cpp-date-type, shu-cpp-datetime-timezone-type, shu-cpp-datetime-type, shu-cpp-interval-type, shu-cpp-long-long-type, shu-cpp-string-type, or shu-cpp-time-type plus many of the standard C++ types, such as int, bool, short, etc.

The data types may optionally be preceded by “const”.

If the last character of the line is “;”, it is deleted before a data type is filled in with a new semi-colon following it.

fixp [Command]

(Function: shu-cpp-fix-prototype)

Place the cursor on the beginning of a function declaration that has been copied from a .cpp file to a .h file. This function fixes up the

function prototype to make it suitable for a .h file. For example, this declaration:

```
double Frobnitz::hitRatio(  
    const int  reads,  
    const int  writes)  
const
```

would be transformed into

```
double hitRatio(  
    const int  reads,  
    const int  writes)  
const;
```

gcc [Command]

(Function: shu-gcc)

Get compile command command from current buffer. While in a compile buffer, go to the top of the buffer, search for the end of the prompt line, collect the rest of the line and put it into the kill ring. This takes the string that was used for the last compile command in the current buffer and puts it into the kill ring. To compile again with the same command, kill the buffer, open a new shell, and yank.

get-set [Command]

(Function: shu-get-set)

Generate get and set functions for an instance variable in a C++ class. Position the cursor ahead of the Doxygen comment above the variable. The get and set functions will be placed in the buffer `{*get-set*}`.

getdef [Command]

(Function: shu-cpp-find-h-definition)

While in a cpp file, position point on a variable name that is defined in the corresponding header file and invoke this function. It will find all occurrences of the name in the header file and put them in the message area.

getters *start end* [Command]

(Function: shu-getters)

Mark a region in a file that contains C++ instance variable declarations. This function will create get and set functions for all of the instance variables.

ginclude [Command]

(Function: shu-ginclude)

While in a file buffer, wrap the file name in a C++ include directive and put it in the kill ring. The file name is delimited by double quotes unless *shu-cpp-include-user-brackets* variable is true, in which case the file name is delimited by left and right angle brackets.

make-datetime [Command]

(Function: shu-cpp-make-datetime)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

make-interval [Command]

(Function: shu-cpp-make-interval)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

make-tzdate [Command]

(Function: shu-cpp-tz-make-datetime)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

new-c-class [Command]

(Function: shu-new-c-class)

Place a skeleton class definition in the current buffer at point.

new-c-file [Command]

(Function: shu-new-c-file)

Generate a skeleton code file for a C or C++ file.

new-h-file [Command]

(Function: shu-new-h-file)

Generate a skeleton header file for C or C++ file.

new-x-file [Command]

(Function: shu-new-x-file)
Generate a skeleton Doxygen
file directive.

operators *class-name* [Command]

(Function: shu-operators)
Place skeletons of all of the standard c++ operator functions at point.

qualify-bsl [Command]

(Function: shu-qualify-namespace-bsl)
Add “bsl” namespace qualifier to some of the classes in “bsl”. Return
the count of class names changed.

qualify-class [Command]

(Function: shu-interactive-qualify-class-name)
Interactively call *shu-qualify-class-name* to find all instances of a class
name and add a namespace qualifier to it. First prompt is for the
class name. If a fully qualified class name is supplied, then the given
namespace is applied to the class name. If the name supplied is not
a namespace qualified class name, then a second prompt is given to
read the namespace. This is intended to help rescue code that has one
or more “using namespace” directives in it. The problem with “using
namespace” is that you now have class names from other namespaces
with no easy way to identify the namespace to which they belong. The
best thing to do is get rid of the “using namespace” statements and
explicitly qualify the class names. But if you use a simple replace to
do that, you will qualify variable names that resemble class names as
well as class names that are already qualified. This function only adds
a namespace to a class name that does not already have a namespace
qualifier.

qualify-std [Command]

(Function: shu-qualify-namespace-std)
Add “std” namespace qualifier to some of the classes in “std”. Return
the count of class names changed.

set-default-namespace *name* [Command]
(Function: shu-set-default-namespace)
Set the local namespace for C++ classes.

set-modern [Command]
(Function: shu-set-modern)
Unconditionally set shu-cpp-modern to true.

set-no-modern [Command]
(Function: shu-set-no-modern)
Unconditionally set shu-cpp-modern to false.

sort-includes [Command]
(Function: shu-sort-includes)
When positioned on a line that is an `#include` directive, find all of the `#include` directives above and below that line that are not separated by a blank line and sort them into alphabetical order with case ignored. If not positioned on a line that is an `#include` directive, do nothing. Case is ignored for the sort. The return value is the number of lines sorted. If no lines were sorted because (point) is not positioned on an `#include` directive, return nil. The return value is for the benefit of the unit tests. Additionally, if there are spaces surrounding the “`#`” of the `#include`, they are removed.

tciterate *type-name-1 type-name-2 var-name-1* [Command]
(Function: shu-tciterate)
Insert the code to iterate through a pair of data structures of types *type-name-1* and *type-name-2*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.
The first two prompt reads the two type names, third and fourth prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name-1::const_iterator,  
        type-name-2::const_iterator>  
    its(var-name-1.begin(), var-name-2.begin());  
    its.first != var-name-1.end() && its.second != var-name-2.end();  
    ++its.first, ++its.second)
```

```
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

titerate *type-name-1 type-name-2 var-name-1* [Command]
(Function: `shu-titerate`)

Insert the code to iterate through a pair of data structures of types *type-name-1* and *type-name-2*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first two prompt reads the two type names, third and fourth prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name-1::iterator,
            type-name-2::iterator>
     its(var-name-1.begin(), var-name-2.begin());
     its.first != var-name-1.end() && its.second != var-name-2.end();
     ++its.first, ++its.second)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

to-camel [Command]
(Function: `shu-to-camel`)

Convert the variable name at point from snake case to camel case.

For example, “`mumble_something_other`” becomes “`mumbleSomethingOther`”.

to-snake [Command]
(Function: `shu-to-snake`)

Convert the variable name at point from camel case to snake case.

For example, “mumbleSomethingOther” becomes “mumble_something_other”.

7.3 List of functions and variables

List of functions and variable definitions in this package.

shu-add-cpp-base-types *ntypes* [Function]

Add one or more data types to the list of C++ native data types defined in `shu-cpp-base-types` in `shu-cpp-general.el`. Argument may be a single type in a string or a list of strings. This modifies `shu-cpp-base-types`.

shu-aix-show-allocators *sizes gb* [Function]

sizes is an alist whose car is an allocator name and whose cdr is the number of allocations attributed to that allocator. For each allocator, display in the buffer *gb*, the name of the allocator and its counts

shu-aix-show-malloc-list *mlist gb* [Function]

Print the number of buffers allocated by size from an AIX dbx malloc command.

shu-attr-name [Variable]

The name of an attribute.

shu-author [Command]

(Alias: author)

Insert the doxygen author tag in an existing file.

shu-binclude [Command]

(Alias: binclude)

If point is sitting on something that resembles a fully qualified class name, first check to see if it is in list of standard class names defined in *shu-cpp-include-names*. If it is found there, that defines the name of the defining include file. If it is not found there, then use the standard BDE algorithm to turn the class name into the name of an include file. The standard BDE algorithm replaces the `::` between namespace and class name with an underscore, makes all letters lower case, and appends “.h”

to the end of the name.

Thus “abcdef::MumbleFrotz” becomes “abcdef_mumblefrotz.h”.

An include directive for the file is then created and put into the kill ring for a subsequent yank.

The file name is delimited by double quotes unless *shu-cpp-include-user-brackets* variable is true, in which case the file name is delimited by left and right angle brackets.

Return true if a class name was found an an include generated. This is for the benefit of unit tests.

shu-bsl-include-list [Constant]

An alist that maps include file names to class names when using BDE.

shu-cciterate *type-name var-name* [Command]

Insert the code to iterate through a data structure of type *type-name* whose instance is identified by *var-name*. First prompt reads the type name. Second prompt read the variable name.

The generated code sequence is as follows:

```
for (type_name::const_iterator it = var_name.begin();
    it != var_name.end(); ++it)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable *shu-cpp-indent-length*.

shu-cdo [Command]

(Alias: cdo)

Insert an empty do statement.

shu-celse [Command]

(Alias: celse)

Insert an empty else statement.

shu-cfor [Command]

(Alias: cfor)

Insert an empty for statement.

shu-cif [Command]

(Alias: cif)

Insert an empty if statement.

shu-citerate *type-name var-name* [Command]

Insert the code to iterate through a data structure of type *type-name* whose instance is identified by *var-name*. First prompt reads the type name. Second prompt read the variable name.

The generated code sequence is as follows:

```
for (type_name::iterator it = var_name.begin();
    it != var_name.end(); ++it)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

shu-class-is-blocked *pos &optional in-string* [Function]

Return true if a class name should be ignored because it is either in a string or a comment.

We have found something at point *pos* that looks as though it might be a class name. If it is in a string or is preceded on the same line by “//” (also not in a string), then it is either in a string or is probably in a comment, so we may want to ignore it. *in-string* is true if a class name inside of a string is to be replaced. *in-comment* is true if a class name inside of a comment is to be replaced.

Return true if the class name should be ignored.

shu-clc [Command]

(Alias: clc)

Place a skeleton Doxygen header definition at point.

shu-cpp-allocator-type [Custom]

The data type that represents an allocator.

shu-cpp-base-types [Constant]

A list of all of the base types in C and C++. This may be modified by `shu-add-cpp-base-types`

shu-cpp-check-streaming-op *start end* [Command]
(Alias: ck)

Check a streaming operation. Mark a region that contains a set of streaming operators and invoke this function. It will make sure that you have no unterminated strings and that you are not missing any occurrences of `<<`.

shu-cpp-date-type [Custom]

The data type that represents a date.

shu-cpp-datetime-timezone-type [Custom]

The data type that represents a date and time with an associated time zone.

shu-cpp-datetime-type [Custom]

The data type that represents a date and time.

shu-cpp-fill-test-area *start end* [Command]
(Alias: fill-area)

For all lines between the marked start and end points, if a recognized data type has been declared on a line, fill it with random test data.

For the benefit of unit tests, this function returns a cons cell whose car is the number of unrecognized data types and whose cdr is the number of values generated.

shu-cpp-fill-test-data [Command]
(Alias: fill-data)

If the data type at the beginning of the line is a recognized data type, then fill in a random value for that data type at point. This allows someone writing a test to declare a data type and a name and then call this function. If the author creates a line that looks like this and then invokes this function

```
std::string  abc
```

The line will be transformed into one that looks something like this:

```
std::string    abc('‘RDATZC’');
```

The recognized data types are the ones that are defined by the custom variables `shu-cpp-date-type`, `shu-cpp-datetime-timezone-type`, `shu-cpp-datetime-type`, `shu-cpp-interval-type`, `shu-cpp-long-long-type`, `shu-cpp-string-type`, or `shu-cpp-time-type` plus many of the standard C++ types, such as `int`, `bool`, `short`, etc.

The data types may optionally be preceded by “`const`”.

If the last character of the line is “`;`”, it is deleted before a data type is filled in with a new semi-colon following it.

shu-cpp-find-h-definition [Command]
(Alias: `getdef`)

While in a `cpp` file, position point on a variable name that is defined in the corresponding header file and invoke this function. It will find all occurrences of the name in the header file and put them in the message area.

shu-cpp-find-using &optional *top-name* [Command]

Return the name of the class found on the next “`using namespace`” directive or nil if no such directive found.

top-name, if present is a higher level namespace. Given a top level namespace of “`WhammoCorp`”, then the following line:

```
using namespace WhammoCorp::world;
```

would be interpreted as though it had been written:

```
using namespace world;
```

shu-cpp-find-variable-name-by-token *var-name* [Function]

Tokenize the entire buffer and return the position of the first token that matches *var-name*.

shu-cpp-find-variable-name-lines-by-token *var-name* [Function]

Tokenize the entire buffer and return a string that is composed of each line that contains the token.

shu-cpp-fix-prototype [Command]
(Alias: fixp)

Place the cursor on the beginning of a function declaration that has been copied from a .cpp file to a .h file. This function fixes up the function prototype to make it suitable for a .h file. For example, this declaration:

```
double Frobnitz::hitRatio(  
    const int  reads,  
    const int  writes)  
const
```

would be transformed into

```
double hitRatio(  
    const int  reads,  
    const int  writes)  
const;
```

shu-cpp-general-set-alias [Function]

Set the common alias names for the functions in shu-cpp-general. These are generally the same as the function names with the leading shu- prefix removed.

shu-cpp-get-variable-name [Function]

If point is sitting on something that looks like a legal variable name, return it, otherwise, return nil.

shu-cpp-get-variable-name-position [Function]

If point is sitting on something that looks like a legal variable name, return a cons cell that contains the start and end positions of the name otherwise, return nil.

shu-cpp-include-names [Variable]

A hash table that maps class names to include file names This is the hash table inversion of shu-std-include-list or shu-bsl-include-list.

shu-cpp-internal-fill-test-data [Function]

If the data type at the beginning of the line is a recognized data type, then fill in a random value for that data type at point. This allows someone writing a test to declare a data type and a name and then call this function. If the author creates a line that looks like this and then invokes this function

```
std::string    abc
```

The line will be transformed into one that looks something like this:

```
std::string    abc('RDATZC');
```

The recognized data types are the ones that are defined by the custom variables shu-cpp-date-type, shu-cpp-datetime-timezone-type, shu-cpp-datetime-type, shu-cpp-interval-type, shu-cpp-long-long-type, shu-cpp-string-type, or shu-cpp-time-type plus many of the standard C++ types, such as int, bool, short, etc.

The data types may optionally be preceded by “const”.

If the last character of the line is “;”, it is deleted before a data type is filled in with a new semi-colon following it.

Return t if a recognized data type was found and a value was filled in.

shu-cpp-internal-make-bool [Function]

Return value for a bool type.

shu-cpp-internal-make-char [Command]

Return a string that can be used to initialize a test variable of type int.

shu-cpp-internal-make-date [Function]

Return a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

shu-cpp-internal-make-datetime [Function]

Return a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

shu-cpp-internal-make-double [Command]
Return a string that can be used to initialize a test variable of type double.

shu-cpp-internal-make-float [Command]
Return a string that can be used to initialize a test variable of type float.

shu-cpp-internal-make-int [Command]
Return a string that can be used to initialize a test variable of type lint.

shu-cpp-internal-make-interval [Function]
Return a string that is the list of values to be passed to the constructor of a datetime type that accepts days, hours, minutes, seconds, milliseconds, microseconds.

shu-cpp-internal-make-long-long [Command]
Return a string that can be used to initialize a test variable of type long long.

shu-cpp-internal-make-short [Command]
Return a string that can be used to initialize a test variable of type short.

shu-cpp-internal-make-short-interval [Function]
Return a string that is the list of values to be passed to the constructor of a datetime type that accepts seconds and nanoseconds.

shu-cpp-internal-make-time [Function]
Return a string that is the list of values to be passed to the constructor of a time type that accepts hour, minute, second, milliseconds, microseconds.

shu-cpp-internal-make-unsigned-int [Command]

Return a string that can be used to initialize a test variable of type `lint`.

shu-cpp-internal-stream-check *token-list* [Function]

Take a list of tokens found in a C++ streaming operation and check to ensure that every other token is a `<<` operator. Two adjacent occurrences of `<<` represent an extraneous `<<` operator. Two adjacent occurrences of tokens that are not `<<` represent a missing `<<` operator.

shu-cpp-internal-tz-make-datetime [Function]

Return a string that is the list of values to be passed to the constructor of a timezone datetime type.

shu-cpp-interval-type [Custom]

The data type that represents a time interval type.

shu-cpp-is-enclosing-op *op* [Function]

Return true if the single character in *op* is an enclosing character, a left or right parenthesis or a left or right square bracket.

shu-cpp-long-long-type [Custom]

The data type that represents a 64 bit integer.

shu-cpp-make-date [Command]

insert a string that is the list of values to be passed to the constructor of a date type that accepts year, month, day.

shu-cpp-make-datetime [Command]

(Alias: `make-datetime`)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

shu-cpp-make-interval [Command]

(Alias: `make-interval`)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

- shu-cpp-make-short-interval** [Command]
 insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.
- shu-cpp-make-size-type** [Function]
 insert a string that is a possible value for a size type.
- shu-cpp-make-time** [Command]
 insert a string that is the list of values to be passed to the constructor of a time type that accepts hour, minute, second, milliseconds, microseconds.
- shu-cpp-map-class-to-include** *class-name* [Function]
class-name is a fully qualified class name (std::string as an example). This function returns the name of the include file that defines the class, if known.
- shu-cpp-member-prefix** [Variable]
 The character string that is used as the prefix to member variables of a C++ class. This is used by shu-internal-get-set when generating getters and setters for a class.
- shu-cpp-qualify-classes** *class-list namespace* [Function]
buffer
 Repeatedly call *shu-qualify-class-name* for all class names in *class-list*. *namespace* is either the name of a single namespace to apply to all classes in *class-list* or is a list of namespaces each of which has a one to one correspondence with a class name in *class-list*. The optional *buffer* argument may be a buffer in which the actions are recorded. Return the number of names changed.
- shu-cpp-rmv-blocked** *class-list top-name gb* [Function]
 Do a pre-check on a file to see if we will be able to remove its “using namespace” directives. *class-list* is the a-list passed to *shu-cpp-rmv-using*. USING is the regular expression used to search for “using namespace” directives. TOP-QUAL is the regular expression used to strip out a higher level qualifier from the class name in a “using namespace” directive, if any. *gb* is the buffer into which diagnostic messages

are written.

This function finds all of the “using namespace” directives in the file and checks to see if there is any ambiguity in the resulting class list. For example, if namespace “mumble” contains class “Bumble” and namespace “stubble” also contains class “Bumble”, we will not know which namespace to apply to instances of class “Bumble”. But this is not an ambiguity if there is a “using namespace” directive for only one of those classes. That is why we do the ambiguity check only for namespaces referenced by “using namespace” directives.

This function returns true if such an ambiguity exists.

shu-cpp-rmv-using-old *class-list* &**optional** [Function]

Remove “using namespace” directives from a C++ file, adding the appropriate namespace qualifier to all of the unqualified class names. *class-list* is an a-list in which the car of each entry is a namespace and the cdr of each entry is a list of class names. Here is an example of such an a-list:

```
(list
  (cons 'std' (list 'set' 'string' 'vector'))
  (cons 'world' (list 'Hello' 'Goodbye'))))
```

top-name, if present is a higher level namespace. Given a top level namespace of “WhammoCorp”, then the following line:

```
using namespace WhammoCorp::world;
```

would be interpreted as though it had been written:

```
using namespace world;
```

NB: This version is deprecated. See the new version in shu-match.el

shu-cpp-short-interval-type [Custom]

The data type that represents a short time interval type.

shu-cpp-size-type [Custom]

The data type that represents a size.

shu-cpp-string-type [Custom]

The data type that represents a string type.

shu-cpp-time-type [Custom]

The data type that represents a time.

shu-cpp-tz-make-datetime [Command]

(Alias: make-tzdate)

insert a string that is the list of values to be passed to the constructor of a datetime type that accepts year, month, day, hour, minute, second, milliseconds, microseconds.

shu-cpp1-class *class-name* [Command]

(Alias: cpp1-class)

Place a skeleton class definition in the current buffer at point.

shu-cpp2-class *class-name* [Command]

(Alias: cpp2-class)

Place a skeleton class definition in the current buffer at point.

shu-creplace *prefix* [Command]

(Alias: creplace)

This function will replace the C++ string in which point is placed with the C++ string in the kill ring. The C++ string in the kill ring is expected to be a single string with or without quotes. The C++ string in which point is placed may have been split into smaller substrings in order to avoid long lines.

Assume you have the sample string that is shown in *shu-csplit*

```
static const std::string x(“This is a very long line of text that looks “  
“as though it will go on forever.””);
```

You wish to replace it with a slightly different line of text, perhaps something that came from the output of a program. Copy the new string into the kill ring. Then put the cursor into any part of any line of the string to be replaced and invoke this function. This function will remove the old string, replace

it with the contents of the string in the kill ring, and then split it up into shorter lines as in the following example. The string in the kill ring may have opening and closing quotes or not.

```
static const std::string x(“This is a very long line of text that looks “  
    “as though it will go on forever and probably “  
    “already has done so or is threatening to do “  
    “so.””);
```

This is especially useful if you have a a string constant in a unit test and you have modified the code that creates the string. gtest will complain that the expected string did not match the actual string. If the actual string is correct, copy it into the kill ring, go into your unit test, find the old string, place the cursor in the old string, and replace it with the new.

shu-csplit *prefix* [Command]
(Alias: csplit)

Split a C++ string into multiple strings in order to keep the line length below a certain minimum length.. The line length used is defined by the custom variable *shu-cpp-line-end*.

For example, you may copy a very long line of text into a section of code as follows:

```
static const std::string x(“This is a very long line of text that looks as though
```

To be polite to future code readers, you want to split this into multiple lines. This can be a bit cumbersome if the text is very long. This function splits the text at a somewhat arbitrary boundary so that it can be read by others whose text editors do not show code much beyond column 80 or so. This is an example of the above line after csplit was invoked:

```
static const std::string x(“This is a very long line of text that looks “  
    “as though it will go on forever.””);
```

This function normally splits lines on a word boundary. If any prefix argument is present, the split will be composed of fixed length lines with no respect to word boundaries.

shu-cunsplit [Command]

(Alias: cunsplit)

The beginnings of a re-write of *shu-cunsplit*. Needs more testing. Undo the split that was done by *csplit*. Place the cursor anywhere in any of the strings and invoke this function.

shu-cwhile [Command]

(Alias: cwhile)

Insert an empty while statement.

shu-dbx-summarize-malloc [Command]

(Alias: dbx-malloc)

Go through the output of a dbx malloc dump and generate a summary. dbx is the AIX debugger. It has a malloc command that goes through the heap and prints one line for every allocated buffer. Here is a sample of some of its output:

ADDRESS	SIZE	HEAP	ALLOCATOR
0x30635678	680	0	YORKTOWN
0x30635928	680	0	YORKTOWN
0x30635bd8	680	0	HEAPCACHE
0x30635bcf	680	0	YORKTOWN

YORKTOWN is the name of the default allocator on AIX. This function goes through the malloc output and gets the number and sizes of all buffers allocated. This tells you how many buffers were allocated, the total number of bytes allocated, and the total number of buffers allocated by size. The output is placed in a separate buffer called `{**shu-aix-malloc**}`.

shu-dcc [Command]

(Alias: dcc)

Place a skeleton Doxygen header definition at point.

shu-dce [Command]

(Alias: dce)

Place a skeleton Doxygen header definition at point.

shu-dciterate *type-name var-name-1 var-name-2* [Command]

(Alias: dciterate)

Insert the code to iterate through a pair of data structures of type *type-name*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first prompt reads the type name, second and third prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name::const_iterator,  
        type-name::const_iterator>  
    its(var-name-1.begin(), var-name-2.begin());  
    its.first != var-name-1.end() && its.second != var-name-2.end();  
    ++its.first, ++its.second)  
{  
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

shu-diterate *type-name var-name-1 var-name-2* [Command]
(Alias: `diterate`)

Insert the code to iterate through a pair of data structures of type *type-name*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first prompt reads the type name, second and third prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name::iterator,  
        type-name::iterator>  
    its(var-name-1.begin(), var-name-2.begin());  
    its.first != var-name-1.end() && its.second != var-name-2.end();  
    ++its.first, ++its.second)  
{  
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

shu-dox-brief [Command]
(Alias: `dox-brief`)

Place a skeleton Doxygen header definition at point.

shu-dox-cbt [Command]
(Alias: `dox-cbt`)

Convert a section of comments delimited by `//!` into Doxygen brief format.

shu-dox-cvt [Command]
(Alias: `dox-cvt`)

Convert a section of comments delimited by `//` into Doxygen format.

shu-dox-hdr [Command]

Place a skeleton Doxygen header definition at point.

shu-dox2-hdr [Command]
(Alias: `dox2-hdr`)

Place a skeleton Doxygen header definition at point.

shu-drc [Command]
(Alias: `drc`)

Place a skeleton Doxygen header definition at point.

shu-emit-get [Function]

Undocumented

shu-emit-set *arg* [Function]

Undocumented

shu-find-end-include *dir* [Function]

On entry, (point) is positioned on an `#include` directive. This function searches either backwards or forward for the next line that does not contain an include directive. If *dir* is positive the search is forward, else

backward. The return value is the (line-beginning-position) of the last line that contains an `#include` directive.

shu-gcc [Command]
(Alias: gcc)

Get compile command command from current buffer. While in a compile buffer, go to the top of the buffer, search for the end of the prompt line, collect the rest of the line and put it into the kill ring. This takes the string that was used for the last compile command in the current buffer and puts it into the kill ring. To compile again with the same command, kill the buffer, open a new shell, and yank.

shu-gen-return-ptr [Function]
Undocumented

shu-get-set [Command]
(Alias: get-set)

Generate get and set functions for an instance variable in a C++ class. Position the cursor ahead of the Doxygen comment above the variable. The get and set functions will be placed in the buffer `{*get-set*}`.

shu-getters *start end* [Command]
(Alias: getters)

Mark a region in a file that contains C++ instance variable declarations. This function will create get and set functions for all of the instance variables.

shu-ginclude [Command]
(Alias: ginclude)

While in a file buffer, wrap the file name in a C++ include directive and put it in the kill ring. The file name is delimited by double quotes unless *shu-cpp-include-user-brackets* variable is true, in which case the file name is delimited by left and right angle brackets.

shu-interactive-qualify-class-name [Command]
(Alias: qualify-class)

Interactively call *shu-qualify-class-name* to find all instances of a class name and add a namespace qualifier to it. First prompt is for the class name. If a fully qualified class name is supplied, then the given

namespace is applied to the class name. If the name supplied is not a namespace qualified class name, then a second prompt is given to read the namespace. This is intended to help rescue code that has one or more “using namespace” directives in it. The problem with “using namespace” is that you now have class names from other namespaces with no easy way to identify the namespace to which they belong. The best thing to do is get rid of the “using namespace” statements and explicitly qualify the class names. But if you use a simple replace to do that, you will qualify variable names that resemble class names as well as class names that are already qualified. This function only adds a namespace to a class name that does not already have a namespace qualifier.

shu-internal-citerate *type-name var-name &optional* [Function]

Insert the code to iterate through a data structure of type *type-name* whose instance is identified by *var-name*. First prompt reads the variable name. Second prompt read the variable name.

The generated code sequence is as follows:

```
for (type_name::iterator it = var_name.begin();
    it != var_name.end(); ++it)
{
}
```

If optional *const* is true, a const iterator is generated.

shu-internal-cpp2-class *class-name* [Function]

Place a skeleton class definition in the current buffer at point.

shu-internal-creplace **&optional** *fixed-width* [Function]

This is the internal implementation of *shu-creplace*.

shu-internal-csplit **&optional** *fixed-width* [Function]

This is the internal implementation of *shu-csplit*.

shu-internal-double-citerate *type-name-1* [Function]
var-name-1 var-name-2

con

Insert the code to iterate through a pair of data structures of types *type-name-1* and *type-name-2*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The generated code sequence is as follows:

```

for (std::pair<type-name-1::const_iterator,
            type-name-2::const_iterator>
     its(var-name-1.begin(), var-name-2.begin());
     its.first != var-name-1.end() && its.second != var-name-2.end();
     ++its.first, ++its.second)
{
}

```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

If optional *const* is true, a const iterator is generated.

shu-internal-get-set *comment shu-lc-comment* [Command]
 Generate get and set functions for an instance variable in a C++ class.

shu-internal-replace-class-name *target-name* [Function]
replace-arg

string

in-

Find all instances of the class name *target-name* and if it actually appears to be a class name, call REPLACE-FUN passing to it *replace-arg* and the class name. REPLACE-FUN issues the appropriate replace-match call, constructing the replacement for the class name from some combination of *replace-arg* and the class name. *in-string* is true if a class name inside of a string is to be replaced. *in-comment* is true if a class name inside of a comment is to be replaced.

shu-is-const [Variable]
 Set true if the C++ data member we are working is declared to be const.

shu-lc-comment [Variable]

Comment string with the first letter downcased.

shu-line-is-include *&optional fix* [Function]

Return true if the line on which (point) resides is an `#include` directive.

shu-make-padded-line *line tlen* [Function]

Add sufficient spaces to make *line* the length *tlen*.

shu-nc-vtype [Variable]

Set true if the C++ data member we are working is declared to be non-const.

shu-new-c-class [Command]

(Alias: new-c-class)

Place a skeleton class definition in the current buffer at point.

shu-new-c-file [Command]

(Alias: new-c-file)

Generate a skeleton code file for a C or C++ file.

shu-new-deallocate *var-name* [Command]

Insert the code to do a standard deallocation of memory allocated by a specific allocator. First prompt reads the variable name that points to the memory to be deallocated. Second prompt reads the name of the class whose destructor is to be called.

This generates a code sequence as follows:

```
if (var-name)
{
    m_allocator->deleteObject(var-name);
    var-name = 0;
}
```

If *shu-cpp-modern* is true, the code sequence is:

```
if (var-name != nullptr)
{
    m_allocator->deleteObject(var-name);
    var-name = nullptr;
}
```

var-name is read from a prompt. The number of spaces to indent inside that braces is defined in the custom variable `shu-cpp-indent-length`. The name of the member variable that points to the allocator in use by the class comes from the custom variable `shu-cpp-default-allocator-name`

shu-new-h-file [Command]
(Alias: new-h-file)
Generate a skeleton header file for C or C++ file.

shu-new-x-file [Command]
(Alias: new-x-file)
Generate a skeleton Doxygen file directive.

shu-operators *class-name* [Command]
(Alias: operators)
Place skeletons of all of the standard c++ operator functions at point.

shu-qualify-class-fun *namespace name* [Function]
This is the replacement function for *shu-qualify-class-name*. It is called with the *namespace* to be applied to the class whose name is *name*. It constructs a new name and issues `replace-match` to replace it.

shu-qualify-class-name *target-name namespace* [Function]
Find all instances of the class name *target-name* and add an explicit namespace qualifier *namespace*. If the *target-name* is “Mumble” and the *namespace* is “abcd”, then “Mumble” becomes “abcd::Mumble”. But variable names such as “d.Mumble” or “MumbleIn” remain unchanged and already qualified class names remain unchanged. This is intended to help rescue code that has one or more “using namespace” directives in it. The problem with “using namespace” is that you now have class names from other namespaces with no easy way to identify the namespace to which they belong. The best thing to do is get rid of the “using namespace” statements and explicitly qualify the class names. But if you use a simple replace to do that, you will qualify variable names that resemble class names as well as class names that are already qualified. This function only adds a namespace to a class name that does not already have a namespace qualifier.

- shu-qualify-namespace-bsl** [Command]
 (Alias: qualify-bsl)
 Add “bsl” namespace qualifier to some of the classes in “bsl”. Return the count of class names changed.
- shu-qualify-namespace-std** [Command]
 (Alias: qualify-std)
 Add “std” namespace qualifier to some of the classes in “std”. Return the count of class names changed.
- shu-replace-class-fun** *new-name name* [Function]
 This is the replacement function for *shu-replace-class-name*. It is called with the *new-name* to replace the class *name*. It calls *replace-match* to replace *name* with *new-name*.
- shu-replace-class-name** *target-name new-name* [Function]
in-string in-comment
 Find all instances of the class name *target-name* and replace it with the name *new-name*. If the target name is “Mumble”, then all instances of “Mumble” that resemble class names are replaced. But names such as “d_Mumble” or “MumbleIn” remain unchanged. if *in-string* is true, then instances of the class name found inside a string are replaced. if *in-comment* is true, then instances of the class name found inside a comment are replaced.
- shu-return-ptr** [Function]
 Undocumented
- shu-return-ref** [Function]
 Undocumented
- shu-rmv-classes** [Variable]
 An alist of “using namespace” directives and their line numbers where first declared. Used to filter duplicates.
- shu-s-mode-find-long-line** [Command]
 Place point in column 79 of the next line whose length exceeds 79 characters. No movement occurs if no lines, starting with the current position, exceed 79 characters in length.

shu-set-author *name* [Command]
 Set the author name to be placed in generated C++ classes.

shu-set-default-global-namespace *name* [Command]
 Set the global namespace for C++ classes.

shu-set-default-namespace *name* [Command]
 (Alias: set-default-namespace)
 Set the local namespace for C++ classes.

shu-set-modern [Command]
 (Alias: set-modern)
 Unconditionally set shu-cpp-modern to true.

shu-set-no-modern [Command]
 (Alias: set-no-modern)
 Unconditionally set shu-cpp-modern to false.

shu-set-obj [Function]
 Undocumented

shu-set-ptr [Function]
 Undocumented

shu-simple-hother-file [Function]
 Return the name of the .h file that corresponds to the .cpp file or .t.cpp file that is in the current buffer. This version of the function creates the name of the .h file from the name of the file in the current buffer. This is in contrast with the function shu-hother which finds the corresponding .h file from the list of files in the current project.

shu-sort-includes [Command]
 (Alias: sort-includes)
 When positioned on a line that is an #include directive, find all of the #include directives above and below that line that are not separated by a blank line and sort them into alphabetical order with case ignored. If not positioned on a line that is an #include directive, do nothing. Case is ignored for the sort. The return value is the number of lines sorted. If no lines were sorted because (point) is not positioned on an #include

directive, return nil. The return value is for the benefit of the unit tests. Additionally, if there are spaces surrounding the “#” of the #include, they are removed.

shu-std-include-list [Constant]

An alist that maps include file names to class names.

shu-tciterate *type-name-1 type-name-2 var-name-1* [Command]

(Alias: tciterate)

Insert the code to iterate through a pair of data structures of types *type-name-1* and *type-name-2*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first two prompt reads the two type names, third and fourth prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name-1::const_iterator,
            type-name-2::const_iterator>
     its(var-name-1.begin(), var-name-2.begin());
     its.first != var-name-1.end() && its.second != var-name-2.end();
     ++its.first, ++its.second)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable shu-cpp-indent-length.

The name of the namespace used for the standard library is defined in the custom variable shu-cpp-std-namespace.

shu-titerate *type-name-1 type-name-2 var-name-1* [Command]

(Alias: titerate)

Insert the code to iterate through a pair of data structures of types *type-name-1* and *type-name-2*, whose first instance is identified by *var-name-1* and whose second instance is identified by *var-name-2*.

The first two prompt reads the two type names, third and fourth prompts read the two variable names.

The generated code sequence is as follows:

```
for (std::pair<type-name-1::iterator,
            type-name-2::iterator>
     its(var-name-1.begin(), var-name-2.begin());
     its.first != var-name-1.end() && its.second != var-name-2.end();
     ++its.first, ++its.second)
{
}
```

The number of spaces to indent inside the braces is defined in the custom variable `shu-cpp-indent-length`.

The name of the namespace used for the standard library is defined in the custom variable `shu-cpp-std-namespace`.

shu-to-camel [Command]

(Alias: to-camel)

Convert the variable name at point from snake case to camel case.

For example, “mumble_something_other” becomes “mumbleSomethingOther”.

shu-to-snake [Command]

(Alias: to-snake)

Convert the variable name at point from camel case to snake case.

For example, “mumbleSomethingOther” becomes “mumble_something_other”.

shu-var-name [Variable]

The variable name that corresponds to an attribute name.

8 shu-cpp-match

Functions to match patterns against list of tokens produced by shu-cpp-token.el.

The functions in shu-cpp-token.el can scan a section of C++ source code and turn it into a list of tokens. Each token has a type (comment, string, keyword, operator or unquoted token) and a value, which is the token itself. Each token also contains its start and end position within the file.

The functions in this file, shu-cpp-match.el, allow one to use data structures to describe patterns to be found in a list of tokens.

The simplest data structure consists of a list of match items that must exactly match a list of tokens. If you want to find something that looks like

```
pointer->thing
```

you put together a list of three match items. The first specifies a regular expression that can match a C++ name. The second is an exact match for the operator "->", and the third is another regular expression that can match a C++ name.

You can also search for more than one pattern at a time by supplying a list of lists of match items. Suppose you want to search for an occurrence of a "using namespace" directive. You might have one list that matches "using namespace name;", another list that matches "using namespace ::name;", and another list that matches "using namespace component::name";.

If you try to match this set of three lists against the following string

```
using namespace thing::Bob;
```

The three lists are evaluated as follows:

The first list matches "using," "namespace," "thing," but fails when it does not find a semi-colon following "thing."

The second list matches "using," "namespace," but fails when it does not find an operator "::".

The third list matches "using," "namespace," "thing," "::," "Bob," and ";".

With one function call, you have found a reasonably complex pattern. The tokens scanned do not include comments. This means that the above example would have worked identically on a list of tokens derived from

```
using /* Hello \emph{*/} namespace
// Something here
thing /* How are you? \emph{*/} :: Bob ;
```

A list of match items can also include a side list. A side list is another list of match items that is to be matched. There are different types of side lists. One of them is a repeating side list. A repeating side list matches zero or more occurrences of a list.

In the above example, we matched the name `thing::Bob` by having three match items. But what if we want to match an arbitrary nesting of namespaces, such as

```
thing::Bob::Fred::Ted
```

One way to do this is with a repeating side list.

This list of tokens above could be matched by a single match item followed by a repeating side list. The first item in the list is a regular expression match for a C++ name. The second item in the list is a repeating side list, which contains two items, the first of which is an exact match for operator `::`, and the second of which is a regular expression that matches a C++ name.

The match would work as follows:

The first match item would match `"thing"`. Then the repeating side list would match `::`, `"Bob"`, `::`, `"Fred"`, `::`, and `"Ted"`.

On return from a successful match, how do you know what was matched? Each match item can specify that when the item is matched, the matched item is to be added to a list of items to be returned to the caller.

Let us return to our original example in which we had three lists, the last of which finally matched.

```

          *      *      *
using namespace thing::Bob
```

An asterisk has been placed over each item that is marked to be returned to the caller. At the end of the match, the matching function would return the list

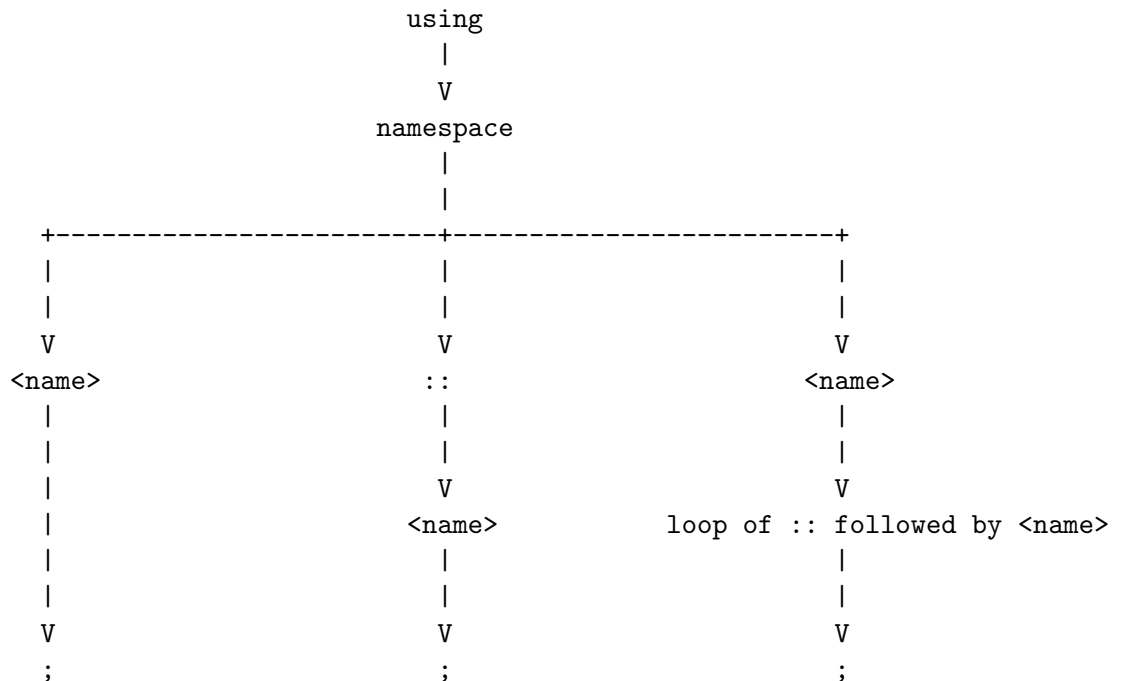
```
namespace
thing
Bob
```

Now the caller knows what was matched and has a copy of the matched tokens. In our example of matching a nested namespace name, the function might return

```
thing
Bob
Fred
Ted
```

Note that since the matched tokens are pushed onto the list, the list is actually returned in reverse order, which the caller can reverse with `nreverse`.

A final example is the list of match items that is actually used elsewhere to find occurrences of using namespace directives. It uses another type of side list, which is a list of lists. This is an illustration of that match structure:



```
using namespace thing::Bob::Ted;
```

77

side list stops the matching when it encounters the terminating semi-colon, and then the next match item in the list matches the terminating semi-colon.

The returned list would be

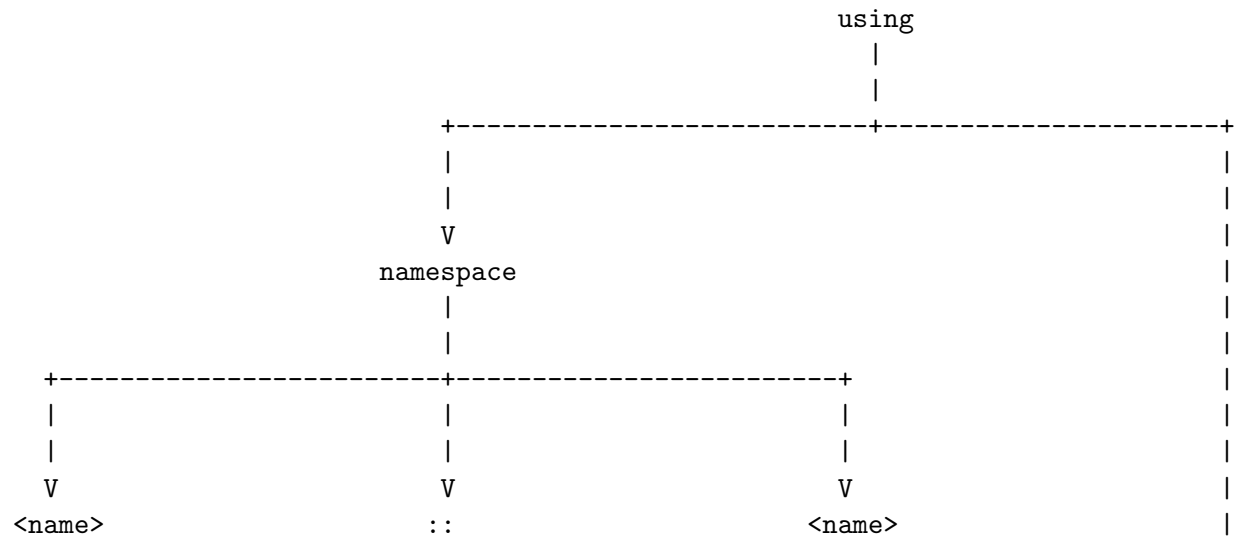
```
using
namespace
thing
Bob
Ted
```

You can look in the unit tests and in other code that uses the matching code here for more examples of the power of match lists.

Here is an example taken from shu-match.el. This match list is used to find all forms of the using statement, including

```
using namespace name;
using namespace ::name;
using namespace name::name;
using name;
using name::name;
```

The actual data structures that define this match list are found in shu-match.el in the constant shu-cpp-match-using-list-single, which is used by the function shu-match-find-all-using-internal, which returns all statements in the above form with one pass through the token list.



If the return value is non-nil, its *rlist* is the list of all of the returned tokens from all of the matches.

shu-cpp-internal-sub-match-tokens *rlist mlist* [Command]

Do the matching for one list only.

shu-cpp-make-match-info *op-code match-eval-func* [Function]
match-token-type

Return a match-info structure from the given arguments

shu-cpp-make-match-side-list *op-code match-list* [Function]
side-parameter

Return a match-info structure from the given arguments that represents a side list.

shu-cpp-match-evaluate-side-list *op-code rlist* [Function]
match-info

Evaluate a side list in a match list. Use the op-code in the match item to find the function that should evaluate the side list.

shu-cpp-match-extract-info *match-info op-code* [Macro]
match-ret-ind *match-*
token-value

Extract the information out of a match-info

shu-cpp-match-extract-op-code *match-info* [Function]

Return the op code from the match-info.

shu-cpp-match-extract-side-list *match-info op-code* [Macro]
side-parameter

Extract the side-list information out of a match-info that represents a side-list..

shu-cpp-match-extract-side-list-only *match-info* [Function]

Extract only the side list from the match info. This is in contract to shu-cpp-match-extract-side-list, which extracts all of the properties of a side list.

shu-cpp-match-extract-token *match-info* [Function]

Return the token from the match-info.

shu-cpp-match-extract-type *match-info* [Function]

Return the token type from the match-info.

shu-cpp-match-is-side-list *op-code* [Function]
Return true if the *op-code* represents a side list operation.

shu-cpp-match-many-list *rlist token-list* [Function]

Do a recursive call to shu-cpp-match-tokens.

shu-cpp-match-op-code-name *op-code* [Function]
Return the name of an op-code.

shu-cpp-match-or-list *rlist token-list match-info* [Function]
rlist points to the current return value list, if any. *token-list* points to the next token-info to match. *match-info* is the head of the side list with which to match. The match succeeds if the first token-info in *token-list* matches any of the match-info members of *match-info*. If the match fails, return nil. If the match succeeds, return a cons cell pointing to two items. The car is the next token-info in *token-list*. The cdr is the return list, *rlist*. *rlist* remains unchanged if the match-info that matched did not specify that the matched token-info was to be returned.

shu-cpp-match-repeat-list *rlist token-list* [Function]

rlist points to the current return value list, if any. *token-list* points to the next token-info to match. *match-info* is the head of the side list with which to match. The match succeeds if the token-infos in *token-list* match all of the match-infos in MATCH-LIST zero or more times. The token-infos are matched repeatedly against the match-infos. If there is a failure matching the first match-info, the match is successful. If there is a failure matching any other match-info, the match fails.

This is useful when matching repeating but optional patterns. For example, a C++ name could be any of the following:

```
a
a::b
a::b::c
```

You can match this with a match list that requires an unquoted token that matches a C++ name, followed by a side list looking for operator “::” followed

by an unquoted token. If there is no match, then you have an unqualified name. If it matches once, you have a name with one level of qualification. But if it fails in the middle, then you have found something that looks like “a::”, which is not a valid C++ name.

shu-cpp-match-repeat-list-once *rlist token-list* [Function]

rlist points to the current return value list, if any. *token-list* points to the next token-info to match. *match-info* is the head of the side list with which to match. The match succeeds if the token-infos in *token-list* match all of the match-infos in MATCH-LIST zero or more times. The token-infos are matched repeatedly against the match-infos. If there is a failure matching any match-info, the match fails.

This is useful when matching repeating patterns. For example, a C++ qualified name could be any of the following:

```
a::b
a::b::c
```

You can match this with a match list that requires an unquoted token that matches a C++ name, followed by a side list looking for operator “::” followed by an unquoted token. If there is a match, you have a name with one level of qualification.

fails in the middle, then you have found something that looks like “a::”, which is not a valid C++ name.

shu-cpp-match-repeat-sub-list *rlist token-list* [Function]
first-required

Go through one iteration of the repeating list. If *first-required* is true, then the first match must succeed. Otherwise, we are matching zero or more instances, so a first match failure is the same as a success.

The iteration is considered a success if either of the following are true: 1. The first match fails and *first-required* is false, or 2. All matches succeed. If all matches succeed, the updated *rlist* and *token-list* are returned. If the first match fails, the *rlist* and *token-list* are returned unaltered. It is as though no match was ever attempted. If some match other than the first fails, nil is returned. If the *token-list* is nil on entry, this is the equivalent of a first match failure.

shu-cpp-match-tokens *rlist match-lists token-list* [Function]

match-lists is a list of match lists. *token-list* is a list of tokens. *rlist* is either nil or an existing list of returned tokens on which to build. For each match-list in *match-lists*, try to match every element of the match list to the token list. if a match fails or if you reach the end of the token list before reaching the end of the match list, move to the next match list and try again. if all elements of a match list match the tokens in the token list, stop the matching process and return (pushed onto RLIST) a list which consists of matched tokens whose corresponding entry in the match list indicated that the matched token was to be added to the list to be returned.

shu-cpp-search-match-tokens *rlist match-list* [Command]

A function that advances through the *token-list* until the first item in the single *match-list* matches the token. If that happens, try to match the whole list. If the whole list is matched, return. If the whole list does not match, restore the original *rlist* and *token-list* and continue. Return only when the whole list is matched or the *token-list* is exhausted.

This could be extended to multiple lists by having the search part check for a match with the head of any of the lists in order. When the head of one list matches, pursue that list. If the list match fails, move to the next list. When no list head or entire list matches the current token, then move to the next token.

shu-cpp-side-list-functions [Constant]

A-list that maps a side list op-code to the function that implements it.

shu-cpp-token-match-same *match-info token-info* [Function]

Perform a single match operation between an item in a token list and an item in a match list. The token types must be the same and the token value in the match list must be the same as the token value in the token list.

shu-cpp-token-match-same-rx *match-info token-info* [Function]

Perform a single match operation by regular expression between an item in a token list and an item in a match list. The token types must be the same and the regular expression in the match list must match (via string-match) the token in the token list.

shu-cpp-token-match-skip *tlist* [Function]

Skip one cell in the input list.

shu-cpp-token-match-type-non-loop-max [Constant]

The maximum match type value that does not indicate a side loop.

shu-cpp-token-match-type-same [Constant]
The match type constant that indicates that the token type and token value must both match.

shu-cpp-token-match-type-same-rx [Constant]
The match type constant that indicates that the token type must match and the token value must satisfy the regular expression for a C++ variable name.

shu-cpp-token-match-type-side-choose [Constant]
The match side constant that indicates a choice. The match is considered a success if any one item in the side list matches the current token.

shu-cpp-token-match-type-side-loop [Constant]
The match side constant that indicates a looping side list. The token list must match the side list zero or more times. If the first item in the list does not match, this is considered a success. If the first item matches, then all items in the side list must match. If all items in the side list match, we go back to the top of the side list and try again until we find a token that does not match the first item in the side list. The match is considered a failure only if there is a partial match between the tokens and the side list.

shu-cpp-token-match-type-side-loop-once [Constant]
The match side constant that indicates a looping side list. The token list must match the side list one or more times. All items in the side list must match at least once. If all items in the side list match, we go back to the top of the side list and try again until we find a token that does not match the first item in the side list. The match is considered a failure if there is no match or only a partial match between the tokens and the side list.

shu-cpp-token-match-type-side-many [Constant]
The match side constant that indicates a choice among multiple lists. This does a recursive call to `shu-cpp-match-tokens`.

shu-cpp-token-match-type-skip [Constant]
The match type constant that indicates skip one input cell.

shu-cpp-token-show-match-info *match-info* [Function]
Show the data in an instance of match-info.

shu-cpp-token-show-match-info-buffer *match-info gb* [Function]

Show the data in an instance of match-info.

`shu-cpp-token-show-match-list` *match-list* **&optional** [Function]

Show the data in a list of match-info.

`shu-cpp-token-show-match-lists` *match-lists* [Function]
title

Show the data in a list of match lists.

9 shu-cpp-misc

A collection of useful functions for dealing with C++ code

9.1 List of functions by alias name

A list of aliases and associated function names.

acgen <i>class-name</i>	[Command]
(Function: shu-cpp-acgen)	
Generate a skeleton class code generation at point.	
ccdecl <i>class-name</i>	[Command]
(Function: shu-cpp-ccdecl)	
Generate a skeleton class declaration at point.	
ccgen <i>class-name</i>	[Command]
(Function: shu-cpp-ccgen)	
Generate a skeleton class code generation at point.	
cdecl <i>class-name</i>	[Command]
(Function: shu-cpp-cdecl)	
Generate a skeleton class declaration at point.	
cgen <i>class-name</i> &optional <i>use-allocator</i>	[Command]
(Function: shu-cpp-cgen)	
Generate a skeleton class code generation at point.	
dox-file	[Command]
(Function: shu-dox-file)	
Place a skeleton Doxygen file definition at point.	
fline	[Command]
(Function: shu-fline)	
Place a stream of <code>__FILE__</code> and <code>__LINE__</code> at point.	
gen-component <i>class-name</i>	[Command]
(Function: shu-gen-component)	
Generate the three files for a new component: <code>.cpp</code> , <code>.h</code> , and <code>.t.cpp</code>	
hcggen <i>class-name</i>	[Command]
(Function: shu-cpp-hcggen)	

Generate a skeleton class code generation at point.

9.2 List of functions and variables

List of functions and variable definitions in this package.

shu-cpp-acgen *class-name* [Command]
(Alias: acgen)

Generate a skeleton class code generation at point.

shu-cpp-ccdecl *class-name* [Command]
(Alias: ccdecl)

Generate a skeleton class declaration at point.

shu-cpp-ccgen *class-name* [Command]
(Alias: ccgen)

Generate a skeleton class code generation at point.

shu-cpp-cdecl *class-name* [Command]
(Alias: cdecl)

Generate a skeleton class declaration at point.

shu-cpp-cgen *class-name* **&optional** *use-allocator* [Command]
(Alias: cgen)

Generate a skeleton class code generation at point.

shu-cpp-decl-class-name *class-name* [Function]
Generate a comment which is the name of the class with a line of outline characters above and below it. *class-name* is the name of the class. *outline-character* is a string containing the outline character (usually “=” or “-”)

shu-cpp-decl-cpp-class-name *class-name* [Function]
Generate a comment which is the name of the class with a line of dashes above and below it to set off the class name in a cpp file. *class-name* is the name of the containing C++ class.

shu-cpp-decl-cpp-print-self *class-name* [Function]
Generate the skeleton code for the printSelf() function. *class-name* is the name of the containing C++ class.

shu-cpp-decl-cpp-stream *class-name* [Function]

Generate the code for the streaming operator (`operator<<()`). *class-name* is the name of the containing C++ class.

shu-cpp-decl-h-class-name *class-name* [Function]

Generate a comment which is the name of the class with a line of equal signs above and below it to set off the class name in a header file. *class-name* is the name of the containing C++ class.

shu-cpp-decl-h-print-self [Function]

Generate the declaration of the `printSelf()` function.

shu-cpp-decl-h-stream *class-name* [Function]

Generate the declaration for the streaming operator (`operator<<()`). *class-name* is the name of the containing C++ class.

shu-cpp-gen-decl-h-private *class-name* **&optional** [Function]

Generate the private section of the class declaration. If *copy-allowed* is false, generate private an unimplemented copy constructor and `operator=()`

shu-cpp-gen-h-class-intro *class-name* [Function]

Generate the preamble to a class declaration in a header file. This is all of the code that precedes the DATA comment. Return the position at which the class comment was placed.

shu-cpp-gen-inline-template-header [Function]

Doc string.

shu-cpp-hcgen *class-name* [Command]

(Alias: `hcgen`)

Generate a skeleton class code generation at point.

shu-cpp-inner-cdecl *class-name* *copy-allowed* [Function]

use-allocator

Generate a skeleton class declaration at point.

shu-cpp-misc-gen-ctor-not-implemented *class-name* [Function]

Generate a declaration of a non-implemented copy constructor and `operator=()`.

shu-cpp-misc-gen-h-ctor *class-name* **&optional** [Function]

Generate a declaration of a constructor for the given *class-name*. If *use-allocator* is true, the constructor declaration includes an optional allocator.

shu-cpp-misc-gen-h-dtor *class-name* [Function]
Generate a commented out declaration of a destructor for the given *class-name*.

shu-cpp-misc-gen-nested-traits *class-name* [Function]
Generate a nested traits declaration.

shu-cpp-misc-gen-not-implemented *class-name* [Function]
Generate a declaration of a non-implemented copy constructor and operator=().

shu-cpp-misc-gen-op-equal-not-implemented *class-name* [Function]
Generate a declaration of a non-implemented copy constructor and operator=().

shu-cpp-misc-h-tail-gen *class-name* [Function]
Generate the tail code in a header file starting with “INLINE AND TEMPLATE FUNCTION IMPLEMENTATIONS”

shu-cpp-misc-inline-template-label [Constant]
The text of the header that starts inline and template functions in a C++ header file

shu-cpp-misc-not-implemented-label [Constant]
The label put in a header file for functions and operators that are deliberately not implemented.

shu-cpp-misc-set-alias [Function]
Set the common alias names for the functions in shu-cpp-misc. These are generally the same as the function names with the leading shu- prefix removed.

shu-dox-file [Command]
(Alias: dox-file)
Place a skeleton Doxygen file definition at point.

shu-fline [Command]
(Alias: fline)
Place a stream of `__FILE__` and `__LINE__` at point.

shu-gen-component <i>class-name</i>	[Command]
(Alias: gen-component)	
Generate the three files for a new component: .cpp, .h, and .t.cpp	
shu-generate-cfile <i>author namespace class-name</i>	[Function]
Generate a skeleton cpp file	
shu-generate-hfile <i>author namespace class-name</i>	[Function]
Generate a skeleton header file	
shu-generate-tfile <i>author namespace class-name</i>	[Function]
Generate a skeleton t.cpp file	

10 shu-cpp-project

A collection of useful functions for dealing with project files and treating a set of source files in multiple directories as a single project

10.0.1 Toggle back and forth between files

If you are editing a C or C++ file and wish to switch to its associated header file, *shu-hother* will switch to the header file. *shu-cother* will switch back to the original C or C++ file. *shu-tother* will switch to the associated unit test file that ends in “t.cpp.” This set of functions allows you to treat a set of files as a single project.

You define a project by creating a project file. A project file is simply a text file with one or more directory names in it. You select the text in the file and invoke the command *shu-make-c-project*. This searches the given directories for all of the header files and C and C++ files and remembers where these files are located.

When you want to visit a file you start typing the first few characters of the file name. You can use auto completion to complete the name. Since *shu-project* knows where all of the files are located, you do not have to remember that. Once you have typed in a complete file name, the file will be visited wherever it happens to reside.

If you have two or more files with the same name in different directories, you will be presented with a menu and asked to select the file you have in mind.

10.0.2 Creating a project file

If you do not have a project file, *shu-project* gives you a convenient way to create one. Go to the root directory of your project, open a new empty file (I usually call it *project.txt*), and invoke the command *shu-make-c-project*. It will prompt you for the root directory with a default being the current directory. It will then find all of the directories at or below the root that contain code and it will place the directory names in the file. Now you have your project file.

10.0.3 File names within the project

shu-project now knows where all of the file names reside so you do not have to remember that. But it does more to simplify your life. Many files in large projects start with a common prefix. If you have a class called *ThingLoader*, it might be defined in a file called *thingloader.h*. But in a large project, the full file name might be something like *myproject_thingloader.h*.

As shu-project is creating the list of file names, it is also creating an index of short names with common prefixes stripped. So if you want to visit `myproject_thingloader.h`, just type “thing” and hit tab to autocomplete. If no other file starts with “thing”, shu-project will autocomplete to “thingloader” and then look in its index to find `myproject_thingloader.h`.

10.0.4 Visiting related files

If you are in a `.cpp` file and you want to visit its associated `.h` file, issue the command `shu-hother` and you will be taken to the `.h` file even if it is in a different directory. Similarly, you can visit the associated unit test file (`t.cpp`) with the command `shu-tother` to visit the unit test file.

10.0.5 Visiting files based on error messages

You compile a file and the compiler complains ...

```
..\myproject_thingloader.cpp:190:6: error: Invalid type for ...
```

Simply place the cursor under any part of the file name and type `Ctrl-x h`. shu-project will take you to line 190, column 6 of `myproject_thingloader.cpp`.

10.1 List of functions by alias name

A list of aliases and associated function names.

clear-c-project	[Command]
(Function: <code>shu-clear-c-project</code>)	
Clear an existing project, if any.	

clear-prefix	[Function]
(Function: <code>shu-clear-prefix</code>)	
Clear the default file name prefix for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.	

cother	[Command]
(Function: <code>shu-cother</code>)	
Visit a <code>.cpp</code> file from the corresponding <code>.t.cpp</code> or <code>.h</code> file. If visiting a <code>t.cpp</code> or <code>.h</code> file, invoke this function and you will be taken to the corresponding <code>.cpp</code> or <code>.c</code> file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.	

count-c-project [Command]
(Function: shu-count-c-project)
Count the number of lines of code in a project. The final count is shown in the minibuffer. The counts of individual subdirectories are stored in the temporary buffer `{*shu-project-count*}`

hother [Command]
(Function: shu-hother)
Visit a `.h` file from the corresponding `.cpp` or `t.cpp` file. If visiting a `.cpp` or `t.cpp` file, invoke this function and you will be taken to the corresponding `.h` file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

iother [Command]
(Function: shu-iother)
Visit a `i.cpp` file from the corresponding `.cpp` or `.h` file. If visiting a `.c` or `.cpp` file, invoke this function and you will be taken to the corresponding `.i.cpp` file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

list-c-directories [Command]
(Function: shu-list-c-directories)
Insert into the current buffer the names of all of the directories in a project.

list-c-duplicates [Command]
(Function: shu-list-c-duplicates)
Insert into the current buffer a list of all of the duplicate files names. Under each duplicate file name, insert a list of the full paths to all of the duplicates.

list-c-file-names [Command]
(Function: shu-list-c-file-names)
Insert into the buffer a list of all of the unique file names in the project. This does not include the file path as it will be different for duplicate names.

list-c-prefixes [Command]
(Function: shu-list-c-prefixes)
List all of the file prefixes found in the current project, if any. See the doc-string for *shu-project-split-file-name* for further information about extracted file prefixes.

list-c-project [Command]

(Function: `shu-list-c-project`)
 Insert into the current buffer the names of all of the code files in the current project.

`list-completing-names` [Command]
 (Function: `shu-cpp-list-completing-names`)
 List all of the names that are used to do a completing read of a file name along with the names of the actual files to which they map.

`list-project-names` [Command]
 (Function: `shu-cpp-list-project-names`)
 List all of the names in a project with the names of the files to which they map.

`list-short-names` [Command]
 (Function: `shu-cpp-list-short-names`)
 List all of the short names in a project with the names of the files to which they map.

`make-c-project proj-root` [Command]
 (Function: `shu-make-c-project`)
 Create a project file of all directories containing c or h files. Starts at the specified root directory and searches all subdirectories for any that contain c or h files. Top level directories whose names are found in *shu-project-exclude-list* are excluded from the search. Typically *shu-project-exclude-list* is used to exclude CMake directories that include c or h files that have been created as part of the build process and are not members of the repository itself. It then inserts all of the directory names into the current file at point.

`make-full-c-project proj-root` [Command]
 (Function: `shu-make-full-c-project`)
 Create a project file of all directories containing c or h files. Starts at the specified root directory and searches all subdirectories for any that contain c or h files. It then inserts all of the directory names into the current file at point.

`make-p-project proj-root` [Command]
 (Function: `shu-make-p-project`)
 Create a Python project that is analogous to a c project.

`other` [Command]

(Function: shu-other)

Visit an h file from a c file or a c file from an h file. If visiting a .h file, invoke this function and you will be taken to the .c or .cpp file. If visiting a .c or .cpp file, invoke this function and you will be taken to the corresponding .h file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

renew-c-project [Command]

(Function: shu-renew-c-project)

Renew a previously established project to pick up any new files.

set-c-project *start end* [Command]

(Function: shu-set-c-project)

Mark a region in a file that contains one subdirectory name per line. Then invoke set-c-project and it will find and remember all of the c and h files in those subdirectories. You may then subsequently visit any of those files by invoking M-x vh which will allow you to type in the file name only (with auto completion) and will then visit the file in the appropriate subdirectory. If this function is called interactively, it clears the project name that was established by either *shu-setup-project-and-tags* or *shu-visit-project-and-tags*.

set-dir-prefix *prefix* [Command]

(Function: shu-set-dir-prefix)

Set the default file name prefix to be the current directory name end for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.

set-p-project *start end* [Command]

(Function: shu-set-p-project)

Mark a region in a file that contains one subdirectory name per line. Then invoke set-c-project and it will find and remember all of the c and h files in those subdirectories. You may then subsequently visit any of those files by invoking M-x vh which will allow you to type in the file name only (with auto completion) and will then visit the file in the appropriate subdirectory. If this function is called interactively, it clears the project name that was established by either *shu-setup-project-and-tags* or *shu-visit-project-and-tags*.

set-prefix *prefix* [Command]

(Function: shu-set-prefix)

Set the default file name prefix for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.

tother [Command]

(Function: shu-tother)

Visit a `t.cpp` file from the corresponding `.cpp` or `.h` file. If visiting a `.c` or `.cpp` file, invoke this function and you will be taken to the corresponding `.t.cpp` file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

vf [Command]

(Function: shu-vf)

If point is on something that looks like a file name, visit the file. If the file name is followed by a colon and a number, the number is interpreted as a line number within the file and point is moved to the beginning of that line. If the line number is followed by a colon and another number, then the second number is interpreted as a column number and point is moved to that column number.

which-c-project [Command]

(Function: shu-which-c-project)

Identify the current project by putting into a project buffer the name of the file from which the project was derived as well as the name of all of the directories in the project. Then switch to that buffer. The idea is to invoke this function, look at the results in that buffer, and then quit out of the buffer.

10.2 List of functions and variables

List of functions and variable definitions in this package.

shu-add-cpp-c-extensions *xtns* [Function]

Add one or more file extensions to the list of C and C++ extensions recognized by the C package functions. Argument may be a single extension in a string or a list of strings. This modifies both `shu-cpp-c-extensions` and `shu-project-extensions`.

shu-add-cpp-h-extensions *xtns* [Function]

Add one or more file extensions to the list of C and C++ extensions recognized by the C package functions. Argument may be a single extension in a string or a list of strings. This modifies both `shu-cpp-h-extensions` and `shu-project-extensions`.

shu-add-cpp-package-line *dir-name* [Function]

Called with point at the beginning of the line. Take the whole line as the name of a directory, look into the directory, and create an alist of all of the files in the directory as described in shu-cpp-subdir-for-package.

shu-clear-c-project [Command]
 (Alias: clear-c-project)
 Clear an existing project, if any.

shu-clear-prefix [Function]
 (Alias: clear-prefix)
 Clear the default file name prefix for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.

shu-completion-is-directory [Variable]
 True if we are to use the current directory name as the file name prefix.

shu-cother [Command]
 (Alias: cother)
 Visit a .cpp file from the corresponding .t.cpp or .h file. If visiting a t.cpp or .h file, invoke this function and you will be taken to the corresponding .cpp or .c file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

shu-count-c-project [Command]
 (Alias: count-c-project)
 Count the number of lines of code in a project. The final count is shown in the minibuffer. The counts of individual subdirectories are stored in the temporary buffer `{*shu-project-count*}`

shu-count-in-cpp-directory *directory-name pbuf* [Function]
 t-h-files t-c-files *t-*
c-count

Count the lines of code in each of the code files in the given directory, updating the message in the minibuffer and passing the totals back to the caller.

shu-cpp-c-extensions [Constant]
 A list of file extensions for all of the C file types we want to find. This is defined as defconst in shu-cpp-base.el but may be modified by shu-add-cpp-c-extensions.

shu-cpp-c-file-count [Variable]
 This is the count of the number of C files found in the project.

- shu-cpp-choose-file** *assoc-result* [Function]
 Choose the file to visit for a given unqualified name. If there is only one file associated with the name then visit it. If there are multiple files put all of the fully qualified file names in the completion buffer and give the user the opportunity to select the desired file. Then visit that file.
- shu-cpp-choose-other-file** *newfile* [Function]
 Try to visit a file first within a project and, if not successful, in the current directory. If no project is in use or if the file does not belong to the project, try to find the file in the current directory. If a file was found and visited, return true.
- shu-cpp-choose-project-file** *newfile* [Function]
 Try to visit a file within a project. If a project is in use, try to visit the given file in the list of files that belong to the project. This goes through the standard project selection process, including prompting the user to choose the desired file if more than one file with the same name exists. If a file was found and visited, return true.
- shu-cpp-class-list** [Variable]
 This is an alist whose keys are unqualified file names and whose values contain a list of the fully qualified files with the same unqualified name. if *shu-cpp-project-short-names* is nil, this list is identical to the one stored in *shu-cpp-completing-list*.
- shu-cpp-common-completion** [Function]
 Called when the user hits enter or clicks mouse button 2 on completion window. At this point the users selected choice is in the current buffer. We get the answer from the current buffer and call the function that is currently pointed to by *shu-cpp-completion-target*.
- shu-cpp-completing-list** [Variable]
 This is an alist whose keys are unqualified file names and whose values contain a list of the fully qualified files with the same unqualified name. If *shu-cpp-project-short-names* is non-nil, then this alist includes the short file names as well.
- shu-cpp-completion-current-buffer** [Variable]
 Active buffer just before we have to do a completion.
- shu-cpp-completion-prefix** [Variable]

The default file name prefix when we are looking for a file and point is not sitting on something that appears to be a file name.

shu-cpp-completion-scratch [Variable]
Scratch buffer used by C file name completions.

shu-cpp-completion-target [Variable]
Global variable used to hold the function to be invoked at the end of the current completion.

shu-cpp-final-list [Variable]
The name of the shared variable that contains the list of directories assembled by shu-make-c-project

shu-cpp-finish-project [Function]
Finish constructing a C project from a user file list. The input is KEY-LIST, which is an a-list. The cdr of each entry is the short (unqualified) file name. The cdr of each entry is the fully qualified name. This alist may have duplicate short names. This function produces a new list. The car of each item is still the short (unqualified) file name. The cdr is a list of all of the fully qualified file names to which the short name maps. If a user selects a file that has only one fully qualified file name, we open the file. But if it has more than one fully qualified file name, we have to ask the user which one is wanted.

shu-cpp-found-extensions [Variable]
This is a list of all of the file extensions found in the current project. While shu-project-extensions contains all of the extensions that we look for. This variable contains those that we actually found in building the current project.

shu-cpp-h-extensions [Constant]
A list of file extensions for all of the H file types we want to find. This is defined as defconst in shu-cpp-base.el but may be modified by shu-add-cpp-h-extensions

shu-cpp-h-file-count [Variable]
This is the count of the number of H files found in the project.

shu-cpp-internal-list-names *name-list type-name* [Function]
Implementation function for *shu-cpp-list-short-names*, *shu-cpp-list-project-names*, and *shu-cpp-list-completing-names*.

shu-cpp-list-completing-names [Command]

(Alias: list-completing-names)

List all of the names that are used to do a completing read of a file name along with the names of the actual files to which they map.

shu-cpp-list-project-names [Command]

(Alias: list-project-names)

List all of the names in a project with the names of the files to which they map.

shu-cpp-list-short-names [Command]

(Alias: list-short-names)

List all of the short names in a project with the names of the files to which they map.

shu-cpp-prefix-list [Variable]

This is the list of prefixes removed from the short names if *shu-cpp-project-short-names* is non-nil.

shu-cpp-project-collapse-list *key-list* [Function]

key-list is an alist in which the cdr of each item is the unqualified file name and the car of each item is the fully qualified file name, including the path to the file. The output is a different alist in which the car of each item is the unqualified file name and the cdr of each item is the list of fully qualified file names to which the unqualified file name refers.

For example, if *key-list* contains:

```
(('xxx_mumble.h' . '/foo/bar/xxx_mumble.h'))
('xxx_stumble.h . '/foo/bar/xxx_stumble..h'))
('xxx_stumble.h . '/boo/baz/xxx_stumble..h'))
```

then the returned list will contain

```
(('xxx_mumble.h' . '/foo/bar/xxx_mumble.h'))
('xxx_stumble.h . '/foo/bar/xxx_stumble..h' '/boo/baz/xxx_stumble..h'))
```

shu-cpp-project-extract-base-name *name* [Function]

This is the implementation function of *shu-cpp-project-get-base-name* so that the logic of the function can be unit tested.

shu-cpp-project-file [Variable]

The name of the file from which the current project was read.

shu-cpp-project-get-base-name [Command]

When visiting a file within a project, the name might consist of two or three parts - the file name, the normal extension, such as .h or .cpp, and the intermediate extension, such as .i or .t when visiting .i.cpp or .t.cpp. This function gets the buffer file name and removes the one or two extensions in order to get the name with no extensions at all.

shu-cpp-project-get-list-counts *proj-list* [Function]

proj-list is an alist whose structure is identical to that of *shu-cpp-class-list*. This function returns a list with three items on it: the number of c / cpp files, the number of h files, and the number of duplicate names found in the list.

shu-cpp-project-invert-list *proj-list* [Function]

proj-list is an alist in which the cdr of each item is the unqualified file name and the car of each item is the list of fully qualified file names to which the unqualified name refers. The returned output is a single list of fully qualified file names.

shu-cpp-project-list [Variable]

List that holds all of the subdirectories in the current project.

shu-cpp-project-name [Variable]

If the current project was established by either *shu-setup-project-and-tags* or *shu-visit-project-and-tags*, this is the name of the interactive function that was invoked by the user to set it up. This is useful when you are in a project and you forgot the name of the interactive function that got you there.

shu-cpp-project-set-alias [Function]

Set the common alias names for the functions in shu-cpp-project. These are generally the same as the function names with the leading shu- prefix removed.

shu-cpp-project-short-names [Custom]

Set non-nil if shu-cpp-project creates short names for files in a project. A short name is an approximation of the file name that may be easier to type. For example, if all of the files in a project begin with a common prefix (e.g., “x_server_mumble.cpp” and “x_server_stumble.cpp”, then the short names for these two files would be “mumble.cpp” and “stumble.cpp”. This

means that the user does not have to type the prefix in order to find the file. If the user types “mumble.cpp” as the file name, emacs will open the file “x_server_mumble.cpp”.

shu-cpp-project-subdirs *dir-name level* [Function]

Starting with the directory name *dir-name*. create a list of subdirectories whose head is in *shu-cpp-final-list*, that contains the name of every directory and subdirectory that contains C, C++, or H files. This is used by shu-make-c-project and other functions that wish to discover all directories that might contain source code.

shu-cpp-project-time [Variable]

This is the time at which the current project was created.

shu-cpp-project-very-short-names [Custom]

Set non-nil if shu-cpp-project creates very short names for files in a project. A short name is an approximation of the file name that may be easier to type. For example, if all of the files in a project begin with a common prefix (e.g., “my_own_server_mumble.cpp” and “my_own_server_stumble.cpp”, then the short names for these two files would be “mumble.cpp” and “stumble.cpp”. This means that the user does not have to type the prefix in order to find the file. If the user types “mumble.cpp” as the file name, emacs will open the file “my_own_server_mumble.cpp”.

shu-cpp-resolve-choice *full-name-list target* [Function]

Choose from a number of possible file names. We have found an unqualified file name of interest but it resolves to multiple fully qualified file names. Display all of the possibilities in a completion buffer and ask the user to choose the desired one. The string containing the chosen fully qualified file name will then be passed to the function pointed to by target.

shu-cpp-short-list [Variable]

This is the list of short names, if there are any. The car of each item is the short name. The cdr of each item is the full path to the associated file name.

shu-cpp-subdir-for-package *directory-name* [Function]

Given a subdirectory name return an alist that contains as keys the names of all of the c and h files in the subdirectory, and as values the the fully qualified name and path of the c or h file. So if the directory “/u/foo/bar” contains thing.c and what.h the returned alist would be


```
( ( 'thing.c' ' /u/foo/bar/thing.c' )
  ( 'what.h' ' /u/foo/bar/what.h' ) )
```

This allows us to associate the key “thing.c” with the fully qualified name “/u/foo/bar/thing.c”.

shu-cpp-target-file-column [Variable]

If non-nil, this represents the column number that is to be located after a file is visited by `vh()` and has gone through buffer completion selection.

shu-cpp-target-file-line [Variable]

If non-nil, this represents the line number that is to be located after a file is visited by `vh()` and has gone through buffer completion selection.

shu-cpp-visit-target *file-name* [Function]

This is the function that visits the file name chosen by `vh()` and perhaps by a completing read from a completion buffer.

shu-default-file-to-seek [Variable]

The default file to seek that is proposed as a possible file when `vh()` finds a file name under the cursor, possibly with a line number. If the user chooses a file other than this one, we need to forget the associated line number.

shu-find-default-cpp-name [Function]

Find a default file name to visit. Calls `shu-find-line-and-file` to find a possible file name and possible line number within the file. Return the file name if one is found and sets `shu-cpp-target-file-line` to the line number if one is found

shu-find-line-and-file [Function]

If point is sitting on the word “line”, then look for a string of the form “line 678 of frobnitz.cpp” and return a list whose first item is the file name and whose second item is the line number. If point is not sitting on the word “line”, then check to see if point is sitting on a string that has the syntax of a valid file name. If that is the case, remember the file name. If the file name is followed by a colon, look for a line number following the colon. If found, look for another colon followed by a possible column number. This function will return nil if none of the above are found. If only a file name is found, return a list with one entry. If file name and line number, a list with two entries. If file name, line number, and column number, a list with three entries.

shu-get-line-column-of-file [Function]

Fetch the potential line number and column number within a file. On entry, point is positioned at the character following a file name. This file name may be followed by a line number and the line number may be followed by a column number. This function recognizes four forms of line and column specifications.

thing.cpp:1234:42

indicates the file thing.cpp line number 1234, column 42

[file=thing.cpp] [line=1234]

indicates the file thing.cpp line number 1234.

“thing.cpp”, line 55:16:

indicates the file thing.cpp line number 55, column 16.

“thing.cpp”, line 55:

indicates the file thing.cpp line number 55.

The purpose of this function is only to gather the line and column specification following the file name. The return value is a list, which is empty if no line or column number was found. It has only one element, which is the line number if only a line number was found. It has two elements, which are the line number and column number if both line number and column number were found.

This should probably be turned into a hook at some point so that other line and column number indications may be used.

shu-get-real-this-command-name [Function]

Return the symbol name of the variable “real-this-command” if it is defined. If not defined, return the string “{**unknown**}”. Some older versions of emacs do not support real-this-command.

shu-global-operation *documentation* [Function]

&optional *search-target*

Invoke a function on every file in the project. *documentation* is the string of put in the buffer to describe the operation.

shu-global-search-replace *file argument1 argument2* [Function]

This function is called once for each file in the project. The first argument is the file name. The second argument is a list holding lists of search and replace operations. Each search and replace operation is defined by a list of arguments as follows:

1. A boolean value, `t` means case sensitive search, `nil` means ignore case
2. The function to call to do the search. This must be
`'search-forward`, `'re-search-forward`, or any function with the same
signature and behavior.
3. The string that is the target of the search
4. The string the is to replace the target whenever found
5. An optional second argument to be passed to `replace-match`
The default value is `t`
6. An optional third argument to be passed to `replace-match`
The default value is `t`

For example

```
(list
  (list t 'search-forward 'Mumble' 'Bumble')
  (list nil 'search-forward 'howdy' 'doody'))
```

is a list that defines two search and replace operations. Both operations use the `search-forward` function. The first is a case sensitive search and replace to replace all instances of “Mumble” with “Bumble”. The second is a case insensitive search and replace to replace all instances of “howdy” with “doody”.

These operations may be performed on every file in the project as follows:

```
(setq ops
  (list
    (list t 'search-forward 'Mumble' 'Bumble')
    (list nil 'search-forward 'howdy' 'doody'))))
(setq doc 'Description of change')
(shu-global-operation doc 'shu-global-search-replace ops)
```

shu-hother [Command]
(Alias: `hother`)

Visit a `.h` file from the corresponding `.cpp` or `t.cpp` file. If visiting a `.cpp` or `t.cpp` file, invoke this function and you will be taken to the corresponding `.h` file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

shu-internal-list-c-duplicates *proj-list* [Function]

Internal implementation of shu-list-c-duplicates.

shu-internal-list-c-file-names *proj-list* [Command]
Doc string.

shu-internal-list-c-project *proj-list* [Function]
Insert into the current buffer the names of all of the code files in the project whose files are in *proj-list*.

shu-internal-set-c-project *start end* [Function]
Mark a region in a file that contains one subdirectory name per line. Then invoke set-c-project and it will find and remember all of the c and h files in those subdirectories. You may then subsequently visit any of those files by invoking M-x vh which will allow you to type in the file name only (with auto completion) and will then visit the file in the appropriate subdirectory.

shu-internal-visit-project-file *look-for-target* [Function]
Visit a c or h file in a project.

shu-internal-which-c-project *pbuf* [Function]
Undocumented

shu-iother [Command]
(Alias: iother)
Visit a i.cpp file from the corresponding .cpp or .h file. If visiting a .c or .cpp file, invoke this function and you will be taken to the corresponding .i.cpp file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

shu-list-c-directories [Command]
(Alias: list-c-directories)
Insert into the current buffer the names of all of the directories in a project.

shu-list-c-duplicates [Command]
(Alias: list-c-duplicates)
Insert into the current buffer a list of all of the duplicate files names. Under each duplicate file name, insert a list of the full paths to all of the duplicates.

shu-list-c-file-names [Command]
(Alias: list-c-file-names)
Insert into the buffer a list of all of the unique file names in the project. This does not include the file path as it will be different for duplicate names.

shu-list-c-prefixes [Command]
 (Alias: list-c-prefixes)
 List all of the file prefixes found in the current project, if any. See the doc-string for *shu-project-split-file-name* for further information about extracted file prefixes.

shu-list-c-project [Command]
 (Alias: list-c-project)
 Insert into the current buffer the names of all of the code files in the current project.

shu-list-in-cpp-directory *directory-name* [Function]
 Insert into the current buffer the names of all of the code files in a directory.

shu-make-c-project *proj-root* [Command]
 (Alias: make-c-project)
 Create a project file of all directories containing c or h files. Starts at the specified root directory and searches all subdirectories for any that contain c or h files. Top level directories whose names are found in *shu-project-exclude-list* are excluded from the search. Typically *shu-project-exclude-list* is used to exclude CMake directories that include c or h files that have been created as part of the build process and are not members of the repository itself. It then inserts all of the directory names into the current file at point.

shu-make-full-c-project *proj-root* [Command]
 (Alias: make-full-c-project)
 Create a project file of all directories containing c or h files. Starts at the specified root directory and searches all subdirectories for any that contain c or h files. It then inserts all of the directory names into the current file at point.

shu-make-p-project *proj-root* [Command]
 (Alias: make-p-project)
 Create a Python project that is analogous to a c project.

shu-on-the-word-line [Function]
 Return the character position of the start of the current word if point is sitting anywhere on the word “line”. This is used pick up file positions of the form: “line 628 of frobnitz.cpp”

shu-other [Command]

(Alias: other)

Visit an h file from a c file or a c file from an h file. If visiting a .h file, invoke this function and you will be taken to the .c or .cpp file. If visiting a .c or .cpp file, invoke this function and you will be taken to the corresponding .h file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

shu-possible-cpp-file-name &optional [Function]
any-extension

Return a list containing a possible file name with a possible line number and a possible column number. If the thing on point does not resemble a file name, return nil. If it looks like a file name, save it and call shu-get-line-column-of-file to perhaps harvest a line number and column number within the file. The return result is a list of length one if there is only a file name, a list of length two if there is a file name and line number, a list of length three if there is a file name, line number, and column number. If the optional argument *include-directory* is true, the file name may include the forward slash character, which means that the returned file name may also include directory names. Normally, this function only looks for code files, but if the optional argument *any-extension* is true, then a file name with any extension will be returned.

shu-project-cpp-buffer-name [Constant]
The name of the buffer into which messages are placed as c and h files are being scanned.

shu-project-exclude-hash [Variable]
The hash table that holds the directory names from *shu-project-exclude-list*.

shu-project-exclude-list [Constant]
A list of top level directory names to exclude while creating a project via *shu-make-c-project*. This list is ignored by *shu-make-full-c-project*

shu-project-extensions [Constant]
A list of file extensions for all of the file types we want to find. This is defined as defconst in shu-cpp-base.el but may be modified by shu-add-cpp-c-extensions or shu-add-cpp-h-extensions.

shu-project-file-list [Variable]
This is a list of the full path and name of every file in the project. It is used when a global change needs to visit every file in the project.

shu-project-get-file-info *plist file-name* [Macro]

Extract the file information from one entry in *shu-cpp-class-list*.

shu-project-make-exclude-hash [Command]

Turn *shu-project-exclude-list* into the hash table *shu-project-exclude-hash*. If *shu-project-exclude-list* is nil or not a list or an empty list, then *shu-project-exclude-hash* is also set to nil.

shu-project-make-short-key-list *key-list* [Function]

key-list is an alist in which the car of each item is the unqualified file name and the cdr of each item is the fully qualified file name, including the path to the file. This function creates two lists. One is an alist of all of the file prefixes. That car of each item is the prefix. The cdr of each item is the number of times that prefix was found. The second is a list similar to *key-list* with all of the file names changed to their equivalent short names. If the long and short names are the same, then that item is omitted from the new list of short names.

Return is a cons cell whose car is the prefix list and whose cdr is the short name list.

shu-project-split-file-name *file-name* [Function]

Split *file-name* into two parts. The first part is the prefix and the second part is the short name. The rules for splitting are as follows:

If the name has no underscores, then the prefix is empty and the short name is the whole name.

If the name has one underscore in it (e.g., “abcdef_mumble.cpp”), then the prefix is the part before the underscore (“abcdef”) and the short name is all of the rest (“mumble.cpp”).

If the name has more than one underscore in it (e.g., “x_abcdef_mumble.cpp”), then we look at the length of the first part (“x”). If its length is one, then the prefix is the concatenation of the first two parts (“x_abcdef”), and the short name is the rest (“mumble.cpp”). If the length of the first part is not one (e.g., file name is “lovely_looking_mumble.cpp”), then the prefix is the first part (“lovely”) and the short name is the rest (“looking_mumble.cpp”).

After the split, the short name is converted to all lower case.

Return a cons cell of the form (prefix . short-name)

shu-project-user-class-count [Variable]

Undocumented

shu-py-extensions [Constant]
A list of file extensions for Python projects

shu-renew-c-project [Command]
(Alias: `renew-c-project`)
Renew a previously established project to pick up any new files.

shu-set-c-project *start end* [Command]
(Alias: `set-c-project`)
Mark a region in a file that contains one subdirectory name per line. Then invoke `set-c-project` and it will find and remember all of the `c` and `h` files in those subdirectories. You may then subsequently visit any of those files by invoking `M-x vh` which will allow you to type in the file name only (with auto completion) and will then visit the file in the appropriate subdirectory. If this function is called interactively, it clears the project name that was established by either *shu-setup-project-and-tags* or *shu-visit-project-and-tags*.

shu-set-dir-prefix *prefix* [Command]
(Alias: `set-dir-prefix`)
Set the default file name prefix to be the current directory name end for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.

shu-set-p-project *start end* [Command]
(Alias: `set-p-project`)
Mark a region in a file that contains one subdirectory name per line. Then invoke `set-c-project` and it will find and remember all of the `c` and `h` files in those subdirectories. You may then subsequently visit any of those files by invoking `M-x vh` which will allow you to type in the file name only (with auto completion) and will then visit the file in the appropriate subdirectory. If this function is called interactively, it clears the project name that was established by either *shu-setup-project-and-tags* or *shu-visit-project-and-tags*.

shu-set-prefix *prefix* [Command]
(Alias: `set-prefix`)
Set the default file name prefix for those times when we are trying to visit a project file and point is not sitting on something that resembles a file name.

shu-setup-project-and-tags *proj-dir* [Function]

Visit a project file, make a C project from the contents of the whole file, create a file called “files.txt” with the name of every file found, invoke ctags on that file to build a new tags file, and then visit the tags file. *proj-dir* is the name of the directory in which the project file exists and in which the tags file is to be built.

shu-sub-make-c-project *proj-root* [Function]

Create a project file of all directories containing c or h files. Starts at the specified root directory and searches all subdirectories for any that contain c or h files. It then inserts all of the directory names into the current file at point.

shu-tother [Command]

(Alias: tother)

Visit a t.cpp file from the corresponding .cpp or .h file. If visiting a .c or .cpp file, invoke this function and you will be taken to the corresponding .t.cpp file. This function will use a project if one is active. Otherwise, it will assume that all files reside in the same directory.

shu-vf [Command]

(Alias: vf)

If point is on something that looks like a file name, visit the file. If the file name is followed by a colon and a number, the number is interpreted as a line number within the file and point is moved to the beginning of that line. If the line number is followed by a colon and another number, then the second number is interpreted as a column number and point is moved to that column number.

shu-vh [Command]

Visit a c or h file in a project. If point is on something that resembles a file name, then visit that file. If the file name is followed by a colon and a number then go to that line in the file. If the line number is followed by a colon and a number then use the second number as the column number within the line.

shu-visit-project-and-tags *proj-dir* [Function]

Visit a project file, make a C project from the contents of the whole file, and load that tags table from the tags file in the specified directory. This function uses the existing tags table, whereas *shu-setup-project-and-tags* creates a new tags table.

shu-vj [Command]

Visit a c or h file in a project. Ignore any text that point is on and visit the file typed in the completion buffer.

shu-which-c-project

[Command]

(Alias: which-c-project)

Identify the current project by putting into a project buffer the name of the file from which the project was derived as well as the name of all of the directories in the project. Then switch to that buffer. The idea is to invoke this function, look at the results in that buffer, and then quit out of the buffer.

11 shu-cpp-token

Functions to parse a region of C++ code and return a list of tokens found therein. The returned list is a list of token-info, whose structure is shown below.

The two top level functions in this file are `shu-cpp-tokenize-region` and `shu-cpp-reverse-tokenize-region`. The former returns a list of tokens with the first token in the list being the first token found. The latter function returns the reverse of the former.

The tokenized lists are used by the functions in `shu-cpp-match.el` and `shu-match.el` to find and retrieve patterns in the token lists.

11.1 List of functions by alias name

A list of aliases and associated function names.

`parse-region start end` [Command]
(Function: `shu-cpp-parse-region`)

Parse the region between *start* and *end* into a list of all of the C++ tokens contained therein, displaying the result in the Shu unit test buffer.

`reverse-parse-region start end` [Command]
(Function: `shu-cpp-reverse-parse-region`)

Reverse parse the region between *start* and *end* into a list of all of the C++ tokens contained therein, displaying the result in the Shu unit test buffer.

11.2 List of functions and variables

List of functions and variable definitions in this package.

`shu-cpp-adjust-template-parameters token-list` [Function]

Turn each set of template parameters in a reverse parsed list (anything between “>” and “<” into a separate token of type *shu-cpp-token-type-tp*. e.g., the five separate tokens “>”, “double”, “,”, “int”, “<” will be turned into one new token of type *shu-cpp-token-type-tp* whose token value is “int, double”.

`shu-cpp-compare-tlist-sans-comment token-list1` [Function]

Compare the two lists of TOKEN-INFO skipping comments and stopping at the end of the shortest one. The purpose of this function is to determine if two bits of reverse parsed code have the same suffix.

shu-cpp-compare-token-info *token-info1 token-info2* [Function]
 Compare the two instances of TOKEN-INFO, returning true if their contents are the same.

shu-cpp-compare-token-info-sans-pos *token-info1* [Function]
 Compare the two instances of TOKEN-INFO, returning true if their contents are the same. Do not include the start or end points in the comparison.

shu-cpp-copy-token-info *token-info* [Function]
 Return a deep copy of the given *token-info*.

shu-cpp-get-comment *start end* [Function]
 Get the comment that starts at point. If it starts with *//*, get to end of line. If it starts with */{*},* skip to terminating *{*/.}* If there is no terminating *{*/.}* in the region, create a TOKEN-INFO with the appropriate error message in it.

shu-cpp-get-operator-token *length* [Function]
 Fetch the C++ operator that starts at point. *length* is the number of characters in the operator, which is either 1, 2, or 3.

shu-cpp-get-quoted-token *start end* [Function]
 Find the token in the buffer between *start* and *end* that is terminated by an unescaped quote. On entry, point must be positioned on the quote that starts the string. The appropriate error message is returned if there is no unescaped quote before the end of the current line. If the character under point is not a quote start character, nil is returned.

shu-cpp-get-unquoted-token *start end* [Function]
 Find the unquoted token in the buffer that starts at point. The token is terminated either by the position of *end* or by the regular expression that defines the end of an unquoted token.

shu-cpp-is-reverse-token-list-balanced *token-list* [Function]
close-char
 Return t if a token-list contains matched pairs of *open-char* and *close-char*. If imbalance is present, print error message and return nil. Typically *open-char* might be a left parenthesis and *close-char* might be a right parenthesis. Or they might be “<” and “>”, or any other pair types. Note that this function returns t if there are no occurrences of *open-char* and *close-char*

shu-cpp-keywords	[Constant]
alist of C++ key words up to approximately C++20	
shu-cpp-keywords-hash	[Variable]
The hash table of C++ key words	
shu-cpp-make-token-info <i>token token-type spoint</i>	[Function]
&optional <i>error-message</i>	
Pack the supplied arguments into a TOKEN-INFO and return the TOKEN-INFO.	
shu-cpp-operator-start	[Constant]
Define the set of characters that start C++ operators	
shu-cpp-operator-start-chars	[Constant]
Define the set of characters that start C++ operators	
shu-cpp-operators-one	[Constant]
Define the set of one character operators. Note that we include ; as an operator, even though, strictly speaking, it is not an operator.	
shu-cpp-operators-three	[Constant]
Define the set of three character C++ operators	
shu-cpp-operators-two	[Constant]
Define the set of two character C++ operators	
shu-cpp-parse-region <i>start end</i>	[Command]
(Alias: parse-region)	
Parse the region between <i>start</i> and <i>end</i> into a list of all of the C++ tokens contained therein, displaying the result in the Shu unit test buffer.	
shu-cpp-remove-template-parameters <i>token-list</i>	[Function]
<i>preserve-template</i>	
Remove from the token-list any template parameters (anything between “>” and its matching “<”). In addition, adjust the end point of the token immediately prior to the template parameter to be that of the endpoint of the template parameter. Thus something like the following:	
Mumble <int, double>	
becomes the token “Mumble” with a length of 19. If <i>preserve-template</i> is	

true, then we change the token that contains the type name by copying the template parameters into it. If the type name token was “Mumble”, then the token itself is changed to “Mumble<int, double>”. The tokens that represent the template parameters are removed from the token list in either case. This eliminates any comma that does not immediately follow a parameter name. As we scan the reverse ordered token list, any comma that we find immediately precedes a variable name in the parameter list. There may be intervening operators and comments. But once we find a comma, the next unquoted token is the variable name.

shu-cpp-replace-token-info *token-info token* [Function]
spoint epoint &optional

Replace the supplied arguments in the given *token-info* and return the *token-info*.

shu-cpp-reverse-parse-region *start end* [Command]
 (Alias: reverse-parse-region)

Reverse parse the region between *start* and *end* into a list of all of the C++ tokens contained therein, displaying the result in the Shu unit test buffer.

shu-cpp-reverse-tokenize-region *start end* [Function]
limit

Scan the region between *start* and *end* to build a list of tokens that represent the C++ code in the region. Return a cons cell with two items in it. The car of the cons cell is a token-info that represents a parse error. The cdr of the cons cell is the list of tokens. This list is incomplete if the car of the cons cell is not nil. The optional *limit* argument is used to bound the scan as follows. When we have added to the list the first token that is beyond the point specified by *limit*, we stop the scan.

shu-cpp-reverse-tokenize-region-for-command *start* [Function]
&optional

Reverse tokenize the region between *start* and *end* into a list of all of the C++ tokens contained therein, displaying any error message, if there is one. If no error, return the token list, else return nil

shu-cpp-token-delimiter-chars [Constant]
 List of all of the characters that terminate an unquoted C++ token

shu-cpp-token-delimiter-end [Constant]
 Regular expression to define that which terminates an unquoted token in

C++

`shu-cpp-token-extract-epoint` *token-info* [Function]
Return the end point from an instance of token-info.

`shu-cpp-token-extract-info` *token-info token* [Macro]
spoint epoint
Extract the information out of a token-info

`shu-cpp-token-extract-nesting-level` *token-info* [Function]
Return the nesting level from an instance of token-info.

`shu-cpp-token-extract-spoint` *token-info* [Function]
Return the start point from an instance of token-info.

`shu-cpp-token-extract-token` *token-info* [Function]
Return the token from an instance of token-info.

`shu-cpp-token-extract-type` *token-info* [Function]
Return the token type from an instance of token-info.

`shu-cpp-token-find-spanning-info-token` *token-list* [Function]

Find the token-info in *token-list* that spans *here-point*, if any. If there is no such token-info return nil. If there is such a token-info, return a cons cell whose car is the spanning token-info and whose cdr is a pointer to the next token-info in the tlist.

`shu-cpp-token-first-non-comment` *tlist* [Function]
tlist points to a list of token-info. Return *tlist* pointing to the next token-info that does not hold a comment. If you are scanning through a list of tokens, it is not uncommon to want to skip all of the comments. Use this at the bottom of the loop in place of the usual “`setq tlist (cdr tlist)`”.

i.e.,

```
(while tlist
  ...
  (setq tlist (cdr tlist)))
```

becomes

```
(setq tlist (shu-cpp-token-first-non-comment tlist))
(while tlist
  ...
  (setq tlist (shu-cpp-token-first-non-comment tlist)))
```

and you will scan through the list without seeing any comments.

shu-cpp-token-info-replace-epoint *token-info* [Function]

Replace the EPOINT of *token-info* with *new-epoint*

shu-cpp-token-info-replace-token *token-info* [Function]

Replace the TOKEN of *token-info* with *new-token*, returning the modified *token-info*

shu-cpp-token-internal-parse-region *func start end* [Function]

Internal function to do a forward or reverse parse of the region between *start* and *end*. *func* holds the function to be invoked to do the parse. This would be either *shu-cpp-tokenize-region* or *shu-cpp-reverse-tokenize-region*. Once the parse is complete, the token list is shown in the Shu unit test buffer. If any error is detected, it is displayed at the point at which the error was detected.

shu-cpp-token-internal-tokenize-region-for-command *func end* [Function]

Internal function to do a forward or reverse parse of the region between *start* and *end*. *func* holds the function to be invoked to do the parse. This would be either *shu-cpp-tokenize-region* or *shu-cpp-reverse-tokenize-region*. Once the parse is complete, we check to see if an error was detected. If an error was detected we go to the error point and show the error. Then we return nil to the caller. If no error was detected, we return the token-list to the caller. This is a convenient way for a command to get the token-list and not have to do anything to display an error message if an error is encountered. The command calls this function and simply exits if nil is returned, knowing that the error message has already been displayed.

shu-cpp-token-is-comment *token-info* [Function]

Return true if the token-info represents a comment.

shu-cpp-token-next-non-comment *tlist* [Function]

tlist points to a list of token-info. Return *tlist* pointing to the next token-info that does not hold a comment. If you are scanning through a list of tokens, it is not uncommon to want to skip all of the comments. Use this at the bottom of the loop in place of the usual “setq tlist (cdr tlist)”.

i.e.,

```
(while tlist
  ...
  (setq tlist (cdr tlist)))
```

becomes

```
(while tlist
  ...
  (setq tlist (shu-cpp-token-next-non-comment tlist)))
```

and you will scan through the list without seeing any comments.

shu-cpp-token-set-alias [Function]
Set the common alias names for the functions in shu-cpp-token. These are usually the same as the function names with the leading shu- prefix removed.

shu-cpp-token-set-nesting-level *token-info* [Function]

Set the nesting level in an instance of token-info.

shu-cpp-token-show-token-info *token-info* **&optional** [Function]

Show the data returned by one of the functions in this file that scans for tokens.

shu-cpp-token-show-token-info-buffer *token-info gb* [Function]
title

Show the data returned by one of the functions in this file that scans for tokens.

shu-cpp-token-string-token-info *token-info* [Command]
Return a string that represents the contents of the token-info.

shu-cpp-token-token-type-name *token-type* [Function]

Return the name of a token type.

shu-cpp-token-type-cc [Constant]

Token type that indicates a line in which the first non-blank item is a comment that starts in a column greater than or equal to the column defined by `shu-cpp-comment-start`. This is known as a code comment with no code present.

shu-cpp-token-type-ct [Constant]

Token type that indicates a comment

shu-cpp-token-type-kw [Constant]

Token type that indicates a C++ key word

shu-cpp-token-type-op [Constant]

Token type that indicates an operator

shu-cpp-token-type-qt [Constant]

Token type that indicates a quoted string

shu-cpp-token-type-tp [Constant]

Token type that indicates a template parameter. The standard parsing does nothing with template parameters. Something like “<int>” is simply turned into three separate tokens, “<”, “int”, and “>” (or “>”, “int”, and “<” in a reverse parse). But some of the other transform functions will turn this list of tokens into the single template parameter “int”

shu-cpp-token-type-uq [Constant]

Token type that indicates an unquoted token

shu-cpp-tokenize-region *start end* **&optional** *limit* [Function]

Scan the region between *start* and *end* to build a list of tokens that represent the C++ code in the region. Return a cons cell with two items in it. The car of the cons cell is a token-info that represents a parse error. The cdr of the cons cell is the list of tokens. This list is incomplete if the car of the cons cell is not nil. The optional *limit* argument is used to bound the scan as follows. When we have added to the list the first token that is beyond the point specified by *limit*, we stop the scan.

shu-cpp-tokenize-region-for-command *start end* *limit* [Function]

Tokenize the region between *start* and *end* into a list of all of the C++ tokens contained therein, displaying any error message, if there is one. If no error,

return the token list, else return nil

shu-cpp-tokenize-show-list *token-list* **&optional** [Function]

Undocumented

shu-cpp-tokenize-show-list-buffer *token-list gb* [Function]
title

Undocumented

shu-get-cpp-keywords-hash [Command]
Return a hash table containing all of the C++ key words.

12 shu-keyring

This is a set of functions for maintaining and querying a keyring of names, URLs, users IDs, passwords, and related information that are maintained in an external keyring file.

Functions allow you to find a keyring entry by name and to put one piece of its information, such as user ID or password, in the clip board, from which it may be pasted into a browser or other application.

The keyring file may be encrypted with GPG. As of emacs 23, the EasyPG package is included with the emacs distribution. When you tell emacs to open a file that has been encrypted with GPG, you are prompted for the passphrase and the file is decrypted into memory.

The file keyring.txt in the usr directory is an example of a small keyring file that has not been encrypted. Each entry in the file consists of a set of name value pairs. Each value may be enclosed in quotes and must be enclosed in quotes if it contains embedded blanks.

A single set of name value pairs starts with an opening "<" and is terminated by a closing ">".

Here is an example of a set of name value pairs:

```
< name="Fred email" url=mail.google.com id=freddy@gmail.com pw=secret />
```

The names may be arbitrary but there are six names that are recognized by the functions in this package. They are:

acct represents an account number

id represents a user ID

name represents the name of the entry. This is the key that is used to find the entry. If no name is given, then the name of the URL is used. If the URL starts with "www.", the "www." is removed to form the name. An entry that has no name and a URL of "www.facebook.com" would have an auto generated name of "facebook.com".

pin represents a pin number

pw represents a password

url represents a URL

To use a keyring file, place the following lines in your .emacs file:

```
(load-file "/Users/fred/.emacs.d/shu-base.elc")
(load-file "/Users/fred/.emacs.d/shu-nvplist.elc")
(load-file "/Users/fred/.emacs.d/shu-keyring.elc")
(shu-keyring-set-alias)
(setq shu-keyring-file "/Users/fred/shu/usr/keyring.txt")
```

replacing `"/Users/fred/shu/usr/keyring.txt"` with the path to your keyring file. All of the shu functions require shu-base.

If using the sample keyring file, Fred can now use this to log onto his gmail account as follows.

Type `M-x krurl`. This prompts for the name of the desired key. Type `"Fred em"` and hit `TAB` to complete. This fills out the name as `"Fred email"` and puts the URL `"mail.google.com"` into the clip board. Open a browser and paste the URL into it to go to gmail. At gmail, select login. In emacs type `M-x krid`. When prompted for the key, use the up arrow to retrieve the last key used, which will be `"Fred email"`. This puts `"freddy@gmail.com"` into the clip board for conveniently pasting into the gmail widow. To obtain the password, type `M-x krpw`. This puts the password into the clip board from which it may be pasted into the gmail widow.

12.1 List of functions by alias name

A list of aliases and associated function names.

kracct [Command]

(Function: shu-keyring-get-acct)

Find the account for an entry in the keyring file. This displays the entry in the message area and puts the password into the kill ring so that it can be yanked or pasted into the application requesting it.

krfn [Command]

(Function: shu-keyring-get-file)

Display the name of the keyring file, if any. This is useful if you are getting unexpected results from some of the query functions that look up keyring information. Perhaps the unexpected results come from the fact that you are using the wrong keyring file.

krid [Command]

(Function: shu-keyring-get-id)

Find the User Id for an entry in the keyring file. This displays the entry in the message area and puts the user Id into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

krpin [Command]

(Function: shu-keyring-get-pin)

Find the pin for an entry in the keyring file. This displays the entry in the message area and puts the pin into the kill ring so that it can be yanked into

a buffer or pasted into the application requesting it.

krpps [Command]
(Function: shu-keyring-get-passphrase)
Doc string.

krpw [Command]
(Function: shu-keyring-get-pw)
Find the password for an entry in the keyring file. This displays the entry (without the password) in the message area and puts the password into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

krurl [Command]
(Function: shu-keyring-get-url)
Find the url for an entry in the keyring file. This displays the entry in the message area and puts the url into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

krvf [Command]
(Function: shu-keyring-verify-file)
Parse and verify the keyring file, displaying the result of the operation in the keyring buffer (`{**shu-keyring**}`). If one of the queries for a url or other piece of information is unable to find the requested information, it could be that you have the wrong keyring file or that there is a syntax error in the keyring file. `shu-keyring-get-file` (alias `krfn`) displays the name of the keyring file. This function parses the keyring file. After the operation. look into the keyring buffer (`{**shu-keyring**}`) to see if there are any complaints about syntax errors in the file.

set-passphrase *phrase* [Command]
(Function: shu-keyring-set-passphrase)
Function to read and set the external pass phrase.

12.2 List of functions and variables

List of functions and variable definitions in this package.

shu-keyring-account-name [Constant]
Key word that denotes a name.

- shu-keyring-add-values-to-index** *index vlist item* [Function]
 Add a set of keys *vlist* to *index* for *item*. Keys within the item are filtered for duplicates. But this does not prevent two different items from sharing the same key, although it would be unusual in a keyring.
- shu-keyring-buffer-name** [Constant]
 The name of the buffer into which keyring diagnostics and messages are recorded.
- shu-keyring-clear-index** [Function]
 This is called from after-save-hook to clear the keyring index if the keyring file is saved. The keyring index is built the first time it is needed and kept in memory thereafter. But we must refresh the index if the keyring file is modified. The easiest way to do this is to clear the index when the keyring file is modified. The next time the index is needed it will be recreated.
- shu-keyring-external-passphrase** [Variable]
 Holds the external passphrase for the keyring file. This allows the user to type the passphrase at the beginning of an emacs session. Once this is set it can then be put into kill ring by shu-keyring-get-passphrase.
- shu-keyring-file** [Custom]
 Text file in which urls, names, and passwords are stored.
- shu-keyring-find-index-duplicates** *index* [Function]
 Find any duplicates in the keyring index. When the index is built we filter duplicate keys for the same item. But there could be two different items with the same key. This function returns TRUE if two or more items have the same key. The index must be in sorted order by key value before this function is called.
- shu-keyring-get-acct** [Command]
 (Alias: kracct)
 Find the account for an entry in the keyring file. This displays the entry in the message area and puts the password into the kill ring so that it can be yanked or pasted into the application requesting it.
- shu-keyring-get-field** *name* [Function]
 Fetch the value of a named field from the keyring. Prompt the user with a completing-read for the field that identifies the key. Use the key to find the item. Find the value of the named key value pair within the item. Put the

value in the kill-ring and also return it to the caller.

shu-keyring-get-file [Command]
(Alias: krfn)

Display the name of the keyring file, if any. This is useful if you are getting unexpected results from some of the query functions that look up keyring information. Perhaps the unexpected results come from the fact that you are using the wrong keyring file.

shu-keyring-get-id [Command]
(Alias: krid)

Find the User Id for an entry in the keyring file. This displays the entry in the message area and puts the user Id into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

shu-keyring-get-passphrase [Command]
(Alias: krpps)
Doc string.

shu-keyring-get-pin [Command]
(Alias: krpin)

Find the pin for an entry in the keyring file. This displays the entry in the message area and puts the pin into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

shu-keyring-get-pw [Command]
(Alias: krpw)

Find the password for an entry in the keyring file. This displays the entry (without the password) in the message area and puts the password into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

shu-keyring-get-url [Command]
(Alias: krurl)

Find the url for an entry in the keyring file. This displays the entry in the message area and puts the url into the kill ring so that it can be yanked into a buffer or pasted into the application requesting it.

shu-keyring-history [Variable]
The history list used by completing-read when asking the user for a key to an entry in the keyring file.

shu-keyring-id-name	[Constant]
Key word that denotes a user ID.	
shu-keyring-in-index <i>index item value</i>	[Function]
Return true if the <i>index</i> already contains the <i>value</i> for this <i>item</i> .	
shu-keyring-index	[Variable]
The variable that points to the in-memory keyring index.	
shu-keyring-name-name	[Constant]
Key word that denotes a name.	
shu-keyring-parse-keyring-file	[Command]
Parse the keyring file and create the in-memory index if the keyring file contains no duplicate keys.	
shu-keyring-pin-name	[Constant]
Key word that denotes a PIN.	
shu-keyring-pw-name	[Constant]
Key word that denotes a password.	
shu-keyring-set-alias	[Function]
Set the common alias names for the functions in shu-keyring. These are generally the same as the function names with the leading shu- prefix removed. But in this case the names are drastically shortened to make them easier to type.	
shu-keyring-set-passphrase <i>phrase</i>	[Command]
(Alias: set-passphrase)	
Function to read and set the external pass phrase.	
shu-keyring-show-index <i>index</i>	[Function]
Print the keyring index	
shu-keyring-show-name-url <i>type item</i>	[Function]
Show in the message area the name, url, or both of a keyring entry. Also prefix the message with the upper case type, which is the type of the entry that has been placed in the clipboard, (PW, ID, etc.)	
shu-keyring-update-index <i>index item</i>	[Function]
Extract the keys from a keyring item and add them to the keyring index.	

shu-keyring-url-name [Constant]

Key word that denotes a URL.

shu-keyring-values-to-string *values* [Function]

Turn a list of values into a single string of values separated by slashes.

shu-keyring-verify-file [Command]

(Alias: `krvf`)

Parse and verify the keyring file, displaying the result of the operation in the keyring buffer (`{**shu-keyring**}`). If one of the queries for a url or other piece of information is unable to find the requested information, it could be that you have the wrong keyring file or that there is a syntax error in the keyring file. `shu-keyring-get-file` (alias `krfn`) displays the name of the keyring file. This function parses the keyring file. After the operation. look into the keyring buffer (`{**shu-keyring**}`) to see if there are any complaints about syntax errors in the file.

13 shu-match

Functions that use functions in shu-cpp-match.el

13.1 List of functions by alias name

A list of aliases and associated function names.

find-all-variables [Command]
(Function: shu-match-find-variables)

Find what might be all of the variable declarations in a header file by doing a reverse tokenized scan looking for all occurrences of operator “;” followed by something that matches the regular expression for a C++ name. Then take each line that matches and put it in the buffer “**{**shu-vars**}**”.

13.2 List of functions and variables

List of functions and variable definitions in this package.

remove-class-duplicates *proc-classes log-buf* [Function]

proc-classes is the alist of classes that we will process. The car of each item is the containing namespace. The cdr of each item is the list of class names contained within the namespace. The original class list may have had duplicate class names within a given namespace. We may also have added class names to a given namespace from processing one or more “using name” statements.

For example, if the lass list contained “std . set map string” and we also processed a “using std::string” statement, the code that processed that statement would have blindly added “string” to the list of classes in the namespace “std”. The list of classes for namespace “std” would then contain “string set map string”.

This function removes any duplicate class names within a given namespace. This is necessary because we are about to invert the class list to produce a hash table in which the key is the class name and the value is the enclosing namespace name. This operation will fail if a given namespace contains duplicate class names.

shu-cpp-match-classname-forms [Constant]

Match `name::name` or `name::name::name` or `name::name::name::name`, etc. The double colons make up the return value so that you can tell where one name ends and another starts.

shu-cpp-match-colon-name [Constant]
 A repeating side list to match zero or more instances of `{:: <name>}`

shu-cpp-match-colon-name-return [Constant]
 A repeating side list to match one or more instances of `{:: <name>}` with the double colons making up part of the return value.

shu-cpp-match-many-using-list [Constant]
 These two lists match what may follow the key word “using”. The first list matches the key word “namespace” followed by any of the different name types that may follow “using namespace”. The second list matches any of the name forms that may following the key word “using” when it is not followed by the key word “namespace”.

shu-cpp-match-namespace-forms [Constant]
 ;

shu-cpp-match-namespace-list [Constant]
 namespace

shu-cpp-match-namespace-list-single [Constant]
 namespace

shu-cpp-match-some-include [Constant]
 A list of match-info that matches “`#include <name>`”.

shu-cpp-match-using-forms [Constant]
 These two lists match the form of name that can follow a “using” directive that is not a “using namespace” directive. This is either `<name>` or `<name>::<name>`, `<name>::<name>::<name>`, etc. The first list above matches `<name>` followed by semicolon.. The second list matches `<name>` followed by a looping side list for zero or more occurrences of “`::`” followed by `<name>`

shu-cpp-match-using-list-single [Constant]
 This is a single list that is the top level list for matching anything that may follow the key word “using”.

shu-cpp-rmv-using *class-list* **&optional** *top-name* [Function]
 Remove “using namespace” directives from a C++ file, adding the appropriate namespace qualifier to all of the unqualified class names. *class-list* is an a-list in which the car of each entry is a namespace and the cdr of each entry

is a list of class names. Here is an example of such an a-list:

```
(list
  (cons 'std' (list 'set' 'string' 'vector'))
  (cons 'world' (list 'Hello' 'Goodbye'))))
```

top-name, if present is a higher level namespace. Given a top level namespace of “WhammoCorp”, then the following line:

```
using namespace WhammoCorp::world;
```

would be interpreted as though it had been written:

```
using namespace world;
```

shu-match-add-names-to-class-list *un-list* [Function]
proc-rlists

&op-

tional

un-list is the list of rlists that represent the “using name” statements. *proc-classes* is the class list we will be using to do the processing. *proc-rlists* is the set of rlists that represents the “using namespace” statements. This function adds the “using name” directives, if any, to both *proc-classes* and *proc-rlists*. It returns a cons cell in which the car is the modified *proc-classes* and the cdr is the modified *proc-rlists*.

shu-match-erase-using *proc-rlists* *log-buf* [Command]

proc-rlists is the set of rlists we are processing that represent all of the “using namespace” and “using name” statements. This function replaces all of those statements in the buffer with whitespace. This is done in order to preserve the positions of all other items in the buffer.

shu-match-fetch-include-hash-table *token-list* [Function]

token-list is the list of tokenized text from the buffer. *class-ht* is the hash table that holds as its keys the names of all of the classes that we will process.

This function finds all of the names in `#include` statements whose included file name matches a class name in *class-ht*. It then builds and returns a new hash table in which the key is the name of an included file whose name

matches a class name and whose value is a list of the spoints of the included file names.

It is a list of spoints, as opposed to a single point, because an include statement with a given file name may appear several times.

For example, in the following code:

```
#include <set>
#include <map>
#
    include
        <set>

using namespace std;

set      x;
map      y;
```

the member variables x and y should be changed to std::set and std::map respectively, but none of the occurrences of set or map in the include statements should be changed.

shu-match-find-all-some-include *token-list* [Function]

Given a token list, return a list of tokens that represent all of the simple names found in `#include < name >`, where “name” is a C++ name. This search will neither find nor return a name with a . in it. This is a limited search designed to find names that might be mistaken for class names to be qualified.

For example, if we are removing the namespace “std”, then one of the names we may wish to qualify is “set”. But if we encounter “`#include <set>`”, we do not want to transform that into “`#include <std::set>`”.

Names of include files delimited by quotes will not be seen in the scan because those are inside strings. So we only want to find names in include statements that are delimited by angle brackets and do not include a . in them.

shu-match-find-all-using-internal *token-list* [Function]

Given a token list, return two different lists. The first is a list of all “using namespace” statements. The second is a list of all “using” statements that are not “using namespace” statements. “using namespace std;” is an example of the first type. “using std::string” is an example of the second type. The

return value of this function is a single cons cell in which the cdr points to the first list and the car points to the second list. If neither list is present, then the return value is nil.

shu-match-find-any-using-internal *token-list* [Function]

Given a token list, return a list of all “using namespace” statements and all “using” statements that are not “using namespace” statements. “using namespace std;” is an example of the first type. “using std::string” is an example of the second type. This is used to find out whether or not a file of code contains any such statements and to identify them.

shu-match-find-semi-names *token-list* [Function]

With a match list that is a semi-colon followed by the regular expression for a C++ name, do a reverse tokenized match for all occurrences, then take each line that holds a match and put it into the buffer “{**shu-vars**}”

This version (11 Nov 2019) adds one more check. If the token immediately in front of the semi-colon is “}”, then we assume this is the last line of code in an inline function, in which case it is not a variable declaration. This helps to weed out some of the extraneous ones but not all of them.

shu-match-find-unqualified-class-names *class-ht* [Command]
token-list

buf

Go through all of the tokens looking at any unquoted token that is in the hash table of unqualified class names. When an instance of a class name is found, check to see if it is preceded by “::”, “.”, or “->”. “::” indicates that it is probably qualified. “.” or “->” indicate that it is probably a function name.

Check also to see if it is followed by “)” or “[“, which probably indicates that it is a variable name.

Next, check to see if it is wrapped in an #include statement.

If it survives all of those checks, it is probably an unqualified class name, in which case its token-info is pushed onto a list. The list is one of the values returned from this function. Since each token-info is pushed onto the list, the list is returned in reverse order. i.e., the last token in the file is the first in the list.

At this point it is possible to visit each token in the list, which is in reverse order in the file, look up each token, and insert in front of it its qualifying namespace.

The other return value from this function is the count of the number of unquoted tokens that were looked up in the hash table of all of the class names, *class-ht*.

The actual return value from this function is a cons cell whose car is the symbol search count and whose cdr is the list of tokens that represent unqualified class names to be qualified.

shu-match-find-variables [Command]
(Alias: find-all-variables)

Find what might be all of the variable declarations in a header file by doing a reverse tokenized scan looking for all occurrences of operator “;” followed by something that matches the regular expression for a C++ name. Then take each line that matches and put it in the buffer “***shu-vars***”.

shu-match-get-start-end-pos *rlist* &**optional** [Function]

Given an *rlist* return the beginning and end positions. The beginning position is the position of the first character of the first token. The end position is the position of the last character of the last token. These two are returned as a cons cell whose car is the beginning position and whose cdr is the end position. If the optional *whole-lines* is true, the start position is that of the beginning of the line on which the start falls and the end position is that of the end of the line on which the end falls.

shu-match-increment-class-count *alist class-name* [Command]
alist is an alist whose key is a *class-name* and whose cdr is a count of the number of times that class name has been found. This function increments the count by one.

shu-match-internal-rmv-using *class-list log-buf* [Function]
top-name

This is an overview of the entire process used to remove both the using namespace statements (e.g., “using namespace std;”) and the using statements (e.g., “using std::string;”).

The input is a class list which is an alist in which the key for each entry is a namespace name and the value for each entry is a list of class names that are qualified by the namespace name. This is an example of a class list:

```
(list (cons ‘‘std’’ (list ‘‘set’’ ‘‘string’’ ‘‘vector’’)) (cons ‘‘world’’  
  (list ‘‘Hello’’ ‘‘Goodbye’’)))
```


The first step is to tokenize the entire buffer and then use match functions to find each instance of “using namespace” and each occurrence of “using name”. Each of these statements is represented by a list of token-info.

If any “using namespace” statements are found, they are merged with the class list to form a new class list that contains only those namespaces for which a “using namespace” statement was found in the buffer.

A separate list is kept of the “using namespace” statements that have no corresponding entries in the class list. This is printed out later as a diagnostic message.

The next step is to merge in the namespaces from the “using name” statements. Each “using name” statement contributes one namespace name and one class name.

The original class list might have contained duplicate class name instances within a given namespace. The merging in the of the “using name” statements may also have created duplicate class names within a namespace, so the next thing we do is remove any duplicate class names within a given namespace. For example, if the input class list contained “std . string set map” and the buffer contains both “using namespace std;” and “using std::string;”, the newly merged class list will contain the class name “string” twice under the namespace “std”.

Once we have the updated class list with duplicates removed, we create two new data structures. One is a hash table in which the key is a class name and the value is the name of the namespace that qualifies that class name. This will be used to determine if an unquoted token is an unqualified class name. The other is an alist in which the key is a class name and the value is zero. This will be used to count the replacement count for each class.

In creating the hash table, we may discover that one class name maps to more than one namespace name. If that happens there is an unresolvable ambiguity and the operation must cease.

The next step is to remove from the token list (from the tokenized buffer), the lists of token-info that represent the “using namespace” and “using name” statements that we are processing. This prevents any subsequent scan from seeing them again.

Next, we erase the “using namespace” statements and “using name” statements from the buffer. We do this by replacing the statements with an equivalent amount of whitespace, which preserves the positions of all of the other tokens in the buffer.

Some of the class names in the class list may also appear in `#include` statements in the buffer. For example, if we are trying to remove “using namespace std;”, the buffer may well contain `#include <set>` or `#include <map>`. These instances of `set` and `map` are file names. They are not class names that should have the “std” qualifier added to them.

To handle this case we build another hash table. This one contains class names as its key. It is the intersection of class names and names found in include statements. The value of each entry in the hash table is the list of spoints that are the start point for each name within its include statement. It is a list of spoints because the same name may appear in multiple `#include` statements.

Then we go through the token list. Whenever we encounter an unquoted token, we look it up in the hash table to see if this is a class name that we should qualify. If the token exists in the hash table, we then look at its context to see if it really looks like a class name.

If the putative class name is preceded by any of “::”, “.”, or “->”, then we assume that it is either a qualified class name or a function name. There are other various checks that can be found in the function `shu-match-find-unqualified-class-names`. One of the checks is to see if it matches an instance contained in an `#include` statement, which is done with the hash of include names described above.

Each time we find an unqualified class name, we push its token-info onto a new list of class names that need to be qualified. Note that we use push so the list is backwards with respect to buffer order. The first item in the list is that last unqualified class name in the buffer.

This means that we can use the list directly to add the namespace qualifier to each unqualified class name. Since we are going through the buffer backwards, adding a qualifier to an unqualified class name does not change the position of any other unqualified class names in the buffer.

While adding namespace qualifiers to all of the unqualified class names, we also accumulate a change count for each class name.

When all of the unqualified class names have been qualified, we display the final change counts in a buffer and emit a message with the sum of all the change counts.

`shu-match-line-code stmt spos` [Function]
stmt is a string of text taken from the buffer. *spos* is the start position of the string of text. This function splits the lines of text into separate lines, each

one prefixed with the line number where it appears in the file.

For example, if the string of test is “Hello.” and it starts at line 392 of the file, this function will return “ 392. Hello.” with a newline appended to the end of the line.

shu-match-make-class-hash-table-internal *proc-classes* [Function]

proc-classes is the alist of all of the namespaces and classes that we will process, with the namespace name being the key and a list of class names within the namespace name as the value.

This function builds a hash table that inverts the alist. Each entry in the hash table has a class name as the key with the name of the enclosing namespace as the value. If two class names map to the same enclosing namespace name, then there is an unresolvable ambiguity that must terminate the operation. If that is the case, diagnostic messages are placed into the log buffer and a nil value is returned.

shu-match-make-count-alist-from-hash *class-ht* [Command]

Create an alist in which the key is a class name and the value is zero. This alist will be used to generate a count of number of items changed per class name.

shu-match-make-include-hash-table *incl-list* [Function]

incl-list is the list of token-info, each of which represents a name found in an include statement. *class-ht* is the hash table whose keys are the names of the classes that we will be searching for.

For each name that appears in both *incl-list* and *class-ht*, create a new entry in a new hash table whose key is the name of the class that has been found in one or more include statements and whose value is a list of the points in which the class name has been found in one or more include statements.

Whenever we find something that appears to be an unqualified class name, we can look in this hash table to see if this occurrence of the name is one that appears in an include statement and avoid adding a namespace qualifier if that is the case.

shu-match-merge-namespaces-with-class-list *class-list* [Function]
log-buf

name

Merge the *uns-list* with the *class-list*. Return a cons-cell pointing to two lists. The first is the list of classes from the class list that have corresponding “using namespace” directives in the buffer. The second is the lists of rlists that represent each using namespace directive that we will process. The updated class list will be used to identify class names to be qualified and the namespaces with which to qualify them.. The rlists representing the “using namespace” statements will be used to remove the “using namespace” statements from the buffer.

shu-match-qualify-class-names *class-ht count-alist* [Function]
np-rlists *symbol-*
count log-buf

class-ht is the hash table that maps a class name to its containing namespace name. *count-alist* is the alist that counts the number of times each class name has been qualified by its enclosing namespace. *clist* is the list of token-info, each of which represents an unqualified class name. The list is in reverse order, which is important. It means that one can add a qualification to one class name in the list without changing the location of any other class names, which are above the current one in the buffer. *np-rlists* is a list of rlists, each of which represents a “using namespace” statement for which there is no corresponding entry in the class list. There are the “using namespace” statements that we will not be processing..

This function goes to the position of each unqualified class name, finds its containing namespace in the hash table, and inserts the containing namespace followed by “::” in front of the unqualified class name.

After it inserts the qualifying namespace, it increments in *count-alist* the number of times that the class name was explicitly qualified.

shu-match-remove-proc-rlists *token-list* [Function]
log-buf

token-list is the original token list. *proc-rlists* is the set of rlists that represents the set of statements we will be processing. This function removes from *token-list*, all of the items that are contained in the rlists in *proc-rlists*. This is because we do not want a subsequent scan of the token list to include any of the items in the statements we are processing.

The return value from this function is a cons cell whose car is the trimmed *token-list* and whose cdr is the sorted *proc-rlists*, which has been sorted by the start position of each rlist.

shu-match-rmv-might-be-include *incl-ht token* [Function]

incl-ht is a hash table whose key is a name that was found in an include statement and whose value is a list of all of the spoints of all of the occurrences of the name in include statements. It is a list because the same name may be included multiple times.

If the current token matches one of the names in *incl-ht* and the spoint of the token is a member of the list of spoints in the entry in *incl-ht*, then the current name is enclosed in an include statement and should not have a namespace qualifier added to it.

This function returns true if the name is wrapped in an include statement.

shu-match-rmv-show-class-count *count-alist* [Function]
np-rlists ns-lines

count

symbol-

Put into the log buffer the count of class names that were qualified.

shu-match-set-alias [Function]

Set the common alias names for the functions in shu-match, These are generally the same as the function names with the leading shu- prefix removed.

shu-match-using-namespace-string *rlist &optional* [Function]

Given an *rlist* that contains a “using namespace” statement, return the string that is the fully qualified namespace name. If the first part of the name is the optional *top-name*, it is omitted from the final result.

shu-match-using-string *rlist &optional top-name* [Function]

Given an *rlist* that contains a “using” statement (as opposed to “using namespace”), return two strings. One is the class name. The other is the fully qualified namespace name. For example, if the statement is “using std::string,” the fully qualified namespace name is “std” and the class name is “string”. The two strings are returned in a cons cell whose car is the namespace name and whose cdr is the class name.

14 shu-misc

A miscellaneous collection of useful functions

14.1 List of functions by alias name

A list of aliases and associated function names.

add-alexandria [Command]

(Function: shu-add-alexandria)

Add Alexandria coverage to a git repository. This function first checks to ensure that a README.md file exists that does not contain an Alexandria badge and that a Doxyfile does not exist. If those two conditions are met, an Alexandria badge is added to the bottom of the README.md file, a Doxyfile is created, and some of the tags in the Doxyfile are set to reasonable defaults. An ALEXANDRIA_DOC_DEPENDENCIES tag is added to the end of the Doxyfile as a comment. If this function succeeds, it returns true, else nil. The return value may be used by batch mode functions that want to call this function and report whether or not it succeeded.

add-alexandria-badge [Command]

(Function: shu-add-alexandria-badge)

Insert an Alexandria badge for the current project.

add-prefix *prefix* [Command]

(Function: shu-add-prefix)

Put a prefix and a space in front of each line in the region. Prompt is issued for the prefix.

af *frame-no* [Command]

(Function: shu-adapt-frame)

Adapt the current frame to the current display by stretching the frame to the full height of the display and putting the top of the frame at the top of the display. This function makes some assumptions about the display geometry based on the current operating system. It assumes that Windows loses five lines for top and bottom tool bars. Mac OS X loses three lines for the top tool bar. Unix loses two lines for something. These numbers should, at some point, be customizable.

With a numeric prefix argument N, the emacs window is positioned N frames from the right hand side of the display. For example, if you open three frames and type into the first frame C-u 1 M-x *shu-adapt-frame*, into the next frame

C-u 2 M-x *shu-adapt-frame*, and into the third frame C-u 3 M-x *shu-adapt-frame*, then the three frames will be grouped together side by side at the right side of the display.

If the prefix argument is large enough that the left side of the frame would be moved past the left side of the display, the window is positioned such that the left edge of the window is aligned with the left edge of the display.

Implementation note:

If this function is called when the left side of the frame is positioned to the left of the leftmost edge of the display, the function FRAME-POSITION returns a negative value for the x coordinate of the frame. The function SET-FRAME-POSITION takes the x and y coordinates of the new position of the top left corner of the frame.

But if x is negative, it specifies the coordinates of the right edge of the frame relative to the right edge of the display. This puts a frame that is very close to the left edge of the display all the way over to the right edge of the display.

The assumption is that a negative x frame position means that the user has positioned the frame just a bit past the left edge and that the desired frame position is actually the leftmost edge of the display.

all-quit [Command]

(Function: shu-all-quit)

Kill all direcd buffers and all buffers that contain a file and are unmodified. It is not uncommon to have dozens of buffers open that are unrelated to the current task and this is a convenience function for closing many buffers that do not need to be open. If the function *shu-clear-c-project* is defined, it is called to clear the current project.

buffer-number-lines [Command]

(Function: shu-buffer-number-lines)

Create a buffer whose name is derived from the file name of the current buffer but with the string “-numbered” added to the name. Thus “foo.cpp” would become “foo-numbered.cpp” Into this new buffer, copy the contents of the current file with each line prefixed with its line number. This is designed for those times when you want to copy snippets of code with the line number in front of each line because you are commenting on code and want the person receiving the comments to see the line number in front of each line.

case-insensitive [Command]

(Function: shu-case-insensitive)

Set the variable `case-fold-search` to `t` to make searches and matches ignore case. I can never remember which way to set `case-fold-search`, hence this simple, little function.

`case-sensitive` [Command]

(Function: `shu-case-sensitive`)

Set the variable `case-fold-search` to `nil` to make searches and matches respect case. I can never remember which way to set `case-fold-search`, hence this simple, little function.

`comma-names-to-letter` [Command]

(Function: `shu-comma-names-to-letter`)

In a list of names, change all occurrences of Lastname, Firstname to an empty Latex letter. Position to the start of the file and invoke once.

`copy-repo` [Command]

(Function: `shu-copy-repo`)

Call *shu-get-repo* to find the path to the repository and put the result in the kill ring.

`de-star` [Command]

(Function: `shu-de-star`)

Remove leading spaces and asterisk from each line in the region. This is useful for editing doxygen comments of the form:

```
/*!  
  
* This is some commentary.  
* This is more commentary, etc.  
\emph{*/}
```

You snip out the middle lines and put them into a text file for formatting and spell-checking. You want to get rid of all of the asterisks until you are done.

This function gets rid of all the asterisks. You can use *shu-add-prefix* to put them back.

`diff-commits` *commit-range* [Command]

(Function: `shu-git-diff-commits`)

In a buffer that is a numbered git log, query for a range string, find the two commits, and put into the kill ring a git diff command specifying the two commits.

For example, given the following two numbered commits:


```
31. commit 38f25b6769385dbc3526f32a75b97218cb4a6754
33. commit 052ee7f4297206f08d44466934f1a52678da6ec9
```

if the commit range specified is either “31.33” or “31+2”, then the following is put into the kill ring:

```
“git diff -b 38f25b6769385dbc3526f32a75b97218cb4a6754..052ee7f4297206f08d44466934f1a52678da6ec9”
```

dup [Command]

(Function: shu-dup)

Insert a duplicate of the current line, following it.

eld [Command]

(Function: shu-save-and-load)

Save and load the current file as a .el file.

fixup-doxyfile [Command]

(Function: shu-fixup-doxyfile)

The current directory is assumed to have the same name as the project for which the Doxyfile was created. This function sets various default values in the Doxyfile. The current buffer is the Doxyfile.

gcm [Command]

(Function: shu-git-insert-git-commit)

Insert at point the name the git command to commit with the commentary held in a file called “why.txt”.

gd [Command]

(Function: shu-gd)

While in dired, put the full path to the current directory in the kill ring

get-pr-url [Command]

(Function: shu-git-get-pr-url)

Put into the kill ring the path required to create a new pull request for the current branch of the current repository.

get-repo [Function]

(Function: shu-get-repo)

When positioned anywhere in a git repository, return the git path to the repository. This is found in .git/config as the url of [remote “origin”]. Return nil if the path cannot be found.

The search is made from the current directory and upwards for the first directory called “.git”.

gf [Command]
(Function: shu-gf)

While in dired, put the full path to the current file in the kill ring

gfc [Command]
(Function: shu-gfc)

While in a file buffer, put both the current line number and column number and the name of the current file into the kill ring in the form of “foo.cpp:123:2”.

gfl [Command]
(Function: shu-gfl)

While in a file buffer, put both the current line number and the name of the current file into the kill ring in the form of “line 1234 of foo.cpp”.

gfn [Command]
(Function: shu-gfn)

While in a file buffer, put the name of the current file into the kill ring.

gpl [Command]
(Function: shu-git-insert-pull-origin-branch)

Insert at point the name the git command to pull the current branch from origin.

gps [Command]
(Function: shu-git-insert-push-origin-branch)

Insert at point the git command to push the current branch out to origin. If the current branch is the default branch (fka “master”), you are prompted to see if you want to proceed. This is to prevent an accidental push to the default branch.

gquote [Command]
(Function: shu-gquote)

Insert a LaTeX quote environment and position the cursor for typing the quote.

insb [Command]
(Function: shu-git-insert-branch)

Insert at point the name of the current branch in a git repository

inso [Command]

(Function: shu-git-insert-origin-branch)

Insert at point the name of the current branch in a git repository preceded by the word “origin”.. This can be used as part of git push or pull.

kill-system-name [Command]

(Function: shu-kill-system-name)

Place the system name‘ (machine name) in the message area.

loosen-lisp [Command]

(Function: shu-loosen-lisp)

Within the bounds of a lisp function, unwind the parentheses that terminate conditional and containing functions such that it is convenient to insert code inside of them without having to worry about which line contains the closing parenthesis. All closing parentheses are now on separate lines. Once the changes to the function are complete, you can run *shu-tighten-lisp* to put the parentheses back where they belong.

make-md-toc [Command]

(Function: shu-tocify-markdown-file)

Search the file starting at the current position for any markdown headings of the form “## This is a heading”. Add a tag to each heading and then insert a complete markdown table of contents at the current position.

Pound signs that lie inside of markdown literal areas designated by “” are ignored. This prevents something such as an example of an `#include` directive from being treated as a level 1 heading.

If a heading already has a tag, it is removed. If a heading has trailing pound signs, they are also removed.

The default maximum heading level is two, which means that heading levels greater than two are not included in the table of contents. But a numeric prefix argument can change the maximum heading level. The maximum heading level cannot be set to a value less than one.

md-name [Command]

(Function: shu-make-md-name-entry)

The latest item in the kill ring is assumed to be the text of a markdown section name. This function creates from that section name, a markdown table of contents name that will identify the section in the table of contents.

For example, if the kill ring contains “## This is the Overview”, the table

of contents name created and inserted at point will be:

```
<a name=thisistheoverview></a>
```

md-toc [Command]

(Function: shu-make-md-toc-entry)

The latest item in the kill ring is assumed to be the text of a markdown section name. This function creates from that section name, a markdown table of contents entry and inserts that entry at point.

For example, if the kill ring contains “## This is the Overview ##”, the table of contents entry created and inserted at point will be

```
* [This is the overview](#thisistheoverview)
```

modified-buffers [Command]

(Function: shu-modified-buffers)

Show a list of all buffers associated with files whose status is modified. It is not uncommon to have many emacs windows open and to realize that one window has a file open and another window is also trying to edit it. emacs warns the second window of the conflict, but it is sometimes difficult to tell which window holds the modified buffer. The buffer list shows you all of the buffers with an asterisk next to each modified buffer, but if the buffer list is large, it can be difficult to find the one you seek. This command lists only modified buffers that hold the contents of a file.

new-ert *func-name* [Command]

(Function: shu-new-ert)

Insert at point a skeleton lisp ert unit test. Prompt is issued for the function name.

new-latex [Command]

(Function: shu-new-latex)

Build a skeleton, empty LaTeX file.

new-lisp *func-name* [Command]

(Function: shu-new-lisp)

Insert at point a skeleton lisp function. Prompt is issued for the function name.

new-lisp-while *var-name* [Command]

(Function: shu-new-lisp-while)

Insert at point a skeleton lisp while loop. Prompt is issued for the variable name. The while loop is of the form:

```
(while x

  (setq x (cdr x))
)
```

point is placed where the the first line of code in the loop belongs.

number-commits [Command]

(Function: shu-git-number-commits)

In a git log buffer, number all of the commits with zero being the most recent.

It is possible to refer to commits by their SHA-1 hash. If you want to see the difference between two commits you can ask git to show you the difference by specifying the commit hash of each one. But this is cumbersome. It involves copying and pasting two SHA-1 hashes. Once the commits are numbered, then *shu-git-diff-commits* may be used to diff two commits by number. See the documentation for *shu-git-diff-commits* for further information.

This function counts as a commit any instance of “commit” that starts at the beginning of a line and is followed by some white space and a forty character hexadecimal number. Returns the count of the number of commits found.

number-lines [Command]

(Function: shu-number-lines)

Insert in front of each line in the buffer its line number. Starts at point and continues to the end of the buffer.

obfuscate-region *start end* [Command]

(Function: shu-obfuscate-region)

Obfuscate a region of text by replacing every alphabetic character in the region with the next letter of the alphabet, starting with 'a'. For example, if the region contains

Now is the time for all good men to come to the aid of the Party 10 times.

Then the obfuscated text would be:

Abc de fgh ijkl mno pqr stuv wxy za bcde fg hij klm no pqr Stuvw 10 xyzab.

This is useful if you want to capture some text for later testing and manipulation that might contain confidential or proprietary information. This is an encoding that cannot be reversed.

of [Command]

(Function: shu-of)

While in dired, open the current file (Mac OS X only)

os-name [Command]

(Function: shu-show-os-name)

Display the name of the host operating system type. This is a sanity check function for the various functions defined in .emacs to determine the type of the host operating system.

remove-test-names [Command]

(Function: shu-remove-test-names)

Remove from a file all lines that contain file names that end in .t.cpp

reverse-comma-names [Command]

(Function: shu-reverse-comma-names)

In a list of names, change all occurrences of Lastname, Firstname to Firstname Lastname. Position to the start of the file and invoke once.

scan-grok [Command]

(Function: shu-extract-name-open-grok)

The current buffer contains output from an OpenGrok search that has been copied from the web page and pasted into the buffer. This function scans the buffer from the current point and harvests all of the file names that hold the references for which OpenGrok searched. It puts the file names (including their top level directories) into the buffer “{**shu-open-grok**}”. The number of file names found is returned, mostly for the benefit of unit tests.

set-dos-eol [Command]

(Function: shu-set-dos-eol)

Set the end of line delimiter to be the DOS standard (CRLF).

set-unix-eol [Command]

(Function: shu-set-unix-eol)

Set the end of line delimiter to be the Unix standard (LF).

show-branch [Command]

(Function: `shu-git-show-branch`)

Display the name of the current branch in a git repository.

`show-repo` [Command]

(Function: `shu-show-repo`)

Call *shu-get-repo* to find the path to the repository and show the result in the minibuffer.

`show-system-name` [Command]

(Function: `shu-show-system-name`)

Place the system name (machine name) in the message area.

`tighten-lisp` [Command]

(Function: `shu-tighten-lisp`)

Within the bounds of a lisp function or macro, “tighten” some lisp code. Look for any single right parenthesis that is on its own line and move it up to the end of the previous line. This function is the opposite of *shu-loosen-lisp*

`trim-trailing-blanks` [Command]

(Function: `shu-trim-trailing-blanks`)

Eliminate whitespace at ends of all lines in the current buffer.

`unbrace` [Command]

(Function: `shu-unbrace`)

When point is on an opening sexp, this function converts, within the scope of the sexp, all “{” to “(“ and all “}” to “)”. If the number of left braces does not match the number of right braces a warning message is emitted.

For the benefit of unit tests, the count of left braces converted iff the count of left braces matches the count of right braces. If the counts do not match, nil is returned.

`winpath start end` [Command]

(Function: `shu-winpath`)

Take marked region, put in kill ring, changing / to \. This makes it a valid path on windows machines.

14.2 List of functions and variables

List of functions and variable definitions in this package.

`shu-adapt-frame frame-no` [Command]

(Alias: af)

Adapt the current frame to the current display by stretching the frame to the full height of the display and putting the top of the frame at the top of the display. This function makes some assumptions about the display geometry based on the current operating system. It assumes that Windows loses five lines for top and bottom tool bars. Mac OS X loses three lines for the top tool bar. Unix loses two lines for something. These numbers should, at some point, be customizable.

With a numeric prefix argument N, the emacs window is positioned N frames from the right hand side of the display. For example, if you open three frames and type into the first frame C-u 1 M-x *shu-adapt-frame*, into the next frame C-u 2 M-x *shu-adapt-frame*, and into the third frame C-u 3 M-x *shu-adapt-frame*, then the three frames will be grouped together side by side at the right side of the display.

If the prefix argument is large enough that the left side of the frame would be moved past the left side of the display, the window is positioned such that the left edge of the window is aligned with the left edge of the display.

Implementation note:

If this function is called when the left side of the frame is positioned to the left of the leftmost edge of the display, the function FRAME-POSITION returns a negative value for the x coordinate of the frame. The function SET-FRAME-POSITION takes the x and y coordinates of the new position of the top left corner of the frame.

But if x is negative, it specifies the coordinates of the right edge of the frame relative to the right edge of the display. This puts a frame that is very close to the left edge of the display all the way over to the right edge of the display.

The assumption is that a negative x frame position means that the user has positioned the frame just a bit past the left edge and that the desired frame position is actually the leftmost edge of the display.

shu-add-alexandria

[Command]

(Alias: add-alexandria)

Add Alexandria coverage to a git repository. This function first checks to ensure that a README.md file exists that does not contain an Alexandria badge and that a Doxyfile does not exist. If those two conditions are met, an Alexandria badge is added to the bottom of the README.md file, a Doxyfile is created, and some of the tags in the Doxyfile are set to reasonable defaults. An ALEXANDRIA_DOC_DEPENDENCIES tag is added to the end of the

Doxyfile as a comment. If this function succeeds, it returns true, else nil. The return value may be used by batch mode functions that want to call this function and report whether or not it succeeded.

shu-add-alexandria-badge [Command]

(Alias: add-alexandria-badge)

Insert an Alexandria badge for the current project.

shu-add-alexandria-in-batch-mode [Function]

Call the function *shu-add-alexandria* in batch mode. The function *shu-add-alexandria* normally ends in edit mode in Doxyfile so that the user can do a final edit and save. In batch mode, this function does a save of the Doxyfile since there is no interactive user.

If the function succeeds, it returns true, else nil. This allows the top level batch invoking function to terminate emacs with a zero or non-zero return code to indicate to an external script whether or not the add command worked.

shu-add-doxyfile [Function]

Call “doxygen -g” to create a Doxyfile. Return the output from the doxygen command.

shu-add-prefix *prefix* [Command]

(Alias: add-prefix)

Put a prefix and a space in front of each line in the region. Prompt is issued for the prefix.

shu-all-quit [Command]

(Alias: all-quit)

Kill all direcd buffers and all buffers that contain a file and are unmodified. It is not uncommon to have dozens of buffers open that are unrelated to the current task and this is a convenience function for closing many buffers that do not need to be open. If the function *shu-clear-c-project* is defined, it is called to clear the current project.

shu-buffer-number-lines [Command]

(Alias: buffer-number-lines)

Create a buffer whose name is derived from the file name of the current buffer but with the string “-numbered” added to the name. Thus “foo.cpp” would become “foo-numbered.cpp” Into this new buffer, copy the contents of the current file with each line prefixed with its line number. This is designed for

those times when you want to copy snippets of code with the line number in front of each line because you are commenting on code and want the person receiving the comments to see the line number in front of each line.

shu-case-insensitive [Command]

(Alias: case-insensitive)

Set the variable `case-fold-search` to `t` to make searches and matches ignore case. I can never remember which way to set `case-fold-search`, hence this simple, little function.

shu-case-sensitive [Command]

(Alias: case-sensitive)

Set the variable `case-fold-search` to `nil` to make searches and matches respect case. I can never remember which way to set `case-fold-search`, hence this simple, little function.

shu-comma-names-to-letter [Command]

(Alias: comma-names-to-letter)

In a list of names, change all occurrences of Lastname, Firstname to an empty Latex letter. Position to the start of the file and invoke once.

shu-conditional-find-file *file-name* [Command]

Make the buffer for *file-name* the current buffer. If *file-name* is already loaded into a buffer, then make that the current buffer. If *file-name* is not loaded into a buffer, load the file into a buffer and make that the current buffer. Return true if this function created the buffer, nil otherwise.

This function is intended to handle the situation in which a function wants to visit the contents of several files but does not want to leave behind a lot of file buffers that it created.

If this function returns true, then the calling function should kill the buffer when it is finished with it.

shu-copy-repo [Command]

(Alias: copy-repo)

Call *shu-get-repo* to find the path to the repository and put the result in the kill ring.

shu-de-star [Command]

(Alias: de-star)

Remove leading spaces and asterisk from each line in the region. This is useful

for editing doxygen comments of the form:

```
/*!  
  
* This is some commentary.  
* This is more commentary, etc.  
\emph{*/}
```

You snip out the middle lines and put them into a text file for formatting and spell-checking. You want to get rid of all of the asterisks until you are done.

This function gets rid of all the asterisks. You can use *shu-add-prefix* to put them back.

shu-dired-mode-name [Constant]

The name of the mode for a dired buffer

shu-disabled-quit [Command]

Explain that C-x C-c no longer kills emacs. Must M-x quit instead. Far too often, I hit C-x C-c by mistake and emacs vanishes. So I map C-x C-c to this function and use an explicit M-x quit to exit emacs.

shu-dup [Command]

(Alias: dup)

Insert a duplicate of the current line, following it.

shu-eob [Command]

Go to end of buffer without setting mark. Like end-of-buffer but does not set mark - just goes there.

shu-erase-region *start end* [Function]

Replace everything in the region between *start* and *end* with blanks. This is exactly like delete-region except that the deleted text is replaced with spaces. As with delete-region, the end point is not included in the delete. It erases everything up to but not including the end point. The order of *start* and *end* does not matter.

shu-extract-name-open-grok [Command]

(Alias: scan-grok)

The current buffer contains output from an OpenGrok search that has been copied from the web page and pasted into the buffer. This function scans the buffer from the current point and harvests all of the file names that hold the references for which OpenGrok searched. It puts the file names

(including their top level directories) into the buffer “`**shu-open-grok**`”. The number of file names found is returned, mostly for the benefit of unit tests.

shu-find-numbered-commit *commit-number* [Function]

Search through a numbered git commit log looking for the commit whose number is *commit-number*. Return the SHA-1 hash of the commit if the commit number is found. Return nil if no commit with the given number is found. The commit log is assume to have been numbered by shu-git-number-commits.

shu-fix-markdown-section *max-depth* [Function]

On entry, point is positioned after one or more pound signs that define the beginning of a markdown section heading. If the number of pound signs is greater than *max-depth*, ignore the line and return nil. If the number of pound signs is less than or equal to *max-depth*, fix the line as described below and return it.

If the line ends with an expression that looks like

```
‘‘<a name=currentliveupdate></a>‘‘,
```

remove it.

If the line ends with trailing pound signs, remove them as well.

Then return the repaired line.

shu-fix-times [Command]

Go through a buffer that contains timestamps of the form

```
YYYY-MM-DDTHHMMSS.DDD
```

converting them to the form

```
YYYY-MM-DD HH:MM:SS.DDD
```

The latter is a format that Microsoft Excel can import.

shu-fixup-doxyfile [Command]
(Alias: fixup-doxyfile)

The current directory is assumed to have the same name as the project for which the Doxyfile was created. This function sets various default values in the Doxyfile. The current buffer is the Doxyfile.

shu-fixup-project-doxyfile *project-name* [Command]

project-name is the name of the project for which the Doxyfile has been created. This function sets standard default values. If this function succeeds, it return true, else nil. The return value may be used in batch mode to determine if the fixup was successful.

shu-forward-line [Function]

Move forward by one line. If there is a next line, point it moved into it. If there are no more lines, a new one is created.

shu-frame-width [Function]

Return the width of an emacs frame. Different operating systems appear to have slightly different windowing systems, which means that the FRAME-INNER-WIDTH function does not quite report the exact width.

shu-gd [Command]

(Alias: gd)

While in dired, put the full path to the current directory in the kill ring

shu-get-containing-function [Function]

Search backwards from the current point to find the beginning of the enclosing function, macro, etc. If such a beginning is found, return a cons cell whose car is the point that defines the point at the beginning of the function and whose cdr defines the point at the end of the function. If not inside a function, macro, etc., return nil

shu-get-current-line [Function]

Return the current line in the buffer as a string

shu-get-debian-dependencies [Function]

This function tries to find a Debian library dependency file in the current directory tree. If such a file is found, this function returns a sorted list of the dependencies listed in the Debian dependency file. If no such file exists, nil is returned.

shu-get-debian-dependency-file [Function]

Use the name of the current directory as the name of a debian library. Construct a dependency file name which is the name of the current directory with

a file type of “.dep”. Search through the directory tree for such a file. If the file is found return its fully qualified name, i.e., the full path to the file so that it may be opened. If no such file exists, return nil.

shu-get-debian-dependency-line [Function]

This function tries to find a Debian library dependency file in the current directory tree. If such a file is found, this function returns a single line of text that holds the space separated names of all of the dependencies.

shu-get-git-name *path* [Function]

path is the url of a git repository from the [remote “origin”] section of a .git/config file. For example, the entry for this repository is

`https://github.com/codesinger/shu.git`

This function extracts two pieces of information from the URL. One is the name of the repository, which in this case is “shu”. The other is the path to the repository, which includes the owning group, which in this case is “codesinger/shu”.

Those two items are returned in a cons cell with the car of the cons cell holding the path (with owning group) and the cdr of the cons cell holding the repository name.

The assumptions made by this function are as follows: The beginning of the owning group and repository name are preceded by a domain name followed by either a colon or a slash. In the case of this repository, the owning group and repository name are preceded by “github.com/”. The repository name may or may not have a trailing “.git”, which this function removes.

shu-get-git-repo-name [Function]

This function tries to get the name of the current git repository from the .git/config file. Returns nil if it cannot open .git/config.

shu-get-git-repo-path [Function]

This function tries to get the host local path to the current git repository from the .git/config file if possible. If it cannot find the .git/config file, the it uses the shu custom variable *shu-internal-group-name* as the group owner and uses the name of the current directory as the repository name and constructs a host local path that is the owning group name, a slash, and the putative repository name (the name of the current directory).

shu-get-markdown-heading *section-heading* [Function]

Returns the heading text from a markdown section heading, *section-heading*. There must be at least one pound sign at the beginning of the string. If a section heading is

“### This is a section heading”

then the string “This is a section heading” is returned. If the first character in the section heading is not a pound sign, nil is returned.

shu-get-markdown-level *section-heading* [Function]

Return the level of a markdown section heading. The level is defined as the number of leading pound signs that start at the beginning of the string. A level 1 heading begins with “#”. A level 2 heading begins with “##”. If there are no leading pound signs at the beginning of the string, a level of zero is returned.

shu-get-markdown-prefix *section-heading* [Function]

Returns the pound sign prefix from a markdown section heading, *section-heading*. The string of pound signs must begin at the beginning of the string. If a section heading is

“### This is a section heading”

then the string “###” is returned. If the first character in the section heading is not a pound sign, nil is returned.

shu-get-md-boundaries [Function]

Find all pairs of markdown literal text. In markdown, the sequence “” is used to bound literal text. When creating a markdown table of contents, we do not want to look at pound signs contained in literal text. This function finds the location of each pair of “” sentinels. It returns a list of cons cells, each of which has the start and end position of a “” sequence. If there is a start “” with no companion “” close, it is not included in the list.

shu-get-repo [Function]
(Alias: get-repo)

When positioned anywhere in a git repository, return the git path to the repository. This is found in .git/config as the url of [remote “origin”]. Return nil if the path cannot be found.

The search is made from the current directory and upwards for the first directory called “.git”.

shu-get-url-repo [Function]

Return the web URL for the current git repository. If the URL cannot be found, nil is returned.

The url for the git repository in .git/config is of the form

```
git@web-address:repository-name.git
```

This function removes the trailing “.git”, replaces the leading “git@” with “https://” and replaces the “:” between the web-address and repository-name with “/”.

shu-gf [Command]
(Alias: gf)

While in dired, put the full path to the current file in the kill ring

shu-gfc [Command]
(Alias: gfc)

While in a file buffer, put both the current line number and column number and the name of the current file into the kill ring in the form of “foo.cpp:123:2”.

shu-gfl [Command]
(Alias: gfl)

While in a file buffer, put both the current line number and the name of the current file into the kill ring in the form of “line 1234 of foo.cpp”.

shu-gfn [Command]
(Alias: gfn)

While in a file buffer, put the name of the current file into the kill ring.

shu-git-add-file *filename* [Function]
Do a “git add” for *filename*. Return empty string if add succeeds. Otherwise, return git error message.

shu-git-branch-to-kill-ring [Command]
Put the name of the current branch in a git repository into the kill ring.

shu-git-diff-commits *commit-range* [Command]
(Alias: diff-commits)

In a buffer that is a numbered git log, query for a range string, find the two commits, and put into the kill ring a git diff command specifying the two commits.

For example, given the following two numbered commits:

```
31. commit 38f25b6769385dbc3526f32a75b97218cb4a6754
33. commit 052ee7f4297206f08d44466934f1a52678da6ec9
```

if the commit range specified is either “31.33” or “31+2”, then the following is put into the kill ring:

```
‘‘git diff -b 38f25b6769385dbc3526f32a75b97218cb4a6754..052ee7f4297206f08d44466934f1a52678da6ec9
```

shu-git-find-branch [Function]

Return the name of the current branch in a git repository.

shu-git-find-default-branch [Function]

Return the name of the default branch in a git repository. The default branch is the one branch that is created with a new repository.

shu-git-find-short-hash *hash* [Function]

Return the git short hash for the *hash* supplied as an argument. Return nil if the given *hash* is not a valid git revision.

shu-git-get-pr-url [Command]

(Alias: get-pr-url)

Put into the kill ring the path required to create a new pull request for the current branch of the current repository.

shu-git-insert-branch [Command]

(Alias: insb)

Insert at point the name of the current branch in a git repository

shu-git-insert-git-commit [Command]

(Alias: gcm)

Insert at point the name the git command to commit with the commentary held in a file called “why.txt”.

shu-git-insert-origin-branch [Command]

(Alias: inso)

Insert at point the name of the current branch in a git repository preceded by the word “origin”.. This can be used as part of git push or pull.

shu-git-insert-pull-origin-branch [Command]

(Alias: gpl)

Insert at point the name the git command to pull the current branch from origin.

shu-git-insert-push-origin-branch [Command]

(Alias: gps)

Insert at point the git command to push the current branch out to origin. If the current branch is the default branch (fka “master”), you are prompted to see if you want to proceed. This is to prevent an accidental push to the default branch.

shu-git-number-commits [Command]

(Alias: number-commits)

In a git log buffer, number all of the commits with zero being the most recent.

It is possible to refer to commits by their SHA-1 hash. If you want to see the difference between two commits you can ask git to show you the difference by specifying the commit hash of each one. But this is cumbersome. It involves copying and pasting two SHA-1 hashes. Once the commits are numbered, then *shu-git-diff-commits* may be used to diff two commits by number. See the documentation for *shu-git-diff-commits* for further information.

This function counts as a commit any instance of “commit” that starts at the beginning of a line and is followed by some white space and a forty character hexadecimal number. Returns the count of the number of commits found.

shu-git-show-branch [Command]

(Alias: show-branch)

Display the name of the current branch in a git repository.

shu-gquote [Command]

(Alias: gquote)

Insert a LaTeX quote environment and position the cursor for typing the quote.

shu-insert-markdown-toc *entries* [Function]

entries is a list of cons cells. The car of each item on the list is the markdown heading line, which looks something like “## This is a heading”. The cdr of each item on the list is the link name. This function inserts a markdown table of contents in which each line in the table of contents consists of the heading text in brackets followed by the line name in parenthesis and preceded by a

pound sign. Each line that represents a heading level greater than one is also indented to indicate its heading level.

shu-internal-get-repo [Function]

The current buffer holds an instance of the “.git/config” file for the repository. This function returns the git path to the repository, which is the url given after [remote “origin”]. nil is returned if the path cannot be found.

shu-internal-new-lisp *func-type func-name* [Command]
doc-string interactive

Insert at point a skeleton lisp function of type *func-type* whose name is *func-name*. *func-type* is not examined in any way but is only useful if its value is “defun”, “defmacro”, “ert-deftest”, etc. If *interactive* is true, the function is interactive.

shu-kill-current-buffer [Command]

Kills the current buffer.

shu-kill-repo [Command]

When positioned in the top level directory of a git repository, place into the kill ring the git path to the repository. This is found in .git/config as the url of [remote “origin”]

This should probably be extended to do a search for the .git directory anywhere above the current position, which would remove the requirement to be in the root of the repository.

shu-kill-system-name [Command]

(Alias: kill-system-name)

Place the system name‘ (machine name) in the message area.

shu-local-replace *from-string to-string* [Function]

Replaces *from-string* with *to-string* anywhere found in the buffer. This is like replace-string except that it is intended to be called by lisp programs. Note that this function does not alter the value of case-fold-search. The user should set it before calling this function.

shu-loosen-lisp [Command]

(Alias: loosen-lisp)

Within the bounds of a lisp function, unwind the parentheses that terminate conditional and containing functions such that it is convenient to insert code inside of them without having to worry about which line contains the closing

parenthesis. All closing parentheses are now on separate lines. Once the changes to the function are complete, you can run *shu-tighten-lisp* to put the parentheses back where they belong.

`shu-make-md-index-name` *name* [Function]

The input is a string that is assumed to be a markdown section heading from a markdown table of contents. The return value is an all lower case string with any whitespace characters removed. For example, if the input string is

```
This is an Overview
```

The returned string would be

```
thisisanoverview
```

`shu-make-md-name-entry` [Command]

(Alias: md-name)

The latest item in the kill ring is assumed to be the text of a markdown section name. This function creates from that section name, a markdown table of contents name that will identify the section in the table of contents.

For example, if the kill ring contains “## This is the Overview”, the table of contents name created and inserted at point will be:

```
<a name=thisistheoverview></a>
```

`shu-make-md-section-name` *section-name* [Function]

The input is a string that is assumed to be a markdown section heading. The return value is a string with any leading and trailing “#” characters removed. For example, if the input string is

```
## This is an Overview ##
```

The returned string would be

```
This is an Overview
```

shu-make-md-toc-entry [Command]
(Alias: md-toc)

The latest item in the kill ring is assumed to be the text of a markdown section name. This function creates from that section name, a markdown table of contents entry and inserts that entry at point.

For example, if the kill ring contains “## This is the Overview ##”, the table of contents entry created and inserted at point will be

```
* [This is the overview](#thisistheoverview)
```

shu-md-in-literal *literals pt* [Function]
literals is a list of markdown literal boundaries produced by *shu-get-md-boundaries*. This function returns t if the point *pt* lies within a markdown literal boundary.

shu-misc-get-chunk *line-limit &optional escape* [Function]
Return a string that consists of the first *line-limit* characters in the current buffer. If *line-limit* is larger than the buffer size, return a string that is the entire contents of the buffer. Before returning, delete from the buffer the returned string.

shu-misc-get-phrase *line-limit &optional escape* [Function]
Remove from the front of the current buffer and return the longest possible string of whitespace separated things whose length does not exceed *line-limit*. If there is at least one whitespace character before *line-limit*, the string will end with one or more whitespace characters. i.e., the string will end on a word boundary if that is possible.

Words will not be split unless there is no whitespace character before *line-limit* characters have been scanned, in which case a string of exactly *line-limit* length will be removed and returned.

This function is used to split a string of words into a set of smaller strings such that words are not split.

shu-misc-internal-split-buffer *line-limit* [Function]
&optional escape

Split an entire buffer into multiple strings and return a list of the strings. *get-function* is the function to call to fetch each new string. *get-function* is set to either *shu-misc-get-chunk* or *shu-misc-get-phrase*.

shu-misc-get-chunk returns each string as a fixed length string of *line-limit* characters, except for the last one, which may be shorter.

shu-misc-get-phrase returns the longest possible string that ends on a word boundary and whose length is less than or equal to *line-limit*.

shu-misc-make-unique-string *string suffix-length* [Function]

Input is a hash table, *ht*, as well as a *string*. If the string does not already exist in *ht*, add the string to the hash table and return the string. If the string already exists in *ht*, add a suffix to the string that is a random string of length *suffix-length*. If the combination of the original *string* plus the random string added as a suffix, does not exist in the hash table, add the new string to the hash table and return it. This provides the generation of a set of unique string names.

shu-misc-random-internal-string *letters length* [Function]

Return a string composed of random *letters* of length *length*.

shu-misc-random-lad-string *length* [Function]

Return a string composed of random lower case letters and digits of length *length*.

shu-misc-random-ua-string *length* [Function]

Return a string composed of random upper case letters of length *length*.

shu-misc-rx-conditionals [Constant]

Regular expression to find the beginning of a function or macro that encloses a body. Such functions usually require a future closing parenthesis that is likely not on the current line. This is used by the functions *shu-tighten-lisp* and *shu-loosen-lisp*.

shu-misc-rx-functions [Constant]

Regular expression to find the beginning of a function, macro, etc.

shu-misc-rx-lets [Constant]

Regular expression to find the beginning of a let special form. This searches for “let” or “let*” followed by “(“.

shu-misc-set-alias [Function]

Set the common alias names for the functions in shu-misc. These are generally the same as the function names with the leading shu- prefix removed.

shu-misc-split-buffer *line-limit* **&optional** *escape* [Function]

Split an entire buffer into multiple strings and return a list of the strings. If *fixed-width* is true, then each returned string is *line-limit* characters in length, except for the last, which may be shorter. If *fixed-width* is absent or nil, then each returned string is split on a word boundary and no string exceeds *line-limit* characters in length.

shu-misc-split-chunk-buffer *line-limit* **&optional** [Function]

Split an entire buffer into multiple strings and return a list of the strings. Each returned string is *line-limit* characters in length, except for the last one, which may be shorter.

shu-misc-split-phrase-buffer *line-limit* [Function]

Split an entire buffer into multiple strings and return a list of the strings. Each returned string is split on a word boundary and no string exceeds *line-limit* characters in length.

shu-misc-split-string *input line-limit* **&optional** *escape* [Function]

Split a string into multiple strings and return a list of the strings. If *fixed-width* is true, then each returned string is *line-limit* characters in length, except for the last, which may be shorter. If *fixed-width* is absent or nil, then each returned string is split on a word boundary and no string exceeds *line-limit* characters in length.

shu-modified-buffers [Command]
(Alias: modified-buffers)

Show a list of all buffers associated with files whose status is modified. It is not uncommon to have many emacs windows open and to realize that one window has a file open and another window is also trying to edit it. emacs warns the second window of the conflict, but it is sometimes difficult to tell which window holds the modified buffer. The buffer list shows you all of the buffers with an asterisk next to each modified buffer, but if the buffer list is large, it can be difficult to find the one you seek. This command lists only modified buffers that hold the contents of a file.

shu-move-down *arg* [Command]

Move point vertically down. Whitespace in any direction is made if necessary. New lines will be added at the end of a file and lines that are too short will

be expanded as necessary.

shu-new-ert *func-name* [Command]
(Alias: new-ert)

Insert at point a skeleton lisp ert unit test. Prompt is issued for the function name.

shu-new-latex [Command]
(Alias: new-latex)

Build a skeleton, empty LaTeX file.

shu-new-lisp *func-name* [Command]
(Alias: new-lisp)

Insert at point a skeleton lisp function. Prompt is issued for the function name.

shu-new-lisp-while *var-name* [Command]
(Alias: new-lisp-while)

Insert at point a skeleton lisp while loop. Prompt is issued for the variable name. The while loop is of the form:

```
(while x  
  
  (setq x (cdr x))  
)
```

point is placed where the the first line of code in the loop belongs.

shu-next-char-in-seq *current-char* [Function]
current-char is a character in the range a-z (or A-Z). This function returns the next character, where next is the next character in the alphabet unless *current-char* is 'z', in which case the next character returned is 'a'. If *current-char* is 'Z', then the next character returned is 'A'.

shu-number-lines [Command]
(Alias: number-lines)

Insert in front of each line in the buffer its line number. Starts at point and continues to the end of the buffer.

shu-obfuscate-region *start end* [Command]
(Alias: obfuscate-region)

Obfuscate a region of text by replacing every alphabetic character in the region with the next letter of the alphabet, starting with 'a'. For example, if the region contains

Now is the time for all good men to come to the aid of the Party 10 times.

Then the obfuscated text would be:

Abc de fgh ijkl mno pqr stuv wxy za bcde fg hij klm no pqr Stuvw 10 xyzab.

This is useful if you want to capture some text for later testing and manipulation that might contain confidential or proprietary information. This is an encoding that cannot be reversed.

shu-of [Command]
(Alias: of)

While in dired, open the current file (Mac OS X only)

shu-os-name [Function]

Return a string with the name of the type of the host operating system.

shu-put-line-near-top [Command]

Take the line containing point and position it approximately five lines from the top of the current window.

shu-quit [Command]

Invoke save-buffers-kill-emacs. This is the function normally invoked by C-x C-c

shu-remove-test-names [Command]

(Alias: remove-test-names)

Remove from a file all lines that contain file names that end in .t.cpp

shu-reverse-comma-names [Command]

(Alias: reverse-comma-names)

In a list of names, change all occurrences of Lastname, Firstname to Firstname Lastname. Position to the start of the file and invoke once.

shu-reverse2 [Command]

When positioned in front of a pair of parenthesis that contains a pair of expressions separated by a comma, reverse the positions of the two expressions. The first becomes the second and the second becomes the first. i.e.,

```
foo(mumble, bar);
```

becomes

```
foo(bar, mumble);
```

shu-save-and-load [Command]
(Alias: eld)

Save and load the current file as a .el file.

shu-set-buffer-eol-type *eol-type* [Function]
Define what the end of line delimiter is in a text file.

shu-set-dos-eol [Command]
(Alias: set-dos-eol)

Set the end of line delimiter to be the DOS standard (CRLF).

shu-set-mac-eol [Command]
Set the end of line delimiter to be the Mac standard (CR).

shu-set-unix-eol [Command]
(Alias: set-unix-eol)
Set the end of line delimiter to be the Unix standard (LF).

shu-shift-line *count* [Command]
Shift a line of text left or right by *count* positions. Shift right if *count* is positive, left if *count* is negative. Shifting left only eliminates whitespace. If there is a non-whitespace character in column 5, then shift by -10 will only shift left 4.

shu-shift-region-of-text *count start end* [Command]
Shift a region of text left or right. The text to be shifted is defined by the bounds of lines containing point and mark. The shift count is read from the minibuffer.

shu-shift-single-line *count* [Function]
Shift a line of text left or right by *count* positions. Shift right if *count* is positive, left if *count* is negative. Shifting left only eliminates whitespace. If there is a non-whitespace character in column 5, then shift by -10 will only shift left 4.

shu-show-os-name [Command]
(Alias: os-name)

Display the name of the host operating system type. This is a sanity check function for the various functions defined in .emacs to determine the type of the host operating system.

shu-show-repo [Command]

(Alias: show-repo)

Call *shu-get-repo* to find the path to the repository and show the result in the minibuffer.

shu-show-system-name [Command]

(Alias: show-system-name)

Place the system name (machine name) in the message area.

shu-split-range-string *range-string* [Function]

range-string is a string that contains either one or two numbers, possibly separated by plus, minus, or period. If one number then it is the starting number and there is no ending number. If two numbers then the first number is the start. The operator in the middle determines the end. If plus, then the end is the second number added to the first. If minus, then the end is the second number subtracted from the first. If period, then the end is the second number.

Return the two numbers as a cons cell (start . end). If there is no end then the cdr of the cons cell is nil. If range string is not numeric, then both the car and the cdr of the cons cell are nil.

For example, “99+2” has start 99 and end 101. “99-2” has start 99 and end 97. “99.103” has start 99, end 103. “98” has start 98 and end is nil.

shu-system-name [Function]

Return the machine name. Prior to emacs 25.1, this was held in the variable system-name. As of emacs 25.1, system-name is now a function. Return nil if system-name is neither a function nor a variable.

shu-system-name-string [Function]

Return the machine name. Prior to emacs 25.1, this was held in the variable system-name. As of emacs 25.1, system-name is now a function. Unlike *shu-system-name*, this function always returns a string, even if the machine name is not available for some reason.

shu-tighten-hanging-paren *eof* [Command]

Call this function while point is on a left parenthesis. This function will find

the matching right parenthesis. If the matching right parenthesis is on a line by itself and a previous line ends in another right parenthesis, the line and dangling right parenthesis will be moved up to the end of the line that also ends in a right parenthesis. This is an internal part of the function *shu-tighten-lisp*. *eof* is the point at which the current function on which we are operating ends. This function removes some text from the current function. It adjusts *eof* appropriately and returns the new value to the caller.

shu-tighten-lisp [Command]
(Alias: tighten-lisp)

Within the bounds of a lisp function or macro, “tighten” some lisp code. Look for any single right parenthesis that is on its own line and move it up to the end of the previous line. This function is the opposite of *shu-loosen-lisp*

shu-tocify-markdown-file [Command]
(Alias: make-md-toc)

Search the file starting at the current position for any markdown headings of the form “## This is a heading”. Add a tag to each heading and then insert a complete markdown table of contents at the current position.

Pound signs that lie inside of markdown literal areas designated by “” are ignored. This prevents something such as an example of an *#include* directive from being treated as a level 1 heading.

If a heading already has a tag, it is removed. If a heading has trailing pound signs, they are also removed.

The default maximum heading level is two, which means that heading levels greater than two are not included in the table of contents. But a numeric prefix argument can change the maximum heading level. The maximum heading level cannot be set to a value less than one.

shu-tocify-markdown-headings *entries* [Function]

entries is a list of cons cells. The car of each item on the list is the markdown heading line, which looks something like “## This is a heading”. The cdr of each item on the list is the link name. This function searches for each markdown heading in the file and appends to the heading a tag of the form

```
<a name=link name></a>
```

so that it may be referenced from the table of contents.

shu-trim-git-end *path* [Command]

First trim leading and trailing spaces from *path*. If *path* ends in “.git”, trim the last four characters from the path. If *path* does not end in “.git”, do not trim the last four characters.

Return the *path*, leading and trailing spaces trimmed, with perhaps “.git” removed from the end.

shu-trim-trailing-blanks [Command]

(Alias: trim-trailing-blanks)

Eliminate whitespace at ends of all lines in the current buffer.

shu-unbrace [Command]

(Alias: unbrace)

When point is on an opening sexp, this function converts, within the scope of the sexp, all “{” to “(“ and all “}” to “).”. If the number of left braces does not match the number of right braces a warning message is emitted.

For the benefit of unit tests, the count of left braces converted iff the count of left braces matches the count of right braces. If the counts do not match, nil is returned.

shu-winpath *start end* [Command]

(Alias: winpath)

Take marked region, put in kill ring, changing / to \ This makes it a valid path on windows machines.

15 shu-nvplist

elisp code for maintaining directories of name / value pairs.

15.1 List of functions and variables

List of functions and variable definitions in this package.

shu-get-item-nvplist *item* [Function]
Return the name value pair list from an item.

shu-nvplist-get-item-number *item* [Function]
Return the item number for an item.

shu-nvplist-get-item-value *name item* [Function]
Extract a named list of values from an item. *name* is the name of the values to find. *item* is the item from which to extract the values. A list is returned that contain all of the values whose name matches *name*.

shu-nvplist-make-item *item-number nvplist* [Function]
Create an item entry from an item number and a name value pair list. The item entry is just a cons cell with the item number in the CAR and the name-value pair list in the CDR.

shu-nvplist-make-nvpair-list *tlist* [Function]
Turn a list of tokens from an entry in the file into a list of name value pairs. The CAR of each entry in the list is the name. The CDR of each entry in the list is the value. If errors are found in the token list, then an empty list is returned.

shu-nvplist-make-token-list *tlist* [Function]
Turn an entry in a name / value file into a list of tokens. The CAR of each entry is the point at which the token starts. the CDR of each entry in the list is the token itself. On entry to this function, point is immediately after the start delimiter (“<”). On return, point is positioned immediately after the end delimiter (“/>”).

shu-nvplist-parse-buffer *item-list* [Function]
Parse an nvplist buffer, putting all of the items in the *item-list*.

shu-nvplist-parse-file *file-name file-type* [Function]

Parse a file full of name value pair lists. The name of the file is *file-name*. The type of the file (only for error messages) is *file-type*. *item-list* is the head of the returned item list.

`shu-nvplist-show-item` *item* [Function]
Undocumented

`shu-nvplist-show-item-list` *item-list* [Function]
Undocumented

16 shu-org-extensions

The major function of this file is the function *shu-org-archive-done-tasks*, which can be used as an after-save-hook for org files. It finds each TODO item that was marked DONE more than SHU-ORG-ARCHIVE-EXPIRY days ago and moves it to an archive file by invoking org-archive-subtree on it.

16.1 List of functions by alias name

A list of aliases and associated function names.

gorg [Command]
(Function: shu-goto-home-org-file)
Visit the org home file.

16.2 List of functions and variables

List of functions and variable definitions in this package.

shu-goto-home-org-file [Command]
(Alias: gorg)
Visit the org home file.

shu-org-archive-done-tasks [Command]
Go through an org file and archive any completed TODO item that was completed more than shu-org-archive-expiry-days days ago.

shu-org-archive-expiry-days [Variable]
Number of elapsed days before a closed TODO item is automatically archived.

shu-org-date-match-regexp [Function]
Return a regexp string that matches an org date of the form 2014-04-01 Tue 13:18.

shu-org-done-keywords [Variable]
Key words that represent the DONE state.

shu-org-done-projects-string [Function]
Return a string that is a search for a TODO tag that does not contain any of the words that represent a DONE item. These are the words defined in org-done-keywords-for-agenda. If the two keywords that mean finished

item are DONE and CANCELLED, then this function will return the string: TODO={.+}/-CANCELLED-DONE. This is intended to be used in the definition of the variable “org-stuck-projects”.

shu-org-extensions-set-alias [Function]

Set the common alias names for the functions in shu-misc. These are generally the same as the function names with the leading shu- prefix removed.

shu-org-home [Variable]

Home directory of the org data files.

shu-org-state-regexp *done-word* [Function]

Return a regular expression that will match a particular TODO state record of the form - State “DONE” from “CANCELLED” [2012-04-01 Tue 13:18] *done-word* is the desired state of the record.

shu-org-todo-keywords [Variable]

Key words that represent the not DONE state.

17 shu-xref

A set of functions that scan a set of elisp files and create a cross reference of all of the definitions (functions, macros, constants, variables, etc.). See the doc string for *shu-make-xref* for further details.

17.1 List of functions and variables

List of functions and variable definitions in this package.

shu-get-all-definitions *fun-defs* [Function]
Find all of the emacs lisp function definitions in the current buffer.

shu-make-xref *start end* [Command]
Mark a region in a file that contains one name per line of an emacs lisp file. Then invoke shu-make-xref. It will do a cross reference of all of those files.

shu-xref-buffer [Constant]
The name of the buffer into which the cross reference is placed.

shu-xref-dump *fun-defs max-var-name-length* [Function]
Undocumented

shu-xref-file-compare *t1 t2* [Function]
Compare the file names from two variable names. Return t if the file name in *t1* comes before the type name in *t2*. If the file names are the same, then compare the variable names so that variables are in alphabetical order within file.

shu-xref-get-defs *file-list fun-defs* [Function]
Extract the variable definitions from each file.

shu-xref-get-file-list *start end file-list* [Command]
Return a list of file names from a region of a buffer. *start* and *end* define the region. Each line in the region is assumed to be a file name. *file-list* is the list that is also the return value of this function.

shu-xref-get-longest-name *fun-defs* [Function]
Return the length of the longest variable name in the list and the longest type name in the list. These are returned as a cons cell with the length of the longest type name in the CAR and the longest variable name in the CDR.

shu-xref-get-next-definition *retval* [Function]

Find and return the next definition of an emacs lisp function of variable. *retval* is returned as nil if there are no more function definitions after point. If a definition is found, *retval* is returned as a cons cell with the name of the function in the CAR and the information about the function in the CDR. The information in the CDR is a cons cell with the numeric variable type in the CAR and the line number in which the definition started in the CDR.

shu-xref-get-next-funcall *name retval* [Function]

Find and return the next call to the emacs lisp function *name*. *retval* is returned as nil if there are no more function invocations after point. If a function invocation is found, *retval* is returned as a cons cell with the name of the function in the CAR and the line number in which the function definition starts in the CDR.

shu-xref-lisp-name [Constant]

A regular expression to match a variable name in emacs lisp.

shu-xref-type-compare *t1 t2* [Function]

Compare the type names from two variable names. Return t if the type name in *t1* comes before the type name in *t2*. If the type names are the same, then compare the variable names so that variables are in alphabetical order within type.

shu-xref-var-types [Constant]

Associate a number with each type of variable

Index

author, 38

bde-add-guard, 18
bde-all-guard, 18
bde-decl, 18
bde-gen, 18
bde-include, 18
bde-sdecl, 18
bde-sgen, 19
binclude, 38

cdo, 39
celse, 39
cfor, 39
cif, 39
ck, 39
clc, 39
cpp1-class, 39
cpp2-class, 39
creplace, 40
csplit, 40
cunsplit, 41
cwhile, 41

dbx-malloc, 41
dcc, 42
dce, 42
dciterate, 42
diterate, 42
dox-brief, 43
dox-cbt, 43
dox-cvt, 43
dox2-hdr, 43
drc, 43

fill-area, 44
fill-data, 44
fixp, 44

gen-bb-component, 17
gen-bde-component, 19

shu-add-to-alist, 2
shu-add-to-alist-list, 2
shu-all-commits, 3
shu-all-whitespace-chars, 3
shu-all-whitespace-regexp, 3
shu-all-whitespace-regexp-scf, 3
shu-batch-add-alexandria, 15
shu-batch-copy-trace, 15
shu-batch-fail, 15
shu-batch-hello, 15
shu-batch-init, 15
shu-batch-rmv-using, 16
shu-batch-test-args, 16
shu-bb-cpp-set-alias, 17
shu-bde-add-guard, 19
shu-bde-all-guard, 19
shu-bde-decl, 19
shu-bde-gen, 19
shu-bde-gen-cfile-copyright-hook, 19
shu-bde-gen-file-identifier-hook, 19
shu-bde-gen-h-includes-hook, 20
shu-bde-gen-hfile-copyright-hook, 20
shu-bde-gen-tfile-copyright-hook, 20
shu-bde-include, 20
shu-bde-include-guard, 20
shu-bde-include-guard-fn, 20
shu-bde-insert-guard, 20
shu-bde-sdecl, 21
shu-bde-set-alias, 21
shu-bde-sgen, 21
shu-bool-to-string, 3
shu-capture-a-type-after, 23
shu-capture-a-type-arg, 23
shu-capture-a-type-before, 23

shu-capture-a-type-buf, 23	shu-capture-finish-doc-string-md, 29
shu-capture-a-type-close-quote, 23	shu-capture-func-type-name, 29
shu-capture-a-type-doc-string, 23	shu-capture-get-args-as-alist, 29
shu-capture-a-type-enclose-doc, 23	shu-capture-get-doc-string, 30
shu-capture-a-type-func, 24	shu-capture-get-func-def, 30
shu-capture-a-type-hdr, 24	shu-capture-get-func-def-alias, 30
shu-capture-a-type-keywd, 24	shu-capture-get-func-def-sig, 30
shu-capture-a-type-open-quote, 24	shu-capture-get-name-and-args, 30
shu-capture-alias-list, 24	shu-capture-headers-in-doc, 31
shu-capture-aliases, 24	shu-capture-index-buffer, 31
shu-capture-all-latex, 24	shu-capture-internal-all, 31
shu-capture-all-md, 24	shu-capture-internal-convert-doc-string, 31
shu-capture-arg-to-latex, 24	shu-capture-internal-doc, 32
shu-capture-arg-to-md, 24	shu-capture-keywd-optional, 32
shu-capture-attr-alias, 24	shu-capture-keywd-rest, 32
shu-capture-attr-const, 25	shu-capture-keywd-to-latex, 32
shu-capture-attr-custom, 25	shu-capture-keywd-to-md, 32
shu-capture-attr-inter, 25	shu-capture-latex, 32
shu-capture-attr-macro, 25	shu-capture-latex-arg-end, 32
shu-capture-attr-var, 25	shu-capture-latex-arg-start, 32
shu-capture-buf-to-latex, 25	shu-capture-latex-buf-end, 32
shu-capture-buf-to-md, 25	shu-capture-latex-buf-start, 32
shu-capture-buffer-name, 25	shu-capture-latex-close-quote, 33
shu-capture-code-in-doc, 25	shu-capture-latex-code-end, 33
shu-capture-code-in-md, 26	shu-capture-latex-code-start, 33
shu-capture-commentary, 26	shu-capture-latex-converters, 33
shu-capture-convert-args-to-markup, 26	shu-capture-latex-doc-end, 33
shu-capture-convert-doc-string, 27	shu-capture-latex-doc-start, 33
shu-capture-convert-func-latex, 27	shu-capture-latex-keywd-end, 33
shu-capture-convert-func-md, 28	shu-capture-latex-keywd-start, 33
shu-capture-convert-quotes, 28	shu-capture-latex-open-quote, 33
shu-capture-doc, 28	shu-capture-latex-section-end, 33
shu-capture-doc-code-indent, 28	shu-capture-latex-section-start, 33
shu-capture-doc-convert-args, 28	shu-capture-make-args-latex, 33
shu-capture-doc-convert-args-to-latex, 29	shu-capture-make-args-md, 34
shu-capture-doc-convert-args-to-md, 29	shu-capture-make-latex-section, 34
shu-capture-enclose-doc-latex, 29	shu-capture-make-md-section, 34
shu-capture-enclose-doc-md, 29	shu-capture-md, 34
shu-capture-file-list, 29	shu-capture-md-arg-delimiter, 35
shu-capture-finish-doc-string-latex, 29	shu-capture-md-buf-delimiter, 35

shu-capture-md-code-delimiter, 35
 shu-capture-md-converters, 35
 shu-capture-md-keywd-delimiter, 35
 shu-capture-md-quote-delimiter, 35
 shu-capture-md-section-delimiter, 35
 shu-capture-pre-code-in-doc, 35
 shu-capture-pre-code-latex, 35
 shu-capture-pre-code-md, 36
 shu-capture-set-func-def, 36
 shu-capture-set-func-def-alias, 36
 shu-capture-show-list, 36
 shu-capture-show-list-md, 36
 shu-capture-toc-buffer, 36
 shu-capture-vars, 37
 shu-comment-start-pattern, 3
 shu-conditional-load-library-files, 3
 shu-cpp-author, 4
 shu-cpp-comment-end, 4
 shu-cpp-comment-start, 4
 shu-cpp-default-allocator-name, 4
 shu-cpp-default-allocator-type, 4
 shu-cpp-default-global-namespace, 4
 shu-cpp-default-member, 4
 shu-cpp-default-namespace, 4
 shu-cpp-file-directory-name, 4
 shu-cpp-file-name, 5
 shu-cpp-file-name-list, 5
 shu-cpp-include-user-brackets, 5
 shu-cpp-indent-length, 5
 shu-cpp-line-end, 5
 shu-cpp-modern, 5
 shu-cpp-name, 5
 shu-cpp-name-list, 5
 shu-cpp-std-namespace, 6
 shu-cpp-use-bde-library, 6
 shu-current-line, 6
 shu-current-year, 6
 shu-date, 6
 shu-delete-last-char-if, 6
 shu-doc-internal-func-to-md, 37
 shu-doc-internal-to-md, 37
 shu-doc-sort-compares, 37
 shu-downcase-first-letter, 6
 shu-end-of-dq-string, 6
 shu-end-of-string, 6
 shu-fixed-format-num, 7
 shu-format-num, 7
 shu-gen-bb-component, 17
 shu-gen-bde-component, 21
 shu-gen-bde-create-prompt, 21
 shu-generate-bb-cfile, 17
 shu-generate-bb-hfile, 17
 shu-generate-bb-tfile, 17
 shu-generate-bde-cfile, 21
 shu-generate-bde-hfile, 21
 shu-generate-bde-tfile, 21
 shu-generate-comdb2-code, 16
 shu-generate-component, 16
 shu-generate-git-add, 21
 shu-get-directory-prefix, 7
 shu-global-buffer-name, 7
 shu-goto-line, 7
 shu-group-number, 7
 shu-internal-dev-url, 7
 shu-internal-gen-bde-component, 22
 shu-internal-group-name, 7
 shu-invert-alist-list, 8
 shu-invert-alist-to-hash, 8
 shu-kill-new, 10
 shu-last-commit, 10
 shu-library-files, 10
 shu-line-and-column-at, 10
 shu-line-number-at-pos, 10
 shu-load-library-files, 10
 shu-local-class-list, 16
 shu-minimum-leading-space, 10
 shu-non-cpp-name, 11
 shu-not-all-whitespace-regexp, 11
 shu-old-generate-component, 16
 shu-point-at-sexp, 11

- shu-point-in-string, 11
- shu-random-letter, 11
- shu-random-range, 11
- shu-remove-trailing-all-whitespace, 12
- shu-replace-string, 12
- shu-split-new-lines, 12
- shu-starts-with, 12
- shu-string-starts-ends, 12
- shu-swap, 13
- shu-the-column-at, 13
- shu-the-line-at, 13
- shu-trace-buffer, 13
- shu-trim, 13
- shu-trim-file, 13
- shu-trim-file-hook, 13
- shu-trim-leading, 13
- shu-trim-trailing, 13
- shu-unit-test-buffer, 14
- shu-upcase-first-letter, 14
- shu-version, 14