Speedbar

Eric Ludlam

Speedbar is a program for Emacs which can be used to summarize information related to the current buffer. Its original inspiration is the 'explorer' often used in modern development environments, office packages, and web browsers.

Speedbar displays a narrow frame in which a tree view is shown. This tree view defaults to containing a list of files and directories. Files can be 'expanded' to list tags inside. Directories can be expanded to list the files within itself. Each file or tag can be jumped to immediately.

Speedbar expands upon 'explorer' windows by maintaining context with the user. For example, when using the file view, the current buffer's file is highlighted. Speedbar also mimics the explorer windows by providing multiple display modes. These modes come in two flavors. Major display modes remain consistent across buffers, and minor display modes appear only when a buffer of the applicable type is shown. This allows authors of other packages to provide speedbar summaries customized to the needs of that mode.

Throughout this manual, activities are defined as 'clicking on', or 'expanding' items. Clicking means using using *Mouse-2* on a button. Expanding refers to clicking on an expansion button to display an expanded summary of the entry the expansion button is on. See Chapter 2 [Basic Navigation], page 3.

1 Introduction

To start using speedbar use the command *`M-x speedbar RET`* or select it from the Tools menu in versions of Emacs with speedbar installed by default. This command will open a new frame to summarize the local files. On X Window systems or on MS-Windows, speedbar's frame is twenty characters wide, and will mimic the height of the frame from which it was started. It positions itself to the left or right of the frame you started it from.

To use speedbar effectively, it is important to understand its relationship with the frame you started it from. This frame is the *attached frame* which speedbar will use as a reference point. Once started, speedbar watches the contents of this frame, and attempts to make its contents relevant to the buffer loaded into the attached frame. In addition, all requests made in speedbar that require the display of another buffer will display in the attached frame.

When used in terminal mode, the new frame appears the same size as the terminal. Since it is not visible while working in the attached frame, speedbar will save time by using the *slowbar mode*, where no tracking is done until speedbar is requested to show itself (i.e., the speedbar's frame becomes the selected frame).

The function to use when switching between frames using the keyboard is `speedbar-get-focus`. This function will toggle between frames, and it's useful to bind it to a key in terminal mode. See Chapter 6 [Customizing], page 12.

2 Basic Navigation

Speedbar can display different types of data, and has several display and behavior modes. These modes all have a common behavior, menu system, and look. If one mode is learned, then the other modes are easy to use.

2.1 Basic Key Bindings

These key bindings are common across all modes:

delete, SPC
: Scroll up and down one page.

Q
: Quit speedbar, and kill the frame.

q
: Quit speedbar, and hide the frame. This makes it faster to restore the speedbar frame, than if you press *Q*.

g
: Refresh whatever contents are in speedbar.

t
: Toggle speedbar to and from slowbar mode. In slowbar mode, frame tracking is not done.

n
p
: Move, respectively, to the next or previous item. A summary of that item will be displayed in the attached frame's minibuffer.

M-n
M-p
: Move to the next or previous item in a restricted fashion. If a list is open, the cursor will skip over it. If the cursor is in an open list, it will not leave it.

C-M-n
C-M-n
: Move forwards and backwards across extended groups. This lets you quickly skip over all files, directories, or other common sub-items at the same current depth.

C-x b
: Switch buffers in the attached frame.

Speedbar can handle multiple modes. Two are provided by default. These modes are File mode, and Buffers mode. There are accelerators to switch into these different modes.

b
: Switch into Quick Buffers mode (see Chapter 4 [Buffer Mode], page 10). After one use, the previous display mode is restored.

f
: Switch into File mode.

r
: Switch back to the previous mode.

Some modes provide groups, lists and tags. See Section 2.2 [Basic Visuals], page 4. When these are available, some additional common bindings are available.

RET
e
: Edit/Open the current group or tag. This behavior is dependent on the mode. In general, files or buffers are opened in the attached frame, and directories or group nodes are expanded locally.

`+`
`=` Expand the current group, displaying sub items. When used with a prefix argument, any data that may have been cached is flushed. This is similar to a power click. See Section 2.3 [Mouse Bindings], page 5.

`-` Contract the current group, hiding sub items.

2.2 Basic Visuals

Speedbar has visual cues for indicating different types of data. These cues are used consistently across the different speedbar modes to make them easier to interpret.

At a high level, in File mode, there are directory buttons, sub directory buttons, file buttons, tag buttons, and expansion buttons. This makes it easy to use the mouse to navigate a directory tree, and quickly view files, or a summary of those files.

The most basic visual effect used to distinguish between these button types is color and mouse highlighting. Anything the mouse highlights can be clicked on and is called a button (see Section 2.3 [Mouse Bindings], page 5). Anything not highlighted by the mouse will not be clickable.

Text in speedbar consists of four different types of data. Knowing how to read these textual elements will make it easier to navigate by identifying the types of data available.

2.2.0.1 Groups

Groups summarize information in a single line, and provide a high level view of more complex systems, like a directory tree, or manual chapters.

Groups appear at different indentation levels, and are prefixed with a '`+`' in some sort of 'box'. The group name will summarize the information within it, and the expansion box will display that information inline. In File mode, directories and files are 'groups' where the '`+`' is surrounded by brackets like this:

```
<+> include
<-> src
 [+] foo.c
```

In this example, we see both open and closed directories, in addition to a file. The directories have a box consisting of angle brackets, and a file uses square brackets.

In all modes, a group can be 'edited' by pressing *RET*, meaning a file will be opened, or a directory explicitly opened in speedbar. A group can be expanded or contracted using `+` or `-`. See Section 2.1 [Basic Key Bindings], page 3.

Sometimes groups may have a '`?`' in its indicator box. This means that it is a group type, but there are no contents, or no known way of extracting contents of that group.

When a group has been expanded, the indicator button changes from '`+`' to '`-`'. This indicates that the contents are being shown. Click the '`-`' button to contract the group, or hide the contents currently displayed.

2.2.0.2 Tags

Tags are the leaf nodes of the tree system. Tags are generally prefixed with a simple character, such as '`>`'. Tags can only be jumped to using *RET* or `e`.

2.2.0.3 Boolean Flags

Sometimes a group or tag is given a boolean flag. These flags appear as extra text characters at the end of the line. File mode uses boolean flags, such as a '`*`' to indicate that a file has been checked out of a versioning system.

For additional flags, see Chapter 3 [File Mode], page 7, and Section 6.3 [Version Control], page 14.

2.2.0.4 Unadorned Text

Unadorned text generally starts in column 0, without any special symbols prefixing them. In Buffers mode different buffer groups are prefixed with a description of what the following buffers are (Files, scratch buffers, and invisible buffers.)

Unadorned text will generally be colorless, and not clickable.

2.2.0.5 Color Cues

Each type of Group, item indicator, and label is given a different color. The colors chosen are dependent on whether the background color is light or dark. Of important note is that the 'current item', which may be a buffer or file name, is highlighted red, and underlined.

Colors can be customized from the group `speedbar-faces`. Some modes, such as for Info, will use the Info colors instead of default speedbar colors as an indication of what is currently being displayed.

The face naming convention mirrors the File display mode. Modes which do not use files will attempt to use the same colors on analogous entries.

2.3 Mouse Bindings

The mouse has become a common information navigation tool. Speedbar will use the mouse to navigate file systems, buffer lists, and other data. The different textual cues provide buttons which can be clicked on (see Section 2.2 [Basic Visuals], page 4). Anything that highlights can be clicked on with the mouse, or affected by the menu.

The mouse bindings are:

`Mouse-1` Move cursor to that location.

`Mouse-2`
`Double-Mouse-1`
: Activate the current button. *`Double-Mouse-1`* is called a *double click* on other platforms, and is useful for windows users with two button mice.

`S-Mouse-2`
`S-Double-Mouse-1`
: This has the same effect as *`Mouse-2`*, except it is called a power click. This means that if a group with an expansion button '`+`' is clicked, any caches are flushed, and subitems re-read. If it is a name, it will be opened in a new frame.

`Mouse-3` Activate the speedbar menu. The item selected affects the line clicked, not the line where the cursor was.

Mouse-1 (mode line)
: Activate the menu. This affects the item the cursor is on before the click, since the mouse was not clicked on anything.

C-Mouse-1
: Buffers sub-menu. The buffer in the attached frame is switched.

When the mouse moves over buttons in speedbar, details of that item should be displayed in the minibuffer of the attached frame. Sometimes this can contain extra information such as file permissions, or tag location.

2.4 Displays Submenu

You can display different data by using different display modes. These specialized modes make it easier to navigate the relevant pieces of information, such as files and directories, or buffers.

In the main menu, found by clicking *Mouse-3*, there is a submenu labeled 'Displays'. This submenu lets you easily choose between different display modes.

The contents are modes currently loaded into emacs. By default, this would include Files, Quick Buffers, and Buffers. Other major display modes such as Info are loaded separately.

3 File Mode

File mode displays a summary of your current directory. You can display files in the attached frame, or summarize the tags found in files. You can even see if a file is checked out of a version control system, or has some associated object file.

Advanced behavior, like copying and renaming files, is also provided.

3.1 Directory Display

There are three major sections in the display. The first line or two is the root directory speedbar is currently viewing. You can jump to one of the parent directories by clicking on the name of the directory you wish to jump to.

Next, directories are listed. A directory starts with the group indicator button '`<+>`'. Clicking the directory name makes speedbar load that directory as the root directory for its display. Clicking the '`<+>`' button will list all directories and files beneath.

Next, files are listed. Files start with the group indicator '`[+]`' or '`[?]`'. You can jump to a file in the attached frame by clicking on the file name. You can expand a file and look at its tags by clicking on the '`[+]`' symbol near the file name.

A typical session might look like this:

```
~/lisp/
<+> checkdoc
<+> eieio
<-> speedbar
 [+] Makefile
 [+] rpm.el #
 [+] sb-gud.el #
 [+] sb-info.el #
 [+] sb-rmail.el #
 [+] sb-w3.el
 [-] speedbar.el *!
  {+} Types
  {+} Variables
  {+} def (group)
  {+} speedbar-
 [+] speedbar.texi *
<+> testme
[+] align.el
[+] autoconf.el
```

In this example, you can see several directories. The directory '`speedbar`' has been opened inline. Inside the directory '`speedbar`', the file '`speedbar.el`' has its tags exposed. These tags are extensive, and they are summarized into tag groups.

Files get additional boolean flags associated with them. Valid flags are:

`*` This file has been checked out of a version control system. See Section 6.3 [Version Control], page 14.

`#` This file has an up to date object file associated with it. The variable `speedbar-obj-alist` defines how speedbar determines this value.

`!` This file has an out of date object file associated with it.

A Tag group is prefixed with the symbol '`{+}`'. Clicking this symbol will show all symbols that have been organized into that group. Different types of files have unique tagging methods as defined by their major mode. Tags are generated with either the `imenu` package, or through the `etags` interface.

Tag groups are defined in multiple ways which make it easier to find the tag you are looking for. Imenu keywords explicitly create groups, and speedbar will automatically create groups if tag lists are too long.

In our example, Imenu created the groups '`Types`' and '`Variables`'. All remaining top-level symbols are then regrouped based on the variable `speedbar-tag-hierarchy-method`. The subgroups '`def`' and '`speedbar-`' are groupings where the first few characters of the given symbols are specified in the group name. Some group names may say something like '`speedbar-t to speedbar-v`', indicating that all symbols which alphabetically fall between those categories are included in that sub-group. See Section 6.2 [Tag Hierarchy Methods], page 13.

3.2 Hidden Files

On GNU and Unix systems, a hidden file is a file whose name starts with a period. They are hidden from a regular directory listing because the user is not generally interested in them.

In speedbar, a hidden file is a file which isn't very interesting and might prove distracting to the user. Any uninteresting files are removed from the File display. There are two levels of uninterest in speedbar. The first level of uninterest are files which have no expansion method, or way of extracting tags. The second level is any file that matches the same pattern used for completion in `find-file`. This is derived from the variable `completion-ignored-extensions`.

You can toggle the display of uninteresting files from the toggle menu item '`Show All Files`'. This will display all level one hidden files. These files will be shown with a '`?`' indicator. Level 2 hidden files will still not be shown.

Object files fall into the category of level 2 hidden files. You can determine their presence by the '`#`' and '`!`' file indicators. See Section 3.1 [Directory Display], page 7.

3.3 File Key Bindings

File mode has key bindings permitting different file system operations such as copy or rename. These commands all operate on the *current file*. In this case, the current file is the file at point, or clicked on when pulling up the menu.

U Move the entire speedbar display up one directory.

I Display information in the minibuffer about this line. This is the same information shown when navigating with *n* and *p*, or moving the mouse over an item.

B Byte compile the Emacs Lisp file on this line.

`L` Load the Emacs Lisp file on this line. If a '`.elc`' file exists, optionally load that.

`C` Copy the current file to some other location.

`R` Rename the current file, possibly moving it to some other location.

`D` Delete the current file.

`O` Delete the current file's object file. Use the symbols '`#`' and '`!`' to determine if there is an object file available.

One menu item toggles the display of all available files. By default, only files which Emacs understands, and knows how to convert into a tag list, are shown. By showing all files, additional files such as text files are also displayed, but they are prefixed with the '`[?]`' symbol. This means that it is a file, but Emacs doesn't know how to expand it.

4 Buffer Mode

Buffer mode is very similar to File mode, except that instead of tracking the current directory and all files available there, the current list of Emacs buffers is shown.

These buffers can have their tags expanded in the same way as files, and uses the same unknown file indicator (see Chapter 3 [File Mode], page 7).

Buffer mode does not have file operation bindings, but the following buffer specific key bindings are available:

`k` Kill this buffer. Do not touch its file.

`r` Revert this buffer, reloading from disk.

In addition to Buffer mode, there is also Quick Buffer mode. In fact, Quick Buffers is bound to the `b` key. The only difference between Buffers and Quick Buffers is that after one operation is performed which affects the attached frame, the display is immediately reverted to the last displayed mode.

Thus, if you are in File mode, and you need quick access to a buffer, press `b`, click on the buffer you want, and speedbar will revert back to File mode.

5 Minor Display Modes

For some buffers, a list of files and tags makes no sense. This could be because files are not currently in reference (such as web pages), or that the files you might be interested have special properties (such as email folders.)

In these cases, a minor display mode is needed. A minor display mode will override any major display mode currently being displayed for the duration of the specialized buffer's use. Minor display modes will follow the general rules of their major counterparts in terms of key bindings and visuals, but will have specialized behaviors.

5.1 RMAIL

When using RMAIL, speedbar will display two sections. The first is a layer one reply button. Clicking here will initialize a reply buffer showing only this email address in the 'To:' field.

The second section lists all RMAIL folders in the same directory as your main RMAIL folder. The general rule is that RMAIL folders always appear in all caps, or numbers. It is possible to save mail in folders with lower case letters, but there is no clean way of detecting such RMAIL folders without opening them all.

Each folder can be visited by clicking the name. You can move mail from the current RMAIL folder into a different folder by clicking the '<M>' button. The 'M' stands for Move.

In this way you can manage your existing RMAIL folders fairly easily using the mouse.

5.2 Info

When browsing Info files, all local relevant information is displayed in the info buffer and a topical high-level view is provided in speedbar. All top-level info nodes are shown in the speedbar frame, and can be jumped to by clicking the name.

You can open these nodes with the '[+]' button to see what sub-topics are available. Since these sub-topics are not examined until you click the '[+]' button, sometimes a '[?]' will appear when you click on a '[+]', indicating that there are no sub-topics.

5.3 GDB

If you are debugging an application with GDB in Emacs, speedbar can show you the current stack when the current buffer is the '*gdb*' buffer. Usually, it will just report that there is no stack, but when the application is stopped, the current stack will be shown.

You can click on any stack element and gdb will move to that stack level. You can then check variables local to that level at the GDB prompt.

This mode has the unfortunate side-effect of breaking GDB's repeat feature when you hit *RET* since your previous command is overridden with a stack fetching command.

6 Customizing

Speedbar is highly customizable, with a plethora of control elements. Since speedbar is so visual and reduces so much information, this is an important aspect of its behavior.

In general, there are three custom groups you can use to quickly modify speedbar's behavior.

`speedbar`
: Basic speedbar behaviors.

`speedbar-vc`
: Customizations regarding version control handling.

`speedbar-faces`
: Customize speedbar's many colors and fonts.

6.1 Frames and Faces

There are several faces speedbar generates to provide a consistent color scheme across display types. You can customize these faces using your favorite method. They are:

speedbar-button-face
: Face used on expand/contract buttons.

speedbar-file-face
: Face used on Files. Should also be used on non-directory like nodes.

speedbar-directory-face
: Face used for directories, or nodes which consist of groups of other nodes.

speedbar-tag-face
: Face used for tags in a file, or for leaf items.

speedbar-selected-face
: Face used to highlight the selected item. This would be the current file being edited.

speedbar-highlight-face
: Face used when the mouse passes over a button.

You can also customize speedbar's initial frame parameters. How this is accomplished is dependent on your platform being Emacs or XEmacs.

In Emacs, change the alist `speedbar-frame-parameters`. This variable is used to set up initial details. Height is also automatically added when speedbar is created, though you can override it.

In XEmacs, change the plist `speedbar-frame-plist`. This is the XEmacs way of doing the same thing.

6.2 Tag Hierarchy Methods

When listing tags within a file, it is possible to get an annoyingly long list of entries. Imenu (which generates the tag list in Emacs) will group some classes of items automatically. Even here, however, some tag groups can be quite large.

To solve this problem, tags can be grouped into logical units through a hierarchy processor. The specific variable to use is `speedbar-tag-hierarchy-method`. There are several methods that can be applied in any order. They are:

`speedbar-trim-words-tag-hierarchy`
: Find a common prefix for all elements of a group, and trim it off.

`speedbar-prefix-group-tag-hierarchy`
: If a group is too large, place sets of tags into bins based on common prefixes.

`speedbar-simple-group-tag-hierarchy`
: Take all items in the top level list not in a group, and stick them into a '`Tags`' group.

`speedbar-sort-tag-hierarchy`
: Sort all items, leaving groups on top.

You can also add your own functions to reorganize tags as you see fit.

Some other control variables are:

`speedbar-tag-group-name-minimum-length`
: Default value: 4.

  The minimum length of a prefix group name before expanding. Thus, if the `speedbar-tag-hierarchy-method` includes `speedbar-prefix-group-tag-hierarchy` and one such group's common characters is less than this number of characters, then the group name will be changed to the form of:

  ```
  worda to wordb
  ```

  instead of just

  ```
  word
  ```

  This way we won't get silly looking listings.

`speedbar-tag-split-minimum-length`
: Default value: 20.

  Minimum length before we stop trying to create sub-lists in tags. This is used by all tag-hierarchy methods that break large lists into sub-lists.

`speedbar-tag-regroup-maximum-length`
: Default value: 10.

  Maximum length of submenus that are regrouped. If the regrouping option is used, then if two or more short subgroups are next to each other, then they are combined until this number of items is reached.

6.3 Version Control

When using the file mode in speedbar, information regarding a version control system adds small details to the display. If a file is in a version control system, and is "checked out" or "locked" locally, an asterisk '`*`' appears at the end of the file name. In addition, the directory name for Version Control systems are left out of the speedbar display.

You can easily add new version control systems into speedbar's detection scheme. To make a directory "disappear" from the list, use the variable `speedbar-directory-unshown-regexp`.

Next, you need to write entries for two hooks. The first is `speedbar-vc-path-enable-hook` which will enable a VC check in the current directory for the group of files being checked. Your hook function should take one parameter (the directory to check) and return `t` if your VC method is in control here.

The second function is `speedbar-vc-in-control-hook`. This hook takes two parameters, the *path* of the file to check, and the *file* name. Return `t` if you want to have the asterisk placed near this file.

Lastly, you can change the VC indicator using the variable `speedbar-vc-indicator`, and specify a single character string.

6.4 Hooks

There are several hooks in speedbar allowing custom behaviors to be added. Available hooks are:

`speedbar-visiting-file-hook`
: Hooks run when speedbar visits a file in the selected frame.

`speedbar-visiting-tag-hook`
: Hooks run when speedbar visits a tag in the selected frame.

`speedbar-load-hook`
: Hooks run when speedbar is loaded.

`speedbar-reconfigure-keymaps-hook`
: Hooks run when the keymaps are regenerated. Keymaps are reconfigured whenever modes change. This will let you add custom key bindings.

`speedbar-before-popup-hook`
: Hooks called before popping up the speedbar frame. New frames are often popped up when "power clicking" on an item to view it.

`speedbar-before-delete-hook`
: Hooks called before deleting or hiding the speedbar frame.

`speedbar-mode-hook`
: Hooks called after creating a speedbar buffer.

`speedbar-timer-hook`
: Hooks called after running the speedbar timer function.

`speedbar-scanner-reset-hook`
: Hook called whenever generic scanners are reset. Set this to implement your own scanning or rescan safe functions with state data.

7 Extending

Speedbar can run different types of Major display modes such as Files (see Chapter 3 [File Mode], page 7), and Buffers (see Chapter 4 [Buffer Mode], page 10). It can also manage different minor display modes for use with buffers handling specialized data.

These major and minor display modes are handled through an extension system which permits specialized keymaps and menu extensions, in addition to a unique rendering function. You can also specify a wide range of tagging functions. The default uses `imenu`, but new tagging methods can be easily added. In this chapter, you will learn how to write your own major or minor display modes, and how to create specialized tagging functions.

7.1 Minor Display Modes

A *minor display mode* is a mode useful when using a specific type of buffer. This mode might not be useful for any other kind of data or mode, or may just be more useful that a files or buffers based mode when working with a specialized mode.

Examples that already exist for speedbar include RMAIL, Info, and gdb. These modes display information specific to the major mode shown in the attached frame.

To enable a minor display mode in your favorite Major mode, follow these steps. The string '*name*' is the name of the major mode being augmented with speedbar.

1. Create the keymap variable *name*`-speedbar-key-map`.
2. Create a function, named whatever you like, which assigns values into your keymap. Use this command to create the keymap before assigning bindings:

   ```
   (setq name-speedbar-key-map (speedbar-make-specialized-keymap))
   ```

   This function creates a special keymap for use in speedbar.
3. Call your install function, or assign it to a hook like this:

   ```
   (if (featurep 'speedbar)
       (name-install-speedbar-variables)
     (add-hook 'speedbar-load-hook 'name-install-speedbar-variables))
   ```
4. Create an easymenu compatible vector named *name*`-speedbar-menu-items`. This will be spliced into speedbar's control menu.
5. Create a function called *name*`-speedbar-buttons`. This function should take one variable, which is the buffer for which it will create buttons. At this time `(current-buffer)` will point to the uncleared speedbar buffer.

When writing *name*`-speedbar-buttons`, the first thing you will want to do is execute a check to see if you need to re-create your display. If it needs to be cleared, you need to erase the speedbar buffer yourself, and start drawing buttons. See Section 7.4 [Creating a display], page 19.

7.2 Major Display Modes

Creating a *Major Display Mode* for speedbar requires authoring a keymap, an easy-menu segment, and writing several functions. These items can be given any name, and are made the same way as in a minor display mode (see Section 7.1 [Minor Display Modes], page 16). Once this is done, these items need to be registered.

Because this setup activity may or may not have speedbar available when it is being loaded, it is necessary to create an install function. This function should create and initialize the keymap, and add your expansions into the customization tables.

When creating the keymap, use the function `speedbar-make-specialized-keymap` instead of other keymap making functions. This will provide you with the initial bindings needed. Some common speedbar functions you might want to bind are:

`speedbar-edit-line`
: Edit the item on the current line.

`speedbar-expand-line`
: Expand the item under the cursor. With a numeric argument (*C-u*), flush cached data before expanding.

`speedbar-contract-line`
: Contract the item under the cursor.

These function require that function `speedbar-line-path` be correctly overloaded to work.

Next, register your extension like this;

```
(speedbar-add-expansion-list '("MyExtension"
                               MyExtension-speedbar-menu-items
                               MyExtension-speedbar-key-map
                               MyExtension-speedbar-buttons))
```

There are no limitations to the names you use.

The first parameter is the string representing your display mode. The second parameter is a variable name containing an easymenu compatible menu definition. This will be stuck in the middle of speedbar's menu. The third parameter is the variable name containing the keymap we discussed earlier. The last parameter is a function which draws buttons for your mode. This function must take two parameters. The directory currently being displayed, and the depth at which you should start rendering buttons. The function will then draw (starting at the current cursor position) any buttons deemed necessary based on the input parameters. See Section 7.4 [Creating a display], page 19.

Next, you need to register function overrides. This may look something like this:

```
(speedbar-add-mode-functions-list
 '("MYEXTENSION"
   (speedbar-item-info . MyExtension-speedbar-item-info)
   (speedbar-line-path . MyExtension-speedbar-line-path)))
```

The first element in the list is the name of you extension. The second is an alist of functions to overload. The function to overload is first, followed by what you want called instead.

For `speedbar-line-path` your function should take an optional DEPTH parameter. This is the starting depth for heavily indented lines. If it is not provided, you can derive it like this:

```
(save-match-data
  (if (not depth)
      (progn
        (beginning-of-line)
        (looking-at "^\\([0-9]+\\):")
        (setq depth (string-to-int (match-string 1)))))
```

where the depth is stored as invisible text at the beginning of each line.

The path returned should be the full path name of the file associated with that line. If the cursor is on a tag, then the file containing that tag should be returned. This is critical for built in file based functions to work (meaning less code for you to write). If your display does not deal in files, you do not need to overload this function.

The function `speedbar-item-info`, however, is very likely to need overloading. This function takes no parameters and must derive a text summary to display in the minibuffer.

There are several helper functions you can use if you are going to use built in tagging. These functions can be `or`ed since each one returns non-nil if it displays a message. They are:

`speedbar-item-info-file-helper`
: This takes an optional *filename* parameter. You can derive your own filename, or it will derive it using a (possibly overloaded) function `speedbar-line-file`. It shows details about a file.

`speedbar-item-info-tag-helper`
: If the current line is a tag, then display information about that tag, such as its parent file, and location.

Your custom function might look like this:

```
(defun MyExtension-item-info ()
  "Display information about the current line."
  (or (speedbar-item-info-tag-helper)
      (message "Interesting detail.")))
```

Once you have done all this, speedbar will show an entry in the '`Displays`' menu declaring that your extension is available.

7.3 Tagging Extensions

It is possible to create new methods for tagging files in speedbar. To do this, you need two basic functions, one function to fetch the tags from a buffer, the other to insert them below the filename.

my-fetch-dynamic-tags *file* Function

Parse *file* for a list of tags. Return the list, or `t` if there was an error.

The non-error return value can be anything, as long as it can be inserted by its paired function:

my-insert-tag-list *level lst* Function

Insert a list of tags *lst* started at indentation level *level*. Creates buttons for each tag, and provides any other display information required.

It is often useful to use `speedbar-create-tag-hierarchy` on your token list. See that function's documentation for details on what it requires.

Once these two functions are written, modify the variable `speedbar-dynamic-tags-function-list` to include your parser at the beginning, like this:

```
(add-to-list 'speedbar-dynamic-tags-function-list
     '(my-fetch-dynamic-tags  . my-insert-tag-list))
```

If your parser is only good for a few types of files, make sure that it is either a buffer local modification, or that the tag generator returns `t` for non valid buffers.

7.4 Creating a display

Rendering a display in speedbar is completely flexible. When your button function is called, see Section 7.1 [Minor Display Modes], page 16, and Section 7.2 [Major Display Modes], page 16, you have control to `insert` anything you want.

The conventions allow almost anything to be inserted, but several helper functions are provided to make it easy to create the standardized buttons.

To understand the built in functions, each 'button' in speedbar consists of four important pieces of data. The text to be displayed, token data to be associated with the text, a function to call, and some face to display it in.

When a function is provided, then that text becomes mouse activated, meaning the mouse will highlight the text.

Additionally, for data which can form deep trees, each line is given a depth which indicates how far down the tree it is. This information is stored in invisible text at the beginning of each line, and is used by the navigation commands.

speedbar-insert-button *text face mouse function* &optional *token prevline* — Function

This function inserts one button into the current location. *text* is the text to insert. *face* is the face in which it will be displayed. *mouse* is the face to display over the text when the mouse passes over it. *function* is called whenever the user clicks on the text.

The optional argument *token* is extra data to associated with the text. Lastly *prevline* should be non-nil if you want this line to appear directly after the last button which was created instead of on the next line.

speedbar-make-tag-line *exp-button-type exp-button-char exp-button-function exp-button-data tag-button tag-button-function tag-button-data tag-button-face depth* — Function

Create a tag line with *exp-button-type* for the small expansion button. This is the button that expands or contracts a node (if applicable), and *exp-button-char* the character in it ('`+`', '`-`', '`?`', etc). *exp-button-function* is the function to call if it's clicked on. Button types are `'bracket`, `'angle`, `'curly`, `'expandtag`, `'statictag`, or nil. *exp-button-data* is extra data attached to the text forming the expansion button.

Next, *tag-button* is the text of the tag. *tag-button-function* is the function to call if clicked on, and *tag-button-data* is the data to attach to the text field (such a tag positioning, etc). *tag-button-face* is a face used for this type of tag.

Lastly, *depth* shows the depth of expansion.

This function assumes that the cursor is in the speedbar window at the position to insert a new item, and that the new item will end with a CR.

speedbar-insert-generic-list *level list expand-fun find-fun* Function

At *level*, (the current indentation level desired) insert a generic multi-level alist *list*. Associations with lists get '`{+}`' tags (to expand into more nodes) and those with positions or other data just get a '`>`' as the indicator. '`{+}`' buttons will have the function *expand-fun* and the token is the `cdr` list. The token name will have the function *find-fun* and not token.

Each element of the list can have one of these forms:

`(`*name* `. marker-or-number)`
: One tag at this level.

`(`*name* `(`*name* `. marker-or-number) (`*name* `. marker-or-number) ... )`
: One group of tags.

`(`*name* `marker-or-number (`*name* `. marker-or-number) ... )`
: One Group of tags where the group has a starting position.

When you use `speedbar-insert-generic-list`, there are some variables you can set buffer-locally to change the behavior. The most obvious is `speedbar-tag-hierarchy-method`. See Section 6.2 [Tag Hierarchy Methods], page 13.

speedbar-generic-list-group-expand-button-type Variable

This is the button type used for groups of tags, whether expanded or added in via a hierarchy method. Two good values are `'curly` and `'expandtag`. Curly is the default button, and `'expandtag` is useful if the groups also has a position.

speedbar-generic-list-tag-button-type Variable

This is the button type used for a single tag. Two good values are `nil` and `'statictag`. `nil` is the default, and `'statictag` has the same width as `'expandtag`.

Concept Index

T

V