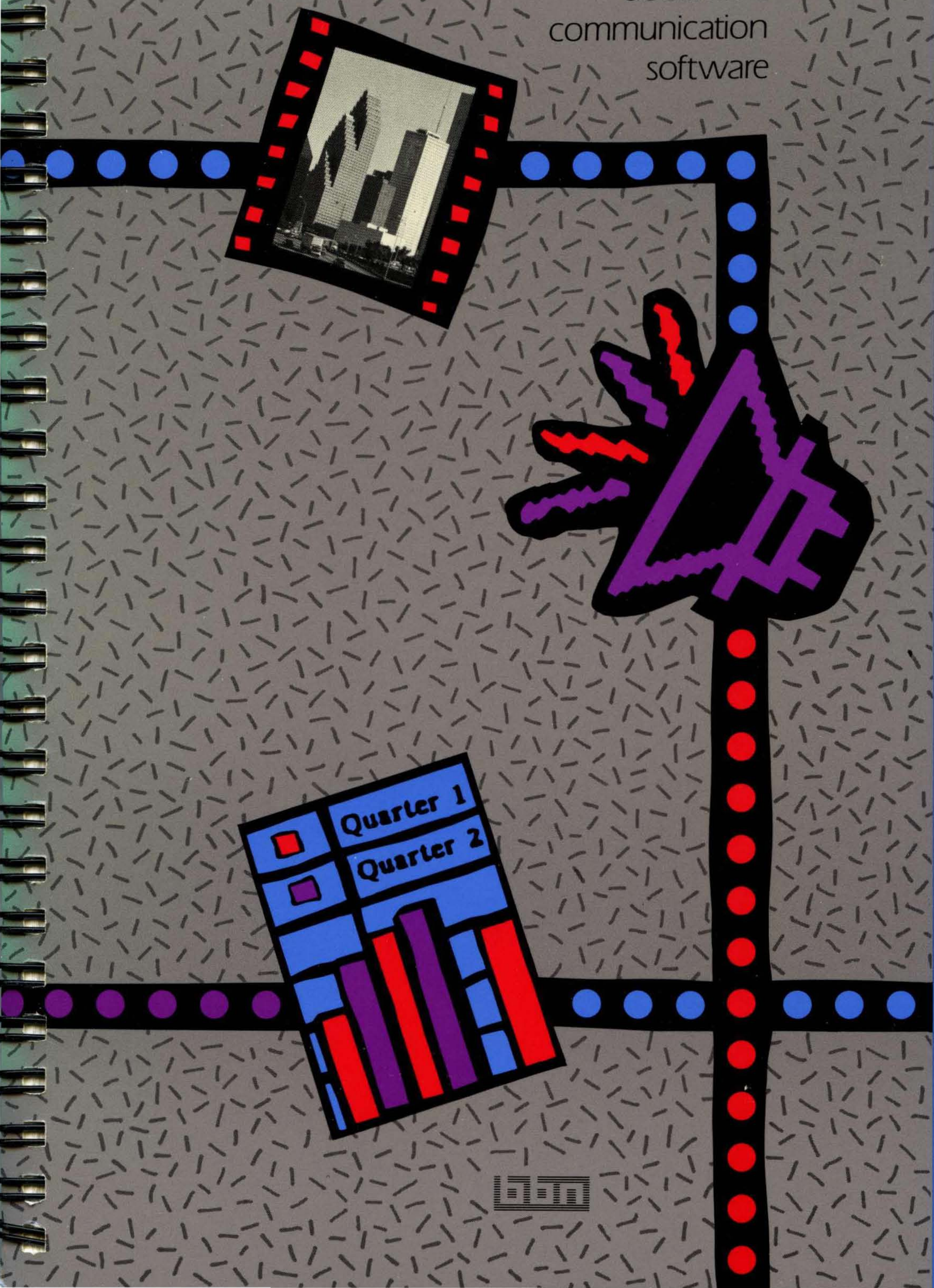
multimedia
document
communication
software



BBN/Slate™

Customizing Manual

multimedia
document
communication
software

BBN/Slate™

This document reflects the BBN/Slate Release 1.1 software.

Customizing Manual

BBN Software Products
A Division of Bolt Beranek and Newman Inc.

BBN Software Products
A Division of Bolt Beranek and Newman Inc
150 CambridgePark Drive
Cambridge, MA 02140

Latest Printing: October 1990

Contents

Preface

This preface contains the following:

- an overview of this manual
- a list of the documentation conventions used in this manual
- an overview of the BBN/Slate™ documentation

About this manual

This is the *Customizing Manual* for the BBN/Slate multimedia document communications system. It introduces you to the ways in which you can adapt BBN/Slate to fit your particular needs. For example, it describes how to reconfigure the BBN/Slate menu system, and how to employ familiar keyboard sequences to assist you in performing document creation and editing. This manual also introduces you to how you can write your own programs that can further extend the capabilities of BBN/Slate.

This *Customizing Manual* contains three chapters and two appendices:

- Chapter 1 discusses *Customizing Basics*, an introduction to BBN/Slate profile files.

- Chapter 2 presents the tools for *Configurable Menus*, so you can change your BBN/Slate menu environment as appropriate.

- Chapter 3 describes the *BBN/Slate Extension Language*, a powerful programming tool to extend the capabilities of your BBN/Slate software and alter your BBN/Slate environment.

- Appendix A lists the *Customizing Variables* that you can employ to control the behavior of BBN/Slate.

- Appendix B is a complete *List of Functions* that provides a reference for all BBN/Slate functions provided with this software release.

If you are interested in learning more in general about how BBN/Slate works (for example, the structure of BBN/Slate document files), we recommend that you also read the *System Topics* manual.

Documentation conventions for this manual

Unless noted in the text, this book uses the conventions described in this section.

Mouse operations

This book uses the following format for describing mouse operations:

BUTTON action

or

modifier BUTTON action

where:

- **BUTTON** is the left, middle, or right button.
- **Action** is click, hold, drag, or release.
- **Modifier** is the key you hold down while performing a mouse operation, either the control or shift key.

For example:

LEFT click means you left click the left mouse button.

Shift MIDDLE hold means you press and hold the middle mouse button while holding down the shift key on the keyboard.

Menu commands

Menu commands are given with the form:

menuname-commandname

For example, the following menu command:

Text-Fonts-Face-Italics

shows that **Text** is the top-level menu, **Fonts** is a submenu of **Text**, **Face** is a submenu of **Fonts**, and **Italics** is an option of the **Face** menu.

Boldface font

This boldface font is used for menu activities and mouse operations. For example:

> You can adjust the spacing of points in the grid with the **Graphics-Style-Grid** command.
>
> To listen to a speech passage in a document, you **LEFT click** while the mouse pointer is over the icon or passage.

Boldface is also used for keys you press on the keyboard, such as the **Return** key.

Italic font

This italic font is used to give emphasis, especially when a new term is introduced in the text. In discussions of command-line activity or programming, italic represents a placeholder in command syntaxes and directory specifications. For example, in the following command syntax:

```
% install/install_lang package
```

package is a placeholder for the actual name of the language package you are installing.

Monospace font

`This monospace font` is primarily used in descriptions of command-line activity and programming. It is used to represent filenames, directories, BBN/Slate commands and functions, UNIX™ operating-system commands, prompts and messages you see on your screen, and user input. For example:

> You must define the program by adding it with an `autoload` statement to your `.slate_editor.init` file.

BBN/Slate folder names are also represented in monospace font.

Control key

For key sequences that use the control key, a caret (^) indicates the control key. For example, **^A** means that you type the control key and the letter A together. You can type A in upper- or lowercase.

Escape key

The escape key is represented by **Esc**, as in **Esc-D**. In this example, you type the escape key and an uppercase D together. You would not type a lowercase d; case matters when you are using the escape key.

Alternate key

Key sequences that use the alternate key represent the key with **Alt**, as in **Alt-w**. In this example, you type the alternate key together with lowercase w. Again, case matters when you use **Alt**. Please see the *Release Notes* for information about which key serves as the **Alt** key on your system.

Overview of BBN/Slate documentation

User documentation

The BBN/Slate documentation set includes the following manuals:

- *Getting Started* introduces you to BBN/Slate through a series of tutorial exercises.
- The *Reference Manual* gives a complete description of the BBN/Slate system.
- The *Customizing Manual* explains how you can tailor the BBN/Slate system to fit your needs. Topics include reconfiguring the menu system, and writing your own programs using the Slate Extension Language (SEL).
- *System Topics* is a collection of documents covering system topics such as the structure of BBN/Slate document files, command-line document management tools, and mail utility programs.
- *Multilingual Documents* describes how to create documents in five other writing systems (Arabic, Cyrillic, Hangul, Hebrew, and Thai) available with the BBN/Slate Multilingual Option. *Note*: If you did not order the Multilingual Option, you will not receive this manual.

On-line help

In addition to the printed documentation, BBN/Slate has an on-line menu help facility that provides information on all command and menu choices. Also, BBN/Slate provides on-line UNIX manual pages with information about BBN/Slate's command-line programs and utilities.

Installation and release notes

System-specific installation instructions and release notes supplement the BBN/Slate user documentation set.

Obtaining Documentation and Services

This section explains how you can:

- order documentation
- obtain technical assistance
- obtain training services

How to order documentation

To order additional copies of this manual or any other BBN/Slate manual:

- In North America, mail a completed purchase order to:

  BBN Software Products
  A Division of Bolt Beranek and Newman Inc.
  10 Fawcett Street
  Cambridge, MA 02138
  ATTN: Sales Administration

  For prices and other ordering information, call Sales Administration at 617/873-5115.

- Outside North America, contact the nearest Sales and Support office.

How to obtain technical assistance

For BBN/Slate customers who are on maintenance and need technical assistance:

- In North America, call the Hotline number: 617/873-3968. Hotline hours are Monday through Friday, 8:30 a.m. to 8 p.m. (EST).
- Outside North America, contact the nearest Sales and Support office.

How to obtain training

BBN supports its customers and products with a full range of training programs. BBN/Slate courses are offered in conjunction with hardware vendors or at BBN education centers. In addition, arrangements can be made with the Education Services department to present the same courses at customer sites.

To receive a course catalog or additional information:

- In North America, call the Course Registrar at 617/873-8383.
- Outside North America, contact the nearest Sales and Support office.

1 Customizing Basics

BBN/Slate software has been designed with *customization* in mind. You have the power to change BBN/Slate in many ways. You should feel free to experiment with some or all of the features described in this manual, so that you can tailor the system to your needs. Customizing BBN/Slate means that you can feel more comfortable. You can make it more like environments you already know; or, you can extend it to do particular tasks you need that are not included in the BBN/Slate software as distributed.

BBN/Slate customization is very flexible. Features can be available to all users (*public features*) or available to a single user (*private features*).

It is possible to create a number of helpful tools for yourself or other BBN/Slate users at your site. Some of these customized tools might serve a simple purpose. For example, you could create an extension language function that automatically inserts a graphic image of your signature at the current active cursor position in your BBN/Slate file. You could then associate this function to a key binding, so with just a few keystrokes, you can add your personal signature to your electronic memos.

BBN/Slate customization also makes far more complex customized tools possible for your site: a mail-merge program to handle the addressing chores for letters to your customers; an external data-handler program that imports numbers from an outside source into BBN/Slate, creates a spreadsheet and generates a graph, prints it, and exports summary data in a format for use by another external program; and more. Several sample programs are included in this manual to assist you in getting started.

The organizational center of BBN/Slate customization is its *profile files.* This chapter focuses on initializing profile files, setting preference variables, key binding functions, and creating mail aliases. Subsequent chapters and appendices present in-depth information on customizing BBN/Slate menus and writing your own custom functions using the BBN/Slate extension language.

1.1 Initialization and profile files

Every time you start the Document Editor (either when you start BBN/Slate, or when you enter the command `editdoc` from the operating system), `editdoc` begins with a search for a public and a private *profile file* (Figure 1-1). The content of the profile files determines the resulting "look and feel," or *configuration,* of BBN/Slate.

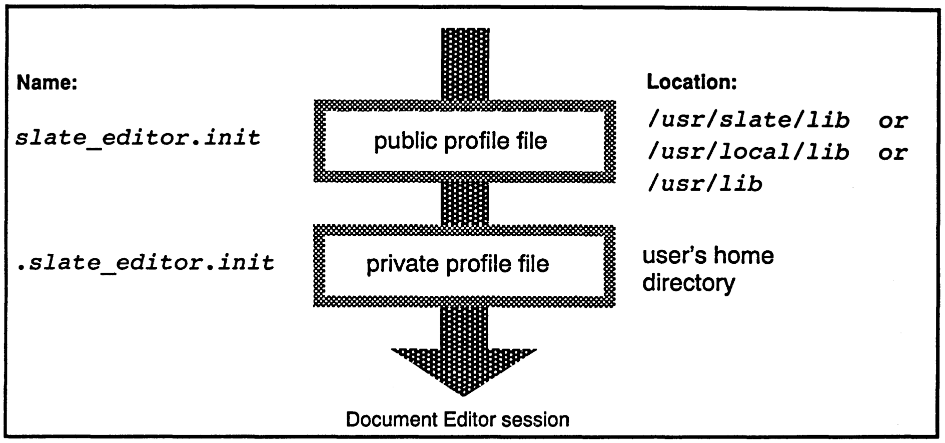


Figure 1-1 Document Editor initialization

By including customization commands in the public profile file, a system administrator can make the commands take effect for everyone who uses a particular installation of BBN/Slate. By including the commands in your private profile file instead, you make them take effect only when you are using BBN/Slate. This makes it possible for each individual BBN/Slate user to create the editing environment in which he or she is most comfortable and productive.

Public profile file

The public file, called `slate_editor.init,` is optional. It can be found in `/usr/slate/lib,` `/usr/local/lib,` or `/usr/lib,` depending on where the `lib` directory was placed by your system administrator at installation. The public profile file allows all BBN/Slate users on the system to share commonly defined Document Editor characteristics, key bindings, and so on.

There is no public profile file distributed with the BBN/Slate software, since it should be tailored to your specific site. However, there is a sample profile file, called `slate_editor.sample_init`, also located in this directory. You can use it as a template for designing your own public or private profile files. If desired, you could make a copy serve as your public profile file by entering the following command from the `lib` directory:

```
cp slate_editor.sample_init slate_editor.init
```

and then editing the `slate_editor.init` file as appropriate for your site.

Private profile file

The private file, called `.slate_editor.init`, is also optional. If it exists, it should be located in each BBN/Slate user's home directory. (Note that because this file name begins with a period, it does not normally appear when listing directory contents.)

The private profile file can contain the same kinds of statements as the public profile file. The only difference is that they will be in effect only for the individual user.

As with the public profile file, there is no private profile file distributed with the BBN/Slate software. However, it is easy for users to share helpful features by copying those portions of their private profile files into the private profile files of others. If many users want to share the same features, then it makes sense to place them within one public profile file rather than in the individual private profile files.

1.1.1 Contents of a profile file

Profile files can contain a variety of statements. These include:

- search paths
- key bindings (general, and for specific media types)
- global variables (both Boolean and value-specific) that control the BBN/Slate environment
- alias definitions that allow the use of nicknames for addressing mail
- extension language definitions and commands
- autoload statements to read files containing extension language definitions

Figure 1-2 is a sample profile file containing each of these types of statements. (Note that a comprehensive profile file could be much larger.)

```
# .slate_editor.init file

# SEARCH PATHS
menu-directory      = "/home/smith/Slate/Menus";
library-search-path = "/home/smith/Slate/sel";

# KEY BINDINGS
# This takes effect everywhere, in all media editors:
bind( default-keymap, "Esc-x", execute-command-dialog);
# Emacs text bindings:
bind( text-keymap,    "^@",    text-set-mark);
bind( text-keymap,    "Esc-f", text-forward-word);
bind( text-keymap,    "^O",    text-open-line);
# Spreadsheet bindings:
bind( sheet-keymap,   "^Xn",   sheet-format-dialog);
bind( sheet-keymap,   "^Xr",   sheet-ruling-dialog);
# Graphics bindings:
bind( graphics-keymap,"B",     graphics-add-rounded-box);
bind( graphics-keymap,"t",     graphics-add-text);

# GLOBAL PREFERENCE VARIABLES
auto-backup             = 1;
header-name-font        = "helvetica14b";
header-value-font       = "helvetica14";
page-width              = "7.0i";
print-bottom-margin     = "1.0i";
spreadsheet-font        = "helvetica10";
print-spooler           = "lpr %s";
mail-spooler = "/usr/mmdf/lib/submit -mrlxto,cc,bcc < %s";
mail-spooler-successful-exitval = 9;

# MAIL ALIASES
alias( "beth",          "bchampion" );
alias( "bruce",         "bsmith" );
```

(continued)

Figure 1-2 **Sample profile file**

```
# EXTENSION LANGUAGE DEFINITIONS
# Function to execute a buffer of extended commands:
define public void execute-buffer()
"Execute a buffer of extended commands."
{
  write-buffer-to-file-as-text(current-buffer(),
    "/tmp/SELtest");
  read-commands-from-file("/tmp/SELtest");
}
# Key bindings for above.  Allows you to create a buffer
# with extended commands, then execute them with ^X^E
bind(text-keymap,       "^X^X", read-commands);
bind(text-keymap,       "^X^E", execute-buffer);

#AUTOLOAD COMMANDS
# Functions for slide-making program:
autoload (set-region-to-copy, "make_slides.sel");
autoload (copy-region-to-file, "make_slides.sel");
autoload (make-slides, "make_slides.sel",
  text-keymap, "Esc-'");
```

Figure 1-2 **Sample profile file** *(continued)*

Each of these types of profile statements is briefly described below. Key bindings, preference variables, and mail aliases are fully described later in this chapter. Search paths, extension language definitions, and autoload statements are fully described elsewhere.

Search paths

These are statements that direct BBN/Slate where to look for certain objects. There are currently two different search paths you can include.

Include a `menu-directory` variable to specify where you have placed any custom-configured BBN/Slate menu files. Configurable menus are discussed in Chapter 2.

Include a `library-search-path` variable to specify those directories where you store files of BBN/Slate extension language definitions. Extension language search paths are discussed in Chapter 3.

Key bindings

These are one of the most common ways to customize BBN/Slate. For example, if you are accustomed to using certain key sequences for performing basic editing functions (deleting lines, selecting text, selecting files, and so on), you can bind those key sequences to perform the same functions in BBN/Slate. It is possible to create a very comprehensive set of key bindings that closely map an existing editing environment (for example, `emacs` or `vi`). You can also define key bindings that take advantage of function keys available for a particular keyboard model.

Key bindings are discussed in detail in Section 1.3.

Global preference variables

You can set global variables to control certain appearance and behavior features of BBN/Slate. See Section 1.2 for more details.

Mail aliases

Mail aliases provide a convenient shorthand naming convention for people with whom you correspond frequently. They are described in detail in Section 1.4.

Extension language definitions

It is possible to include complete BBN/Slate extension language definitions in the profile file (see Chapter 3). Normally, this should be restricted to a few frequently used definitions to minimize startup time. It is more efficient to access your definitions via `autoload` commands.

Autoload commands

The *autoload* command publishes a function (making it available for use), but restrains reading the function definition until you actually invoke it. Thus, `autoload` allows you to include a large number of functions without incurring the overhead of reading their function definitions at startup. This speeds up the startup time for `editdoc` when it reads a public or private profile file.

1.1.2 Editing a profile file

A profile file is a *plain text* file, not a BBN/Slate file. This means that you should edit it in one of the following ways:

- outside of BBN/Slate (for example, using a text editor such as `vi`)
- within BBN/Slate (reading and storing the definition as a text file)

If you edit a profile file with BBN/Slate, always be sure to treat the file as a text file.

Testing new features

It is best to add one or two new features at a time to any profile file, then test them. You can quickly test your features or changes by entering the operating system command:

```
editdoc filename
```

where *filename* is the optional name of any BBN/Slate file (if appropriate, you can include a path as part of the filename). For example:

```
editdoc /home/smith/Slate/test.slt
```

If an error is encountered while reading the profile file, BBN/Slate displays a dialog box that reports the error and the line number on which it was found. You can then correct the error and repeat the test.

1.1.3 Order of profile statements

The order of statements is not important in your profile file, with the exception that search paths should be located at or near the top of the file. This is because the `library-search-path` must be defined before any statements that include extension language definitions. It is helpful to organize and group your statements by type, as illustrated by Figure 1-2.

1.1.4 Syntax and definitions in profile files

Profile files contain definitions and commands that use the BBN/Slate extension language. Accordingly, their definitions and commands follow the BBN/Slate extension language syntax. Profile file examples used in this chapter introduce that syntax. You can find a formal definition in Chapter 3.

The Document Editor reads and executes the extension language commands *after* reading any profile files. If an error is encountered in the profile file or a file containing extension language commands, any remaining definitions and functions are not read. Therefore, it is generally advisable to either fix the error or disable it (for example, by commenting it out), and then restart the Document Editor so that the normal complement of functions are available.

Definitions and redefinitions

The definitions obtained from the public profile file are augmented by those in the private profile file. Because the private file is read last, definitions in that file can also implicitly *redefine* existing public functions.

For example, if the public profile file contains the definitions:

```
bind(default-keymap, "Esc-[210z",
    text-end-of-paragraph);  #R3 key
bind(default-keymap, "^X=",
    text-underline-off);     #CTRL-X=
```

and the private profile file contains the definitions:

```
bind(default-keymap, "Esc-[210z",
    text-end-of-sentence);   # R3 key
bind(default-keymap, "^Xx",
    text-underline-off);     # CTRL-Xx
```

then the private profile file:

- redefines the R3 key (Esc-[210z) to perform the function **text-end-of-sentence** instead of **text-end-of-paragraph**
- defines CTRL-Xx (in addition to CTRL-X=) to perform the function **text-underline-off**

1.1.5 Comments in profile files

You can include comments in your editor profile file to describe its contents. When reading profile files, if the editor encounters a pound sign (#) character on any line, it ignores the rest of that line.

Note that comments can occupy an entire line:

```
# KEY BINDINGS
```

or they may occur at the end of a line, following a statement:

```
bind(text-keymap, "Esc-[211z", previous-pane);  # F1 key
bind(text-keymap, "Esc-[213z", next-pane);      # F2 key
```

In this example, note that including a comment that associates the escape sequence to a particular function key could be helpful as you maintain your profile file later on.

As illustrated in Figure 1-2, it can be helpful to use comments throughout a profile file and to use them to differentiate the sections that are present. For example:

```
###################
# KEY BINDINGS
###################
```

The next section describes how to set global variables in your profile files.

1.2 Setting global preference variables

There are a number of variables that you can set to control the behavior of certain editing commands. Some variables have numeric values; others have string values.

You set variables by including *assignment statements* in your profile file. Assignment statements look like this:

```
variable = value;
```

The semicolon is optional, but it helps to mark the end of the assignment statement. The *value* may be a number or a string surrounded by double quotes, depending on the type of the variable you are setting. To embed double quote marks in string values, precede them with a backslash character (\).

The following are all examples of valid assignment statements:

```
default-document-template = "simple";
header-name-font          = "helvetica10b";
display-classify-menu     = 1;
visit-splits-pane         = 1;
```

See Appendix A for lists of the variables you can use to customize the editor.

The `default-document-template` variable is particularly useful, because it allows you to specify the default text style sheets that are used when you create a new document. (See the *Reference Manual* for more information on document templates.)

1.3 Key bindings

1.3.1 Functions and key bindings

Each menu command in the BBN/Slate Document Editor works by invoking a *function* to perform the actual editing task you have selected. These functions have names so that you can refer to them directly.

You can associate any function with a key or sequence of keys on the keyboard. This association of a function to a key sequence is called a *key binding*, and the function is *bound to* the key sequence you have chosen. BBN/Slate comes with a set of default bindings that are described in the *Reference Manual*, but you can change them to suit your own preferences.

Keymaps

When you are editing documents composed of many different media elements, you may want the same key to mean different things in the different media editors. For instance, you might want **^B** to mean "add a box" when editing a graphics element, but mean "back up one character" when editing text.

To allow for this kind of context-sensitive key binding, BBN/Slate supports the notion of *keymaps*. A keymap is a collection of related key bindings. Each keymap has its own name, so you can refer to them individually.

Each media type in BBN/Slate has its own keymap. When you press keys on the keyboard, BBN/Slate looks up the associated function in the keymap of whatever media type you are editing. Table 1-1 lists the keymap name for each of the media types.

Table 1-1 **Keymap names**

Keymap Name	Media Type
default-keymap	none selected
header-keymap	document headers
text-keymap	text
graphics-keymap	graphics
image-keymap	image
speech-keymap	speech
sheet-keymap	spreadsheet
picture-keymap	rasterfile

The `default-keymap` is the keymap used when not editing a particular media type (for example, when there is no cursor displayed for any media type upon opening a file). It is also used if the keys you type are not found in the media editor keymap. Thus, binding a key in the default keymap is one way to make that binding take effect in all media types.

1.3.2 Using the `bind` command

To create a key binding, you must specify three things: the key you want to bind, the function you want bound to it, and the keymap in which you want the binding to occur.

You specify these things using a `bind` command in your profile file. The syntax of a `bind` command is:

```
bind(keymap, key-sequence, action-routine);
```

The semicolon is optional, but helps to mark the end of the bind command. An example of a `bind` command is:

```
bind(text-keymap, "Esc-k", display-keyboard);
```

This command binds the key sequence Esc-k to the function `display-keyboard`. You can use it whenever a text region (which uses the `text-keymap)` is active.

The *keymap* is one of the keymap names listed in the previous section, and determines which media types this binding will apply to. When you include bind commands in your profile file, the bindings for the default-keymap should precede bindings for all of the other keymaps.

The *action-routine* is the name of one of the editor's functions. The *Reference Manual* lists all of the menu commands in the editor and their associated functions. Appendix B lists all of the functions available in the BBN/Slate system.

The *key-sequence* specifies the key or keys you must press to invoke the function you have specified. The key sequence is surrounded by double quotes, and can include any of the following items:

Simple characters

These are simple alphanumeric or punctuation keys, like **a**.

Control characters

You specify control characters as **^a** or **^A**. You type them by holding down the **Control** key while typing the character that follows the ^. Case does not matter when typing control characters.

Note that when placing **^a** in the `bind` command of your profile file, you can simply insert the characters ^ and **a**; you need not insert the actual control sequence, although it will be recognized if used.

Alt characters

You specify **Alt** characters as **Alt-a** or **Alt-A**. You type them by holding down the **Alternate** key while typing the character that follows the **Alt-**. Please see the *Release Notes* for clarification on which key serves as the **Alternate** key for your system (its function may be performed by a key with a different label than **Alt** or **Alternate**).

Case *does* matter when typing **Alt** characters.

On some systems, **Alt** characters are sometimes used to access special ASCII characters not available directly from the keyboard. These characters can be accessed by including them in `bind` statements in a public or private profile file. For example:

```
bind(text-keymap, "Alt-D", text-self-insert); #Tilde
```

The actual bindings will vary with the specific keyboard and system being used. The documentation for your operating system and keyboard may be of assistance. You can also experiment with special key combinations in the Document Editor by using the `text-quote-next-character` function (the default binding for which is **^Q**).

For example, if you can enter **^Q Alt-7** to create a bullet character (•) on your system, then adding the `bind` statement

```
bind(text-keymap, "Alt-7", text-self-insert); #Bullet
```

to your profile file allows you to create the bullet character simply by pressing **Alt-7**.

Escape characters

You specify escape characters as **Esc-a** or **Esc-A**. You can also represent escape as **ESC-**.

You indicate them by typing the **Esc** key, and then typing the character that follows the **Esc-**. Case *does* matter when typing escape characters.

If you need extra escape sequences, you can also specify two escapes, such as **Esc-Esc-D** (press the **Esc** key twice, then the **D** key).

Mouse buttons

You specify mouse buttons by giving an optional modifier (**Ctrl** or **Shift**), followed by the button name (**LEFT**, **MIDDLE**, or **RIGHT**) and then the button action (**click** or **hold**).

For example, **LEFT click** means clicking the left mouse button. **Shift MIDDLE hold** means pressing and holding the middle mouse button while holding down the shift key on the keyboard.

The case of characters does not matter when specifying mouse buttons.

It is common to bind two-character sequences to a command. For example:

```
bind(default-keymap, "^X^S", save-current-buffer);
bind(default-keymap, "^Xt", keyboard-template-select);
```

In the first example, you must type both the **^X** and the **^S** before the function is invoked. When you type **^X**, the editor waits to see what you type next. The **^X** is a *prefix character*, because it prefixes another keystroke in the keymap. If after typing **^X** you type **^S**, the editor invokes the function for **^X^S** (in this case, `save-current-buffer`). If you type something else, it looks to see whether that key sequence has anything bound to it, or whether it might be the prefix for yet another command.

You can use the same prefix character for a group of related commands. For instance, in the default keymap **^X^S** means "save file" and **^X^R** means "read file". The two commands are similar because they both work on files, so they are grouped together under the **^X** prefix.

If you could not use prefix characters, you could only bind functions to the keyboard until you ran out of keys. Using prefix characters, there is no limit to the number of functions you can bind to the keyboard. However, once you use a character as a prefix, it cannot be used as a simple character. For example, the default BBN/Slate software uses two prefix characters:

```
^X
Esc
```

This means you cannot bind any functions to just **^X** or **Esc**.

Prefix characters are also useful when you want to bind functions to the function keys on your keyboard. When you press one of the function keys, it can generate a sequence of several characters. Because BBN/Slate allows you to use arbitrarily long character sequences for any command, you can still make bindings to the function keys.

Consult your *Release Notes* for lists of the sequences generated by each of the function keys, and for some examples of how to bind editing functions to them.

Special key combinations. Within some media types, it can be appropriate to bind regular alphabetic keys to functions. For example, in the graphics media type, you are normally involved with drawing pictures and choosing actions from the graphics palette. You only use text when labeling parts of the graphic. In this context, it can be appropriate to bind the alphabetic key **t** to initiate the `graphics-add-text` function, so that you simply press the **t** key and begin entering text within the graphic. The bind command for this is:

```
bind(graphics-keymap, "t", graphics-add-text);
```

This binding, together with many other related ones, is included in the sample profile file `slate_editor.sample_init`.

1.3.3 Creating key bindings

There are two basic sources of functions for key bindings:

1. User-callable functions published in the Document Editor; you can find a listing of the BBN/Slate functions that accompany this software release in Appendix B of this manual.
2. Extension language functions that you write yourself and define as public. Chapter 3 describes how to do this.

There is an important exception for both these sources: functions that accept arguments or return values cannot be bound to key sequences. You can easily work around this limitation by providing a top-level extension language function that does not require arguments or return values, but which calls sub-functions that handle obtaining or returning the required information (see Chapter 3).

1.3.4 Emulating another text editor

Often, users come to BBN/Slate with experience using other text editors. They may find it helpful to have BBN/Slate emulate another editor, so many common keyboard commands (to delete lines, move the cursor, mark areas of text, and so on) use familiar key sequences.

Because different software applications almost never have *exactly* the same set of features, you should not expect to map all of the key sequences. Additionally, BBN/Slate can accommodate many more media types than ordinary text editors, so, for example, there may not be a correspondence for handling speech recordings. However, defining even a few of the key sequences that are the most common and familiar to you can help you quickly "feel at home" with BBN/Slate.

Some common key sequences used by an emacs text editor are shown in Table 1-2. (There are many variations of emacs, some using different key sequences than those shown here.) Although emacs has many more functions than those shown here, the principal remains the same for any emulation.

Note that both emacs and BBN/Slate are capable of displaying all key bindings, thus establishing the initial portion of this listing. You can do this in BBN/Slate with the function display-current-keymap-in-buffer. It is helpful to print the list, and then pare it down to those functions you want to emulate.

Use the first column of Table 1-2 to record the names of BBN/Slate functions that correspond to emacs functions. In some cases, no corresponding function is applicable, and is marked "N.A:". In other cases, the default BBN/Slate key binding is identical, so no binding is necessary. These functions are marked with an asterisk (*). Some emacs functions do not have direct equivalents, but could be easily emulated through a short BBN/Slate extension language program. You can write and publish these custom functions and then include them in the profile file. These are marked as "SEL function."

Sometimes, the key sequences you emulate may override the default BBN/Slate key sequences. In these cases, you should decide whether to provide different key sequences for accessing the BBN/Slate function.

Most of the BBN/Slate functions can be found simply by browsing through the text functions in Appendix B of this manual. You can also select **BBN-Invoke Function** from the main menu, or enter execute-command-dialog (the default binding is **Esc-x**) and use the apropos-dialog command to examine possible synonyms for some of the emacs function names. (This technique is described in detail in Chapter 3.)

Table 1-2 **Comparing BBN/Slate and emacs functions**

BBN/Slate function	Key Sequence	Emacs function
text-set-mark	C-@	set-mark-command
text-start-of-line *	C-a	beginning-of-line
text-backward-character *	C-b	backward-char
N.A.	C-c	mode-specific-command-prefix
text-delete-forward *	C-d	delete-char
text-end-of-line *	C-e	end-of-line
text-forward-character *	C-f	forward-char
text-kill-line *	C-k	kill-line
text-down-line *	C-n	next-line
text-open-line *	C-o	open-line
text-up-line *	C-p	previous-line
text-search-reverse-incrementally *	C-r	isearch-backward
text-search-forward-incrementally *	C-s	isearch-forward
text-transpose-characters *	C-t	transpose-chars
scroll-up	C-v	scroll-up
text-cut-region *	C-w	kill-region
text-paste *	C-y	yank
text-delete-back *	DEL	delete-backward-char
change-to-buffer *	C-x C-b	list-buffers
save-buffers-and-exit	C-x C-c	save-buffers-kill-emacs
visit-file-dialog *	C-x C-f	find-file
SEL function	C-x C-o	delete-blank-lines
save-current-buffer *	C-x C-s	save-buffer
write-current-buffer *	C-x C-w	write-file
delete-current-pane	C-x 0	delete-window
delete-other-panes *	C-x 1	delete-other-windows
split-pane-down	C-x 2	split-window-vertically
split-pane-across	C-x 5	split-window-horizontally
scroll-left	C-x <	scroll-left
scroll-right	C-x >	scroll-right

(continued)

Table 1-2 **Comparing BBN/Slate and emacs functions** *(continued)*

BBN/Slate function	Key Sequence	Emacs function
page-up	C-x [	backward-page
page-down	C-x]	forward-page
change-to-buffer	C-x b	switch-to-buffer
insert-file-dialog	C-x i	insert-file-dialog
next-pane	C-x o	other-window
text-start-of-buffer *	ESC <	beginning-of-buffer
text-end-of-buffer *	ESC >	end-of-buffer
text-backward-paragraph	ESC [	backward-paragraph
text-forward-paragraph	ESC]	forward-paragraph
text-end-of-sentence	ESC a	backward-sentence
text-backward-word *	ESC b	backward-word
text-delete-forward-word *	ESC d	kill-word
text-forward-word *	ESC f	forward-word
N.A.	ESC g	fill-region
execute-command-dialog *	ESC x	execute-command-dialog
text-paste	ESC y	yank-pop

The next step is to enter bind commands in your profile file that correspond to columns one and two in Table 1-2. Functions marked with an asterisk need not be bound, since they already use a common key binding. Functions denoted as not applicable are ignored. Functions marked as possible BBN/Slate extension language functions are not included here, but could easily be developed (see Chapter 3 for details).

The remaining functions — those for which a key binding needs to be assigned — are shown in Table 1-3. This table illustrates the actual `bind` commands you would place in a profile file.

Table 1-3 `Emacs` **key bindings in a BBN/Slate profile file**

```
##################
#BBN/Slate emulation of common emacs key sequences
##################
bind( text-keymap, "^@",   text-setmark);
bind( text-keymap, "^v",   scroll-up);
bind( text-keymap, "^X^C", save-buffers-and-exit);
bind( text-keymap, "^X0",  delete-current-pane);
bind( text-keymap, "^X2",  split-pane-down);
bind( text-keymap, "^X5",  split-pane-across);
bind( text-keymap, "^X<",  scroll-left);
bind( text-keymap, "^X>",  scroll-right);
bind( text-keymap, "^X[",  backward-page);
bind( text-keymap, "^X]",  forward-page);
bind( text-keymap, "^Xb",  change-to-buffer);
bind( text-keymap, "^Xi",  insert-file-dialog,);
bind( text-keymap, "^Xo",  next-pane);
bind( text-keymap, "Esc-[",text-backward-paragraph);
bind( text-keymap, "Esc-]",text-forward-paragraph);
bind( text-keymap, "Esc-a",text-end-of-sentence);
bind( text-keymap, "Esc-x",execute-command-dialog);
bind( text-keymap, "Esc-y",text-paste);
```

Finally, you should test each of the bindings to be sure you have entered the key binding correctly and that you have employed the correct BBN/Slate function. Once you are satisfied, you can place the bindings in the public profile file (if all users at your site want to use them) or in the appropriate private profile files.

Section 1.4 describes how to set up mail aliases when ending electronic mail.

1.4 Creating mail aliases

Mail aliases allow you to create a shorthand notation by which you can refer to people or groups of people when sending electronic mail. Mail aliases make it easier to send mail to people whose mail addresses are difficult to remember or type. They are also convenient when you often send mail to the same group of people; you can use a single word to represent the entire group, rather than having to type in each person's address every time you send mail.

You create mail aliases by including *alias commands* in your profile file. The syntax of `alias` commands is:

```
alias("name", "address-or-alias, address-or-alias...");
```

where *address-or-alias* is the mail address of the recipient or another alias which represents the mail address of the recipient.

The semicolon is optional, but it helps to mark the end of the alias command. The *name* is the word you want to use to refer to the *addresses* listed in the alias. For example, if you had an alias command like this:

```
alias("joe", "jsmith23b@shire.bbn.com");
```

you could send a message whose **To** field said simply `joe`, and BBN/Slate would automatically send it to `jsmith23b@shire.bbn.com`. Similarly, you could use an alias like this:

```
alias("cohorts", "paul, ellie, robin@xyz.edu");
```

to send mail to a whole list of people, just by typing "cohorts" in the **To** field of your message. The addresses in an alias can be aliases themselves, so

```
alias("friends", "cohorts, joe, sam@xyzzy.com");
```

makes `friends` equivalent to all of the people listed in both the `joe` and `cohorts` aliases, plus `sam` at `xyzzy`.

2 Configurable Menus

BBN/Slate provides *configurable menus* as a convenient and powerful mechanism for customizing BBN/Slate menus. You can use configurable menus to:

- add menu entries for existing BBN/Slate functions that are not currently displayed as system menu items (Section 2.1)
- rename and/or reorganize existing menu entries (Section 2.2)
- create a menu entry that invokes a new BBN/Slate extension language function (Section 2.3)

Just as key bindings allow BBN/Slate users to customize their keyboard environment, configurable menus allow BBN/Slate users to customize their menu environment.

Changing the existing BBN/Slate menus is not difficult, but you should exercise judgment when, for example, deleting default menu entries. Note that changing the menus can alter the appearance and functionality of the software from what is described in the BBN/Slate documentation.

2.1 Menu files

System menus

The default BBN/Slate menus, called *system menus,* are files located in `/usr/slate/menus` (note that your site may have installed BBN/Slate in a different directory than `/usr/slate`). There are two subdirectories under `menus`. The `full` subdirectory contains the complete, full system menus. The `quick` subdirectory contains the shorter QuickStart menus. This chapter presents examples using the system menu files located in the `full` subdirectory. (You can handle the system menu files located in the `quick` subdirectory in identical fashion.)

The current system menu files are:

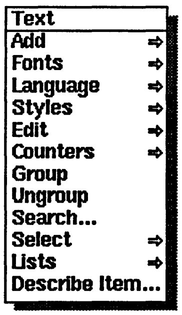
```
bbnicon.menu       image.menu
editdoc.menu       speech.menu
enclosure.menu     spreadsheet.menu
graphics.menu      text.menu
headers.menu
```

BBN/Slate menu files are plain text files. Note that system menu files are organized not around any particular menu structure (which is arbitrary), but around the BBN/Slate media types (text, graphics, enclosures, and so on). Each system menu includes the suffix `.menu` in the filename.

When configuring menus, you make a copy of one or more of these system menus in a local directory, keeping the same filename. You then edit the copy to change the existing menus (see Section 2.3).

File structure and format

A menu file has a regular and defined organization and structure that mirrors the functionality of its actual menu. Consider, for example, the **TEXT** menu and the initial portion of the `text.menu` file:



```
MENU Text
    TITLE Text
    MENUNAME Text/Add
    MENUNAME Text/Fonts
    MENUNAME Text/Language
    MENUNAME Text/Styles
    MENUNAME Text/Edit
    MENUNAME Text/Counters
    ITEM Group
        HELP "Place the selected region in a single
        list or list item. (^Xg)"
        INVOKES text-group
        GENERATED-BY text-group-pertinent
    ENDITEM

    ITEM Ungroup
    HELP "Break the selected region out of a
        list or list item. (^Xu)"
    INVOKES text-ungroup
    GENERATED-BY text-ungroup-pertinent
    ENDITEM
    ITEM Search...
    HELP "Search for some string or replace
        some string. (M-S)"
    INVOKES text-search-forward-dialog
    ENDITEM
    MENUNAME Text/Select
    MENUNAME Text/Lists
    ITEM Describe Item...
    HELP "Provide information about the paragraph
         which contains the cursor. (^Xw)"
    INVOKES text-describe-object
    ENDITEM
ENDMENU
```

This menu contains the two categories of menu entries: *submenus* and *simple commands.*

The first six entries of the menu (from **Add** through **Counters**) are *submenu* entries that lead to further menu choices. These are identified in the menu file as MENUNAME. Subsequent portions of the menu file describe each submenu, using their own MENU entries, but for this menu level, including just the MENUNAME label is sufficient.

The next three menu entries (from **Group** through **Search...**) are *simple command* entries that are used to directly invoke commands or functions. These are identified in the menu file as ITEM.

Note that each ITEM entry includes additional lines, or *fields*. For example, the HELP field provides a help message to display when a user presses **MIDDLE hold** for the current menu item. Section 2.2 describes all of the possible menu fields.

2.2 Menu file syntax

A menu file should contain only the fields necessary to its particular definition; all fields are optional. The possible fields and their syntax are:

```
MENU command (menu-name)
    TITLE              name
    HELP               string
    INVOKES            name
    FONT               name
    GROUPS             (name1, name2, ... namen)
    GENERATED-BY       name
    DEFAULT-SELECTION  index of default
                       selection, starting at 0
    command description
ENDMENU
```

These fields are described as follows:

MENU command (menu-name)

Menu names must be unique over all the BBN/Slate menus. If the command name is unique, then menu-name may be omitted, and name of the menu is considered the same as command. For example:

```
MENU Add (Text/Add)
MENU Text
```

The name of the first menu is Text/Add. The name of the second menu is Text.

The command is what appears in the menu. Menu names cannot include parentheses, but can include spaces.

There are two cases where you must provide menu-name:

1. When you want to use the shorthand "MENUNAME abc" and describe it later with the MENU ... ENDMENU form.
2. When the actual menu (not just the command) is used in two or more places in the menu hierarchy.

   For example, suppose you have a list of the text formats in a menu, and you want that menu to be invoked both by the TEXT/ADD menu item and by the TEXT/STYLES/GLOBAL CHANGE menu item. You describe the menu once, and then use its name to refer to it thereafter.

TITLE

The title is the name that appears at the top of the menu. For example:

```
TITLE Graphics
```

HELP

The help field is a string, enclosed in double quotation marks. This message is displayed whenever **MIDDLE hold** is pressed. Any carriage returns and leading or trailing spaces in the message are stripped. For example:

```
HELP "Place the selected region in a single list or
      list item. (^Xu)"
```

Note that the existing menu help messages sometimes include information about default key bindings. If these bindings change (see Chapter 1), you should also change the appropriate help messages.

`INVOKES`	If you include this field, the named BBN/Slate function is invoked when the menu item is chosen. The function must be published (i.e., user-callable). For example:

```
INVOKES text-group
```

`FONT`	To display a menu entry in a different font, include this field. You can specify any font in the `slate/fonts` subdirectory whose name includes a `.font` suffix (but do not include this suffix in the name). For example, to use 18-point Helvetica bold, specify:

```
FONT helvetica18b
```

`GROUPS`	This field is included in default BBN/Slate menus to impose constraints on the system menus by dimming (making unselectable) various groups of commands according to the circumstance. You should not attempt to alter existing `GROUP` fields or to apply new instances of them to menus that you reconfigure.
`GENERATED-BY`	Include this field if you need to include a function to generate a context-specific menu entry. For example, the submenu for **Styles-Local Change** varies, depending on the current active element (paragraph, example, etc.). By including:

```
GENERATED-BY text-generate-local-edit-menu
```

in the menu description, you can create an appropriate submenu entry. The function must be published (i.e., user-callable).

`DEFAULT-SELECTION` This field is the index (starting with 0) to the default selection in that menu. The default selection is highlighted when you first display the menu. The default selection is chosen if you select its parent menu command without actually displaying the submenu.

`Command description` The `command description` consists of one or more lines. Depending on the circumstances, they can be organized in any of four forms. The first form is for entries that are not submenus, and the next three forms are for entries that are submenus:

- **ITEM.** If the menu entry directly invokes a command or function, use a simple command description (`ITEM...ENDITEM`).
- **MENU.** If the menu entry involves standard menu actions, use another `MENU` entry. In other words, you nest one `MENU` entry within another. This command description consists of its own `MENU...ENDMENU` section. There is no limit to the number of levels you may choose to nest, although you may find more than three or four levels of menus unwieldy. However, you can simplify the organization of your menu file by using `MENUNAME` (see below) entries in place of a full `MENU` definition, and then providing the actual full `MENU` definition later.
- **MENUNAME.** If the menu entry leads directly to another submenu, you can use a submenu command (`MENUNAME`). `MENUNAME` is simply a shorthand convention that serves as a pointer to the full `MENU...ENDMENU` definition later in the menu file.
- **MENUITEM.** Use a `MENUITEM` entry if the field refers to a command that has a submenu and you want to control the `FONT` or `GENERATED-BY` fields.

These four types of command descriptions are described in the following sections.

2.2.1 Item

An ITEM (simple command description) contains many of the same fields as the basic MENU description:

```
ITEM name
    HELP         string
    INVOKES      name
    FONT         name
    GROUPS       (name1, name2, namen)
    GENERATED-BY name
ENDITEM
```

The field descriptions are identical to those listed for MENU in Section 2.2.

2.2.2 Menu

You can nest one MENU...ENDMENU section within another. You may find it more convenient and easier to read your menu files by using the MENUNAME form.

2.2.3 Menuname

A MENUNAME entry has the form:

```
MENUNAME menu-name
```

Using MENUNAME, you can reduce the nesting of menu definitions, making menu files easier to read. For example, the following menu definition:

```
MENU Text
    MENU Add (Text/Add)
        ...
        MENU Face (Text/Fonts/Face)
        ...
        ENDMENU
    ENDMENU

    MENU Fonts (Text/Fonts)
        ...
    ENDMENU
ENDMENU
```

can be organized in a more linear fashion by using MENUNAME statements in place of the original MENU descriptions, followed by each MENU description:

```
MENU Text
    MENUNAME Text/Add
    MENUNAME Text/Fonts
ENDMENU

MENU Add (Text/Add)
...
ENDMENU

MENU Fonts (Text/Fonts)
...
ENDMENU

MENU Face (Text/Fonts/Face)
...
ENDMENU
```

2.2.4 Menuitem

```
MENUITEM menu-name
    FONT         name
    GROUPS       (name, name, name)
    GENERATED-BY name
ENDMENU
```

This form is used if the menu fields refer to a command that has a submenu, and you want to control the `FONT` or `GENERATED-BY` fields.

The `menu-name` is a menu name, and there must be a description (of the form `MENU...ENDMENU`) corresponding to that name somewhere else in the file.

The three fields (`FONT`, `GROUPS`, and `GENERATED-BY`) describe the command that invokes the submenu; the fields in the corresponding `MENU...ENDMENU` section describe the submenu. For example, specifying the `FONT` field in a `MENUITEM` alters the font just for `menu-name`. Specifying the `FONT` field in a `MENU...ENDMENU` section alters the font for the entire submenu.

2.3 Tutorial: Configuring a new menu file

This section describes the steps you take to configure BBN/Slate menus, including examples of changing existing menus and adding new functionality.

Configuring menus involves altering the default BBN/Slate system menus. Generally, you do this *outside* of BBN/Slate, using a text editor from the operating-system level. (You also can edit the files from within Slate, reading and writing them as plain text files.) Changes to menus do not take effect until you restart `editdoc`.

This initial example reconfigures some of the features of the main BBN/Slate system menu file, `editdoc.menu`. Begin as follows:

Step 1

Using a workstation window outside of BBN/Slate, create a subdirectory for menu functions (if you have not already done so). For example:

```
mkdir /home/smith/Slate/Menus
```

While it is not necessary to provide a separate subdirectory for this purpose, it makes it easier to locate menu functions, and it keeps your regular BBN/Slate directories cleaner.

Step 2

Copy the `editdoc.menu` system menu file from the BBN/Slate `Menus` subdirectory. For example, if the system menus are in `/usr/slate/menus/full`, you could enter:

```
cd /home/smith/Slate/Menus
cp /usr/slate/menus/full/editdoc.menu .
```

Step 3

Add the path for the new menu subdirectory to your profile file (`.slate_editor.init`). For example:

```
menu-directory = "/home/smith/Slate/Menus";
```

Step 4

From the new menu subdirectory, start up your text editor. For example:

```
emacs editdoc.menu
```

You can now make several changes to the menu file.

2.3.1 Changing menu order

You are now ready to edit a local copy of the `editdoc.menu` file.

First, you can alter the current order of the **Document** main menu entries:

```
Set Language Font...
Find Object
Check Spelling
```

to alphabetical order:

```
Check Spelling
Find Object
Set Language Font...
```

Step 1

Locate the **Document** menu lines in the `editdoc.menu` file by searching for:

```
MENU Document
```

Step 2

Study the organization of this section. Notice that the three menu entries appear consecutively. A summary of this section is shown below (in detail only to the first level of indent to save space), with the menu entries in bold:

```
MENU Document (Editor/Document)
    HELP "Assorted commands ...
    ITEM Set Language/Font...
         ...
    ENDITEM
    MENU Find Object (Editor/Document/Find Object)
         ...
    ENDMENU
    MENU Check Spelling (Editor/Document/Check Spelling)
         ...
    ENDMENU
ENDMENU
```

Carefully move each block of lines that comprise each menu entry into their new order so that they now appear as follows:

```
MENU Document (Editor/Document)
    HELP "Assorted commands ...
    MENU Check Spelling (Editor/Document/Check Spelling)
         ...
    ENDMENU
    MENU Find Object (Editor/Document/Find Object)
         ...
    ENDMENU
    ITEM Set Language/Font...
         ...
    ENDITEM
ENDMENU
```

Step 3

Save the file in the local directory (be sure to keep the file named `editdoc.menu`). Then, from the operating-system level, test the change by starting a new Document Editor window with the command:

```
editdoc
```

Step 4

Pull down the **Document** menu. You should now see the menu entries in their new, alphabetical order. Although there is a new order, the functionality of each menu item is still the same.

When you are done testing, you can exit from the new Document Editor window.

2.3.2 Adding new functionality

One of the most powerful aspects of BBN/Slate customization is the ability to make custom functions available to all users. One of the most convenient ways to do this is by using customized menus.

There are two basic resources available for adding functionality to BBN/Slate. First, you can draw upon existing BBN/Slate functions that are not currently part of the menu system; Appendix B lists hundreds of functions. Second, you can create your own functions using the BBN/Slate extension language (see Chapter 3), and then reconfigure your BBN/Slate menus for easy access to your new functions. As long as the function you call does not require arguments, you can invoke it from a menu file. (Note that the BBN/Slate extension language gives your function access to system variables and the ability to employ arguments when calling other functions.)

Adding BBN/Slate functions

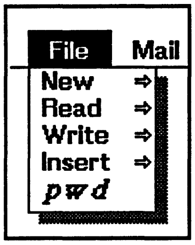
As an initial example, you can add the ability to invoke the BBN/Slate function `pwd-in-dialog` from the main **File** menu. This function displays the current working directory in a pop-up message window.

Step 1

As described in the previous example, edit a local copy of the `editdoc.menu` file. Search for the string `MENU File`. Add the portion shown in bold so that the complete section reads as follows:

```
MENU File (Editor/File)
    HELP "Create, load, and store documents."
    MENUNAME Editor/File/New
    MENUNAME Editor/File/Read
    MENUNAME Editor/File/Write
    MENUNAME Editor/File/Insert
    ITEM pwd
        HELP     "Print Working Directory in
                 pop-up dialog box."
        FONT     Helvetica14bi
        INVOKES  pwd-in-dialog
    ENDITEM
ENDMENU
```

The new menu item specifies a font of 14-point Helvetica bold italic. This makes the menu entry distinct from the others in the **File** menu:



Step 2

Save the file, then invoke the command `editdoc` again to test the change. When you select **pwd** from the menu, you should see your current working directory displayed in a dialog box.

Adding extension language functions. You can also add a new extension language function that you wrote yourself. This is really no different than adding existing BBN/Slate functions such as `pwd-in-dialog`.

Chapter 3 provides complete details on writing and testing your own BBN/Slate extension language functions. Also, make sure that the function is `public` and you have included it in a profile file (for example, using `autoload`).

Using `make-slides` (a program presented in Chapter 3), an example is given here of how to add menu support for an extension language function. This program automatically creates separate viewgraph files from an existing BBN/Slate document. Assuming that you have created the file `make_slides.sel`, entered the `make-slides` extension language functions, and tested the program by invoking it explicitly with the `execute-command-dialog` function, you can now proceed with the steps involved in adding menu support.

Step 1

You must define the program by adding it with an `autoload` statement to your `.slate_editor.init` file. If you plan to invoke your function only from a menu, then you can omit any key binding from the `autoload` statement:

```
autoload (make-slides, make_slides.sel);
```

If you want to be able to invoke the function from the keyboard as well, you can include a key binding:

```
autoload (make-slides, make_slides.sel,
    text-keymap, "Esc-'");
```

Step 2

Again, edit a local copy of the editdoc.menu file. Add this function to the end of the realphabetized entries from the **Document** main menu (see Section 2.3.1), as shown in bold:

```
ITEM Set Language/Font...
    HELP "Change the default ...
    INVOKES set-document-font-dialog
ENDITEM
ITEM Make Slides
HELP "Create viewgraph files from the current file."
INVOKES make-slides
ENDITEM
```

Step 3

Save the file, then invoke the command editdoc again to test the change. When you select **Make Slides** from the menu, the make-slides extension language function runs, you are queried for information, and the file is processed into viewgraphs.

2.3.3 Public accessibility

When you create menu files that you want to be in effect for all BBN/Slate users at your site, you may find the following strategy helpful.

1. Keep your site-specific menu files in a special site-wide location. Placing the files in a subdirectory of your BBN/Slate software is not a good idea, in case you later delete the area when installing a new version of BBN/Slate.

2. Add a `menu-directory` entry to a site-wide BBN/Slate profile file `slate_editor.init`. If this file does not yet exist, you can easily create one. If you installed BBN/Slate in `/usr/slate`, for example, this file should be located in `/usr/slate/lib`. This strategy avoids the extra work of making sure that the `menu-directory` specification is in every user's profile file.

   There is an additional benefit to doing this. Having a `slate_editor.init` file specify a site-wide reconfigured menu allows individual users to create their own, private menu configurations that can further modify the site-wide configuration.

3 The BBN/Slate Extension Language

3.1 Overview

The BBN/Slate Extension Language (SEL) is a tool that extends the capabilities of BBN/Slate software and enhances your BBN/Slate environment. You can use it to:

- build new, custom functions that accomplish specific tasks for your editing environment and, if desired, access them via a reconfigured menu
- control actions in enclosures (including exchanges with external database management software, and interactive dialogues to allow users to load external files)
- define a common BBN/Slate environment for all users in your work group
- tailor BBN/Slate more closely to editing environments with which you are already comfortable (for example, you can define the same keyboard equivalents that you already use in a text editor, such as `emacs` or `vi`, for cursor control, file saving, and window control)
- access BBN/Slate functions not available through default menus and key bindings

SEL can help you work with all media types, and works within the BBN/Slate Document Editor program `editdoc`.

To summarize the purpose of SEL succinctly, it offers *extensibility* and *customization.*

Who can use SEL?

System administrators can use SEL to create a common environment for all BBN/Slate users at their site. Individual users can use SEL to redefine the common environment, or to create additional functionality for their own use.

3.1.1 Using SEL

To develop new functions in the Slate Extension Language, you:

1. create a file containing extension language commands, functions, and variable definitions
2. invoke a command that reads and executes the commands in that file

Once the file of extension language commands has been read by `editdoc`, any function defined in that file can be directly invoked. The file can be read through explicit command, or read automatically at startup by placing a command to read the file in the user's `.slate_editor.init` file.

3.1.2 Accessing functions

The BBN/Slate software comes with hundreds of built-in functions for handling compound documents. While many of these functions are immediately available to all users, many others are not assigned to the default menus or key bindings that come with the system, as in the following illustration.

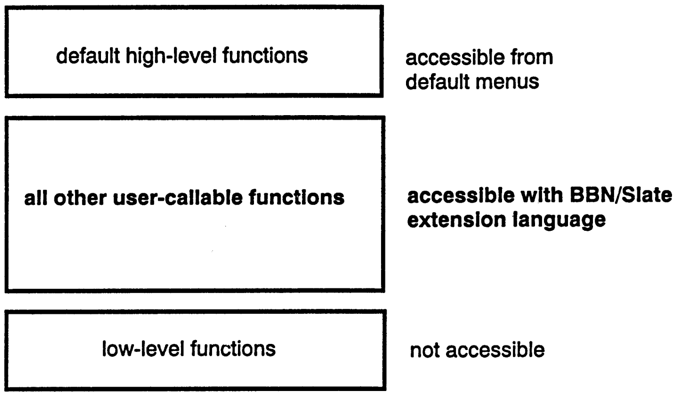

default high-level functions
accessible from default menus
all other user-callable functions
accessible with BBN/Slate extension language
low-level functions
not accessible

Figure 3-1 **Accessibility of functions in BBN/Slate**

You commonly access BBN/Slate functions in the following ways:

Menus

Choosing a pull-down menu item automatically invokes a corresponding function or series of functions. For example, choosing the main-menu function **`keycaps`** in the Document Editor invokes the function **`display-keyboard`**. (Some menu actions involve low-level functions, however.) The use of menus is explained fully in the *Reference Manual.*

Bindings

A key binding can be used to invoke functions. For example, the up-arrow key is bound to the **text-up-line** function. Sections 3.2, 3.4, and 3.5 present key bindings in more detail.

Explicit invocation

You can explicitly invoke functions using the **execute-command-dialog** command. This is itself a function that pops up a window where you enter the function name and then **click** the **OK** button to invoke it. (You may do this only to access functions that do not require arguments. You can allow users access to functions that require arguments by building SEL functions, where the functions either supply the arguments explicitly, or the argument values are supplied by querying the user.) Section 3.4 describes explicit invocation.

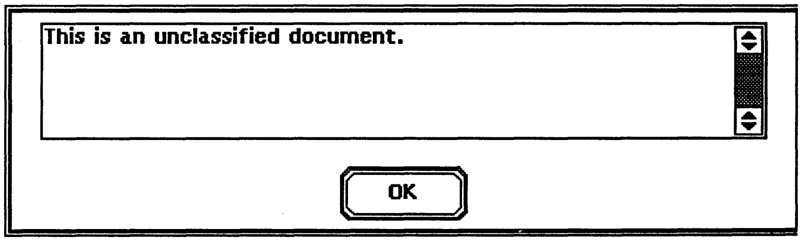
3.1.3 Power and flexibility

One of the principal purposes of the BBN/Slate extension language is to help you take advantage of many of the powerful functions already developed for the BBN/Slate software. You can then use the extension language to combine these functions in unique ways that meet the specialized needs of your site.

SEL keeps "programming overhead" to a minimum by allowing you to perform complicated functions with single statements. For example, in many programming environments, displaying a message in a pop-up dialog box could involve writing a large number of statements; in the BBN/Slate extension language, you can perform this with one statement, such as the following:

```
display-message("This is an unclassified document.");
```

You can then define this statement as part of a function that, when invoked from within the Document Editor, creates the following pop-up window:


This is an unclassified document.

The BBN/Slate extension language also provides a full range of programming constructs. For example, the following commands:

```
local search-string, search-region;
search-string = read-string-dialog(
   "Keyword or phrase to search for.",
   "Word or phrase: ");
search-region = create-region(beginning-of-buffer(),
   end-of-buffer());
if (text-search(search-string, search-region, 1))
   {
   text-line-to-top()
       text-select-current-word()
   }
```

perform these tasks:

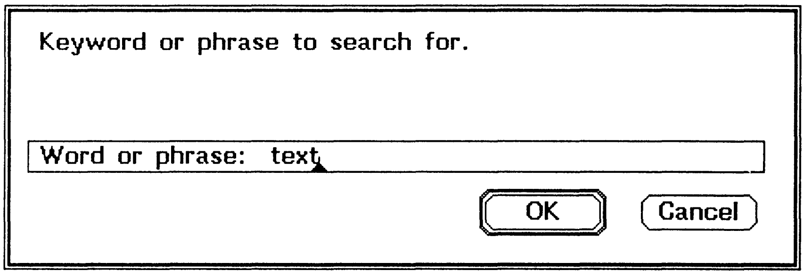
- prompt the user to enter a search string
- search the document for the specified string
- scroll the paragraph containing that string to the top of the display area if the string is found
- highlight the word

The vehicle for applying extension language code such as this is the *function definition.* By adding the lines shown in bold, these same statements can stand alone as a complete function definition:

```
define public void search-word()
"Search for a word, scroll to top, and highlight it."
{
local search-string, search-region;
search-string = read-string-dialog(
    "Keyword or phrase to search for.",
    "Word or phrase: ");
search-region = create-region(beginning-of-buffer(),
    end-of-buffer());
if (text-search(search-string, search-region, 1))
    {
    text-line-to-top();
    text-select-current-word();
    }
}
```

There are several ways to make this function active (see Section 3.3). Once this is done, you can invoke the new function `search-word`, test it, and subsequently put it to work as a new BBN/Slate feature for yourself or for your site. You can also bind it to a specific key such as the R1 key (see Section 3.3 for a complete discussion of key binding).

Pressing the R1 function key invokes the newly defined function `search-word`. It creates a pop-up window that prompts you to enter a search string:


Keyword or phrase to search for.

Word or phrase: text

If it finds the string, it then scrolls the paragraph containing the string to the top of the window and highlights the word. For example:

```
The departmental survey on text-retrieval techniques
is reviewed in this report.
```

The remainder of this manual explains in more detail how you can use and define functions in SEL. While the introductory examples presented here are simple in nature, it is possible to build far more sophisticated programs. Section 3.5 presents some advanced SEL examples that you may want to survey, such as a mail-merge program.

Because SEL is a programming language, the next section introduces you to the syntax of the language so that you can begin to familiarize yourself with its conventions.

3.2 Syntax and semantics

This section specifies the syntax of the BBN/Slate extension language. This information is useful to anyone who plans to use the BBN/Slate extension language. In addition, you may find the examples presented in Section 3.5 to be an aid in learning SEL.

3.2.1 Integer constants

Integer constants are represented by a string of unsigned digits.

3.2.2 Real constants

Real constants are represented by a string of unsigned digits including a decimal point.

3.2.3 String constants

String constants are a sequence of characters bounded by double quotes. To include a double quote in a string, precede it with a backslash character. For special characters, use the character's octal representation preceded by a backslash; for example, to include the special ASCII character **^H**, use the representation \010.

3.2.4 Variable names

Variable names are a string of uppercase and lowercase letters or numbers, and may also contain any of the following characters:

```
- . @ _
```

The dash, period, and numbers cannot be used as the first character of a variable name. Variable names are case dependent; 'xxx' and 'XXX' are different variables.

3.2.5 Reserved words

The following are reserved words in the extension language. These words are case independent (they may be in uppercase, lowercase, or mixed case).

```
autoload
if
define
else
global
local
null
private
public
return
undefine
while
```

Note that the SEL reserved word `null` is analogous to `null` in the C language. You can compare `null` with a pointer or string, assign it to a pointer or string, and so on.

3.2.6 Comments

Text from a pound sign (#) to the end of a line is considered a comment and is ignored.

3.2.7 Variable types and usage

The following types are defined:

```
buffer      object
char        pointer
document    real
int         region
keymap      string
mark        void
```

Variables must be assigned a value (by appearing on the left-hand side of an assignment statement) before they can be used on the right-hand side of an assignment statement, in an expression, or as an argument to a function.

Variable types must match when:

- used in expressions and assignment statements
- passed as arguments to `editdoc` or user-defined functions

The only exceptions are that pointers and integers are considered to match and that strings are converted to and from integers as needed.

You do not declare a variable's type. This is determined when the variable is first assigned a value.

3.2.8 Variable scoping

Variables are either global, file local, or function local. All user-defined variables must be declared before being used. Variables defined in `editdoc` (see Section 3.3.3 and Appendix A) are declared at initialization, before the user's profile file is read. You do not need to declare these types of variables before using or setting them.

Global variables are declared in a `global` statement; *file local* variables are declared in a `local` statement:

```
global a, b, c
local  d, e, f
```

Both global and file local variable declarations must appear outside a function. For example:

```
local  d, e, f
define public void test()
{
}
```

Function local variables are also declared in a `local` statement that appears after the definition of a function, but before the body of the function (see Section 3.4 for more information on the syntax of function definitions):

```
define public void test()
{
    local g, h, i
}
```

When a variable is encountered, the extension language interpreter first looks for its definition as a function local variable (defined in the function currently being executed) or function parameter, then as a file local variable, and finally as a global variable.

Purpose and use of global variables

Once they are declared, global variables *permanently* reside within your BBN/Slate software. Because of this, if you need to pass information between functions scattered over several files, you may find it more practical to pass information via a calling function's arguments. Global variables are most appropriate for selected, general-purpose declarations. For example, you may want to declare:

```
global TRUE, FALSE;
TRUE = 1;
FALSE = 0;
```

so that you can use expressions such as:

```
while (TRUE)
```

3.2.9 Expressions

The syntax of an expression is:

```
   expression binary-operator expression
or
   ( expression )
or
   unary-operator expression
```

The operators are:

```
equality (==)
inequality (!=)
greater-than (>)
less-than (<)
greater-than-or-equal-to (>=)
less-than-or-equal-to (<=)
multiplication (*)
division (/)
addition (+)
subtraction (-)
logical or (||)
logical and (&&)
logical negation (!)
unary-negation (-)
assignment (=)
```

The precedence of the operators is the same as in the C language:

Precedence (descending order)

```
!   - (unary)
*   /
+   -
<   <=   >   >=
==  !=
&&
||
=
```

The entire expression is always evaluated. For example, in the statement:

```
   while (a == 1 && b == 2) c = c + 1;
```

both the values of a and b are tested, even if a is not 1.

All of the operators may be applied to all of the variable types with the following exceptions:

- unary operations are not allowed on strings
- arithmetic operations and logical operations are not allowed on two strings (however, string-to-integer conversions are done for expressions in which one argument is a string and one argument is an integer)
- arithmetic and logical operations are not allowed on two marks
- arithmetic operations are not allowed on two regions
- unary subtraction is supported only for reals and integers

These exceptions are summarized in Table 3-1.

Table 3-1 **Application of operators to variable types**

Description	Operator	String	Strings	Marks	Regions
Arithmetic operators					
Binary	+ - * /		No	No	No
*Unary	-	No	No	No	No
Relational operators	< <= > >=				
Equality operators	== !=				
Logical operators					
Connectors	|| &&		No	No	
Negation	!		No	No	

* *Note:* Unary negation is supported only for reals and integers.

3.2.10 Statements

The following types of statements are defined in the extension language:

Statement Type	Discussion
`if, while, assignment, return, function call, function definition`	Section 3.2
`undefine`	Section 3.2.17
`global`	Section 3.2.8
`local`	Section 3.2.8
`autoload`	Section 3.4.8

Multiple statements can be enclosed in braces; they are then treated as a single statement. Statements may be optionally terminated with a semicolon.

3.2.11 The `if` statement

The syntax of an `if` statement is:

```
    if ( expression ) statement
or
    if ( expression ) statement else statement
```

The parenthesized expression is evaluated. If it is non-zero, the statement following it is executed. If it is zero and there is an `else` statement, then the `else` statement is executed. An `else` statement is always assumed to match the innermost `if` statement possible.

3.2.12 The `while` statement

The syntax of a `while` statement is:

```
while ( expression ) statement
```

The parenthesized expression is evaluated. If it is non-zero, the statement following it is executed. These two steps are repeated until the parenthesized expression is zero; then the statement following the `while` statement is executed.

3.2.13 Assignment statement

The syntax of an assignment statement is:

```
variable = expression
```

The expression is assigned to the variable.

3.2.14 The `return` statement

The syntax of a `return` statement is:

```
  return ( expression )
or
  return
```

A `return` statement must be inside a function. It causes execution of the function to be terminated. The expression in parentheses is the result of the function. It must match the type-name in the function definition (see Section 3.2.7).

3.2.15 Function call

The syntax of a function call is:

```
function-name ( parameters )
```

where *parameters* is a comma-separated list of expressions. If there are no parameters, the parentheses must still be included. Each actual parameter is evaluated and assigned to the corresponding formal parameter when the function is called, i.e., arguments are passed by value. If a parameter is changed within the called function, its value outside that function is not changed.

3.2.16 Indirect function call

The syntax of an indirect function call is:

```
invoke symbol parameters
```

where *symbol* is a variable that contains a function name, and *parameters* is a comma-separated list of expressions (if any).

3.2.17 Function definitions

The syntax of a function definition is as follows:

```
define function-type type-name function-name
    (variable-list) "help string"
{
    local local-variable-list
    .
    .
    .
    statements
    .
    .
    .
}
```

The *function-type* is either `private` or `public`. The default is `public`. You should declare a function to be `private` if it is just a "helper function," one not intended to be used by itself or invoked directly by the user. Only `public` functions can be described via **apropos-dialog** or **describe-command** and executed via **execute-command-dialog** (see Section 3.4).

The *type-name* can be any of the valid types (see section 3.2.7). If the type-name is `void`, then it is assumed that the function does not return a value. If the type-name is not `void`, then a value must be returned by a `return` statement within the function body (see section 3.2.14).

The *variable-list* is a comma-separated list of variable names. It is optional, but the parentheses must be included. When the function is called, each variable in the list is assigned the value of the corresponding actual parameter in the function call (see Section 3.2.10).

The optional *help string* describes the function. The help string must be enclosed in double quotes; to include a double quote in the help string, precede it with a backslash character. This help string is printed by the `editdoc` function, **describe-command** (see Section 3.3.4).

The *local-variable-list* is a comma-separated list of variable names, and is optional. The `local` statement defines variables that are local to the function (see Section 3.2.8).

The body of the function is enclosed in braces and contains zero or more statements.

User-defined functions can be redefined. When a function definition is read, if there is an existing user-defined function with the same name, the user is queried as to whether to redefine the function. If the user replies affirmatively, the function is redefined; if the user replies negatively, the new definition is ignored. The user query can be bypassed in one of two ways:

- Place the following line in your profile file:

  ```
  redefine-silently = 1;
  ```

- Explicitly undefine the function, using the extension language `undefine` command.

The `undefine` command takes a comma-separated list of names of functions to undefine. For example:

```
undefine test, test2
define public void test()
# new definition of test function
{
}

define public void test2()
# new definition of test2 function
{
}
```

3.3 Functional description

This section describes the functional characteristics of the BBN/Slate extension language.

3.3.1 The role of the profile file

The extension language coordinates closely with your profile file. Chapter 1 explains profile files in detail: what they can contain, how you might organize them, and how profile file initialization works. It also covers the use of public and private profile files, and how you can use private profile files to redefine default system characteristics (for example, through the use of global preference variables).

Several kinds of profile file content relate directly to the extension language:

- library search paths
- extension language definitions
- `autoload` commands

Search paths

You must include the variable `library-search-path` in your profile file to specify those directories where you store files of BBN/Slate extension language definitions. You can specify more than one directory, separating each with a colon. The string of directory names must be surrounded in double quote marks. For example:

```
library-search-path = "/home/smith/Slate/sel:/home/sel";
```

Extension language definitions

You can place complete extension language definitions directly in your profile file. However, you should restrict this practice to minimize `editdoc` startup time. It is preferable to use `autoload` commands.

The `autoload` *command*

The `autoload` command restrains reading function definitions from files until the functions are actually invoked. Thus, you can use `autoload` to publish a large number of functions without incurring the overhead of reading their function definitions at startup.

3.3.2 Loading commands

There are several methods of loading commands:

1. List them directly in the profile file.
2. List them in a separate file and use an `autoload` statement in the profile file.
3. Invoke the Document Editor (`editdoc`) with an `-exec` argument
4. Use the `read-commands` function from within the Document Editor.
5. List them in a separate file and use a `read-commands-from-file` statement in the profile file.

Loading commands from a profile file

You can place definitions directly in a profile file. In general, you should refrain from this practice to minimize startup time for the Document Editor.

Loading commands from a separate file

You can improve the startup time for the Document Editor by placing your SEL commands in a separate file. You can then place a reference to each function in your profile file with an `autoload` statement. The functions will be read only when actually invoked by the user.

The syntax of the autoload command is:

```
autoload (function-name, "filename",
                keymap-name,"key-sequence");
```

The *filename* is the name of the file that contains the definition of the function, and may also include a specific path. The *keymap-name* and *key-sequence* are optional. If they are included, the function is bound to the key sequence in the keymap. For example, if you include the statement:

```
autoload (mmerge, "mmerge.sel");
```

in your profile file, you can then invoke execute-command-dialog (the default key binding is **Esc-x**) and select mmerge. If you include the statement:

```
autoload (mmerge, "mmerge.sel", text-keymap, "Esc-#");
```

in your profile file, you can then invoke mmerge by pressing **Esc-#**.

Note that you must precede any autoload statements in your profile file with a library-search-path variable that defines the directory paths containing your SEL definitions.

Loading commands with -exec *argument*

You can load commands from a file directly at editdoc startup by including the argument:

```
-exec filename
```

where *filename* is the name of a text-only file containing commands. For example:

```
editdoc -exec /usr/slate/SEL/public_definitions.sel
```

Also, `editdoc` can be invoked with the `-l` flag:

```
editdoc -l
```

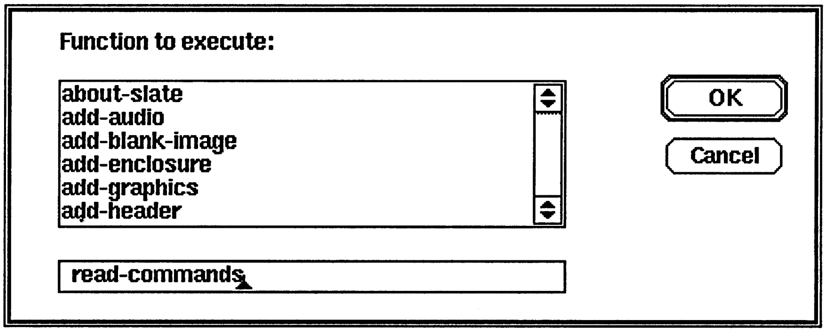
This causes `editdoc` to "listen" on its standard input and accept lines with the same format as its command line arguments. For example, `editdoc` can be sent (on its standard input), additional lines of the form:

```
-exec filename
```

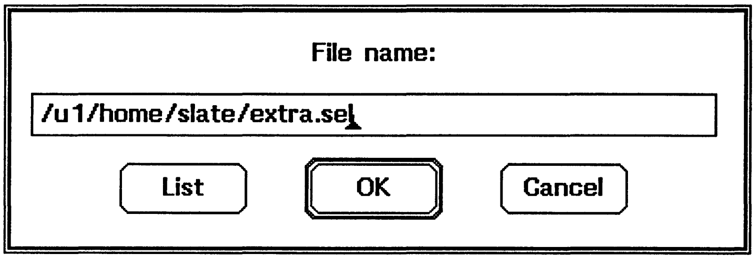
where *`filename`* is again the name of a file of extension language commands. These commands are read and executed as they are received.

Loading commands within the Document Editor

You can read and execute a file of extension language commands directly from the Document Editor. To do this, select **BBN-Load SEL Commands** from the main menu, or invoke the `execute-command-dialog` function (the default binding for this is **Esc-x**) and then enter the `read-commands` function in the dialog box:



Another dialog box appears, requesting a file of extention language commands:



/u1/home/slate/extra.se

Enter the name of the text file that contains the extension language commands. If the file is not located in the current directory, you should precede it with the appropriate pathname. Alternatively, you can first use the `change-directory-dialog` function before invoking `read-commands` to redefine the search path.

You can also compose files of extension language commands during a Document Editor session. After composing the file of extension language commands, write the file as text, and then use the `read-commands` function to read the text file. Alternatively, you can write an extension language function that writes the current buffer (which is assumed to contain extension language commands) as a text file in a temporary area, and then executes that text file. This is a very useful way to develop and debug extension language functions. All of these methods are described and illustrated in Section 3.4.

Once a command is loaded, any variables or functions defined in the file are defined in `editdoc`, and any statements in the file are executed. These functions can now be invoked directly from `editdoc`, either by binding them to key sequences or by invoking `execute-command-dialog` (see Section 3.3.2).

3.3.3 Variables in `editdoc`

The BBN/Slate `editdoc` program utilizes two basic categories of variables: *document manager variables* and *document editor variables.* The document manager variables, of which there are only a few, control the appearance and behavior of the Document Manager. The document editor variables, which are many in number, control the appearance and behavior of the Document Editor.

The document editor variables can be further separated into three categories:

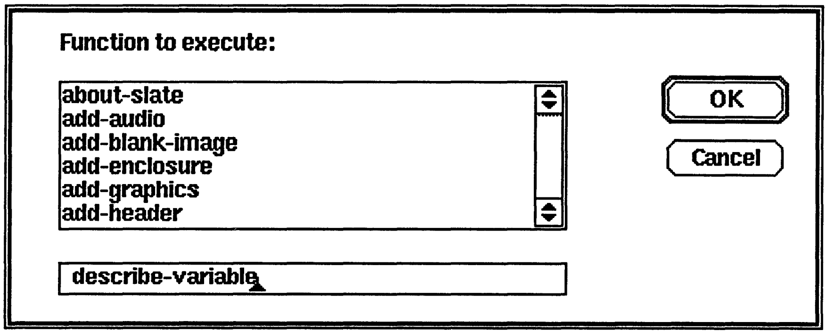
- *editor variables.* These variables control the operation of the Document Editor, and should be set in the `.slate_editor.init` profile file. An example of this type of variable is `classify-names`, which controls the classification markings used to label document elements.
- *document variables.* These variables control the attributes of a document that should persist from one editing session to another, and are set by defining document attributes in the Document Editor. An example of this type of variable is `page-footer`, which you set in the **Preview/Print...** dialog box of the **Print** menu.
- *buffer variables.* These variables control the attributes of a document that should persist during an editing session, but should not persist across editing sessions. These are internal variables set and used by the Document Editor.

All variables are stored in one of three *variables tables,* depending upon the scope of the varible. Variables associated with the current buffer are stored in the *buffer variables table.* Variables associated with the document are stored in the *document variables table.* Variables associated with the Document Editor in general are stored in the *global variables table* (for example, `page-left-pixels`, which controls the margin between the left edge of the Document Editor window and the beginning of the text area).

For a comprehensive list of BBN/Slate variables, consult Appendix A.

3.3.4 On-line Information

There are several functions that provide on-line information. You can access them by selecting **BBN-Invoke Function** from the main menu, or by invoking `execute-command-dialog` (the default binding for this function is **Esc-x**). Either method displays a dialog box (see below). You then enter one of the function names described in this section. For example:


Function to execute:

about-slate
add-audio
add-blank-image
add-enclosure
add-graphics
add-header

describe-variable

Rapid location and entry of names

You can either use the mouse to scroll through the list and select one of the function names, or you can type the name of the command directly from the keyboard. You can use the BBN/Slate typing completion feature to assist you in locating the desired name: type the first set of characters that distinguish one name from all others and then press **Space**. The name is completed. You can **LEFT click** on the **OK** button or press **Return** to confirm the selection. Typing some characters and then **Return** completes the name and, if the name is unique, accepts the choice. For example, typing:

```
   desc Space -d Space Return
```

locates and enters the `describe-document-variable` function.

On-line information functions

The following on-line information functions are available:

`describe-command`
This function displays a description of a function in the extension language. It displays a dialog box requesting the function name. The dialog includes a scrollable alphabetical list of the user-callable functions defined in editdoc and any public user-defined functions. Enter the name of the function to describe (using space for recognition of an initial substring) or select it from the scrollable list of functions. A new dialog box will be displayed with a description of the function selected including the types of its arguments and returned result.

`display-functions-in-buffer`
This function displays the names and descriptions of all the user-callable functions defined in `editdoc` and any user-defined functions. The output is placed in a separate buffer (named `#help`). The information displayed is the same as that given by `describe-command`. (Because the system processes several hundred functions, this command can require a long time to execute.)

`apropos-dialog`
These functions display the names of all the user-callable functions defined in `editdoc` and any user-defined functions that contain the specified word. The `apropos-dialog` function queries the user for the word; the `apropos` function takes the word as an argument. The names of functions are displayed in a scrollable list. If you choose one of the functions from this list, then a description of that function is displayed. The information displayed is the same as that given by `describe-command`.

describe-variable
describe-buffer-variable-dialog
describe-document-variable-dialog
describe-global-variable-dialog
These functions are used to display a description of a variable. They display a dialog box requesting the variable name. This dialog includes a scrollable list of variables. Enter the name of the variable to describe or select it from the scrollable list of variables. The `describe-variable` function describes variables in any of the BBN/Slate variable tables. It searches for variables first in the create-buffer-variable variable table, next in the document-specific variable table, and finally in the global variable table.

The `describe-buffer-variable` function describes variables in the create-buffer-variable variable table for the current buffer.

The `describe-document-variable` function describes variables in the document-specific variable table for the current document.

The `describe-global-variable` function describes global variables that are either defined in `editdoc` or declared in a `global` statement in a file of extension language commands.

The description of the variables includes the current value of the variable and a description of the flags set for that variable. The following flags are defined:

user-readable	The user is allowed to read the variable in a file of extension language commands.
user-writable	The user is allowed to assign a value to the

variable in a file of extension language commands or by using one of the `set` functions (`set`, `set-buffer-variable`, or `set-document-variable`).

`defined`	The variable is defined. In an extension language file, undefined variables can be assigned a value (thus defining them), but they cannot be read.
`create-buffer-variable`	This variable is replicated for each buffer created.
`document-specific`	This variable is replicated for each document created.

Example

An example of looking up on-line information would be to investigate the current value associated with the variable that tracks the page width. Invoking `describe-document-variable` for `page-width` provides a description (in a scrolling message box) such as the following:

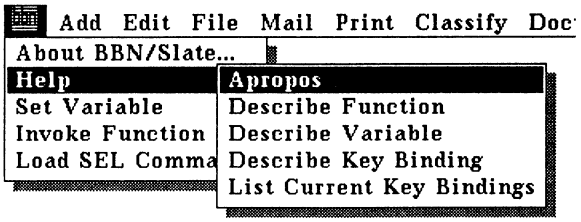
```
A document specific variable.  The width of a page displayed
on the screen.  The default is 6.5 inches.
The following flags are set:
document-specific
defined
user readable
user writable
The value is: 6.5i.
```

Help from the BBN menu

The BBN menu (at the far left of the main menu) provides a **Help** submenu with convenient access to several of the functions discussed in this section.

Help from the BBN menu

The BBN menu (at the far left of the main menu) provides a **Help** submenu with convenient access to several of the functions discussed in this section.



Apropos calls `apropos`; **Describe Function** calls `describe-command`; and **Describe Variable** calls `describe-variable`. In addition, **Describe Key Binding** and **List Current Key Bindings** allow you to obtain information on specific or all key bindings, respectively.

3.3.5 Special extension language functions

Among the user-callable functions listed in Appendix B, the following functions have been defined in the editor specifically for use in the extension language. Special note is made here of these functions because they are useful for manipulating locations in and regions of the current document.

beginning-of-buffer	Return the location of the start of the document. It takes no arguments. The type of the returned value is mark.
end-of-buffer	Return the location of the end of the document. It takes no arguments. The type of the returned value is mark.
current-point	Return the current insertion point (cursor location). It takes no arguments. The type of the returned value is mark.

current-region

Return the current selected region, the area between the current insertion point and the current mark. Regions are used as arguments to several `editdoc` functions, most notably to the search functions. It takes no arguments. The type of the returned value is region.

beginning-of-region

Return the start of the specified region. It takes one argument of type region. The type of the returned value is mark.

end-of-region

Return the end of the specified region. It takes one argument of type region. The type of the returned value is mark.

create-region

Create a new region. It takes two arguments of type mark, which are the start and end of the region created. The type of the returned value is region. For example, the call:

```
region = create-region(beginning-of-buffer(),
    end-of-buffer())
```

creates a region that is the entire document.

text-set-point

Set the insertion point to the specified text location in the document. This function guarantees that the insertion point is set in text (i.e., if the location specified is in a non-text object, then the point is set in the next text object). It takes a single argument of type mark and returns no value.

document-set-point

Set the insertion point to the specified location in the document. It takes a single argument of type mark and returns no value.

text-set-mark	Set a mark at the current point. It takes no arguments and returns no result.
mark-is-equal	Compare two marks. It takes two arguments of type mark and returns true (1) if they are equal and false (0) otherwise. This function must be used to compare values of type mark (for example, the results of calls to `beginning-of-buffer`, `end-of-buffer`, `current-point`, and `current-mark`); they cannot be compared using the equality operator.
mark-add-offset	Add an integer offset to a mark and create and return a new mark. It takes two arguments. The first is of type mark and the second is an integer offset of type int. The type of the returned value is mark.
mark-is-in-region	Determine if a mark is within a region. If the mark is within the region, it returns true (1), otherwise it returns false (0). It takes two arguments. The first is of type mark and the second is of type region.
mark-convert-to-location	Convert a mark to a location in a document. It takes one argument of type mark, and converts it to an integer, which is the location in the document. The type of the returned value is int.
location-convert-to-mark	Convert an integer location in a document to a mark. It takes one argument of type int, which is a location in a document and returns a value of type mark.

3.3.6 Other extension language facilities

The following is a description of the extension language facilities for error handling, executing functions repeatedly, and setting or determining the last function executed.

These facilities make use of global variables that can be read directly in the extension language (simply by referring to the variable), but which must be set by calling `editdoc` functions. That is, these variables cannot be set by simply assigning them a value in an extension language assignment statement.

Error handling

The global variable is used in `editdoc` functions to indicate whether or not an error has occurred. If an error has occurred, this variable is set to 1; if no error has occurred, this variable is set to 0.

To read `global-error` in the extension language, simply refer to it. To set `global-error`, call `set-global-error-code` with a single integer argument, which is the value to assign to `global-error`.

For example, the following extension language function invokes `text-down-line`, which moves the cursor down one line, until the end of the document is reached. `Text-down-line` sets `global-error` when it reaches the end of the document.

```
define public void go-to-end()
    {
    set-global-error-code(0);
    # clear any previous error set
    while (1)
        {
        text-down-line();
        if (global-error)
        return;
        }
    }
```

Repeat count

The global variable global-repeat-count is the number of times that an operation should be repeated. The repeat count is set by the function gather-repeat-count (for which the default key binding is ^U).

The repeat count can be read in the extension language simply by referring to global-repeat-count. To set global-repeat-count in the extension language, invoke the function set-repeat-count, with an integer argument which is the new repeat count.

For example, the following extension language function, indent-region, invokes a user-defined function called indent-worker. The indent-worker function takes an argument that is the number of spaces to indent the selected region; this argument is taken from global-repeat-count. When the repeat count is used in this way, it is important to reset the repeat count to its default value of 1 before executing any other functions. Many editdoc functions use the repeat count, and in this example the function that should be repeated is indent-worker and not any function that indent-worker may call.

```
define public void indent-region()
local count;
    {
    count = global-repeat-count;
    set-repeat-count(1);
    indent-worker(count);
    }
```

Last function executed

The global variable global-last-function contains the name of the last function that was executed. This can be read simply by referring to the variable. It is set by invoking the function set-last-function-executed with a string argument that is the name of a function.

When an extension language function is executed, `global-last-function` is the name of the last `editdoc` function executed from the keyboard (not the last function executed from the extension language). `Global-last-function` is only set when an `editdoc` function is invoked from the keyboard or by explicitly calling `set-last-function-executed`.

For example, suppose that the user visits a file by selecting **File-Read-In New Buffer** from the top-level menu, which executes the `editdoc` function `visit-file-dialog`, and then calls the user-defined function `test`:

```
define private void worker()
    {
    text-insert-string(global-last-function);
    }

define public void test()
    {
    text-add-item("noindent", 0);
    text-insert-string("last function executed was: ");
    text-insert-string(global-last-function);
    text-newline-check();
    set-last-function-executed("test");
    text-insert-string("last function executed was: ");
    worker();
    }
```

The following two lines are written in the buffer:

```
    last function executed was: visit-file-dialog
    last function executed was: test
```

Temporary buffers

When you create a new buffer in an SEL definition (for example, by using `add-empty-buffer`), you can specify that the buffer is temporary (sometimes called a *scratch* buffer) by beginning the buffer name with a percent sign (%). For example:

```
add-empty-buffer("%temp","");
```

This has the effect of eliminating the overhead of automatic backup and checkpointing for these buffers, and improves performance. See Section 3.5.4 for a sample program that employs a temporary buffer.

3.4 Function definition and development

This section presents a comprehensive example of creating, writing, testing, and installing custom BBN/Slate functions. It leads you through the definition and development of two new functions. Then, it shows you how to load them and bind them to key sequences so that you can use them in the same manner as any default BBN/Slate function.

It assumes that you have read the previous sections in this chapter and have an overview of SEL syntax, but that this is your first attempt to write a new function.

Creating a new function involves the following steps:

1. Plan the new functionality; review existing functions and choose those that you will use to create the new function (Section 3.4.1).
2. Edit a text file (*not* a normal BBN/Slate file) and enter the function name and definition (Section 3.4.3).
3. Test the definition (Section 3.4.5).
4. Add the function to your profile file so that it is automatically available at startup; if appropriate, also assign a key binding (Section 3.4.8).

You can profit greatly from spending time reviewing the list of available functions (Appendix B). You may also find it helpful to consult Section 3.5, where more-advanced examples are provided.

The steps described in this section are designed to demonstrate an effective methodology for developing your own BBN/Slate functions. Although it is possible to create a new function directly in your profile file, then debug and test it by repeatedly starting up the Document Editor, it is not an efficient way to work. Special tools and techniques are available to assist you.

3.4.1 Planning new functionality

This section describes the steps you should take prior to entering extension language statements into a function definition. Information is provided about how you can gain a working knowledge of information on existing BBN/Slate functions.

An important initial task is to plan the addition of any new functions. You can create new functions by using:

- functions provided with BBN/Slate
- functions you have created
- a combination of the above

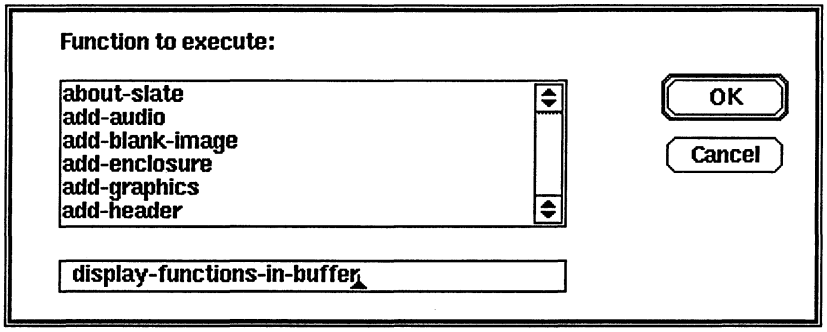
A basic design strategy is to employ existing functions with lower functionality to create functions with higher functionality. To facilitate this, you should be as familiar as possible with the many functions provided with BBN/Slate. There are two steps you can take that will assist you in learning about functions: produce a hard copy of all existing functions, and use the on-line `apropos-dialog` function to locate potentially useful functions.

Reviewing functions with hard copy

Initially, you will find it helpful to study Appendix B, which lists all built-in system functions for this release. Once you begin to add your own functions, you can create your own listings that include your own user-specific or site-specific functions.

To create a new listing of all current functions (similar to Appendix B), take the following steps.

From within the Document Editor, enter **Esc-x** to bring up the function dialog box. Enter `display-functions-in-buffer`, then press **Return** or **LEFT-click** on **OK**:



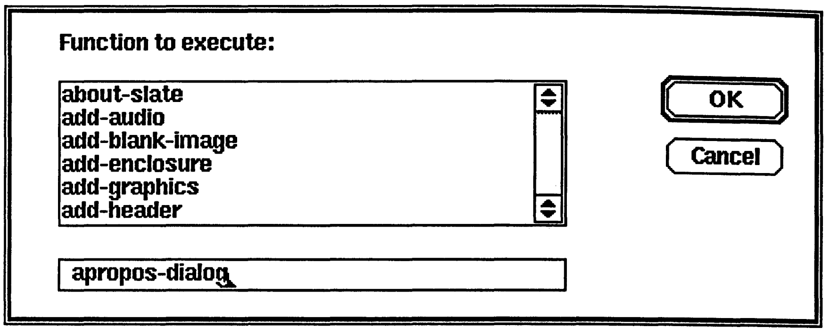
This function displays the names and descriptions of all user-callable functions (initially, these are built-in system functions only; as you add your own functions, these will be folded into any future function listings that you create). The output is placed in a separate buffer named `#help`, which you can then save and print out as you would any document.

Note: Because the system must process several hundred functions, invoking `display-functions-in-buffer` can take a long time to execute.

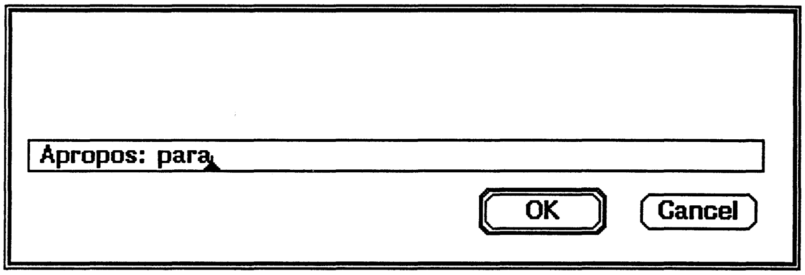
Reviewing functions with on-line lookup

With hundreds of functions to choose from, it can be difficult to be sure, just by looking at a hard copy of the function listings, whether or not you have the best function in mind to do the job. To assist you in surveying possible candidates, the `apropos-dialog` function lists any function whose name contains the keyword or string you enter.

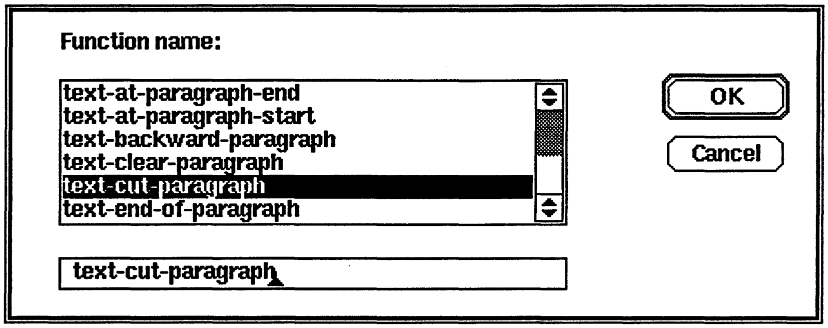
From within the Document Editor, enter **`Esc-x`** to bring up the function dialog box (the scrolling region contains the names of all user-callable functions). Enter `apropos-dialog`, then press **Return** or **LEFT click** on **OK**:



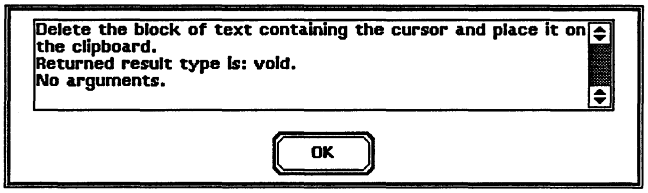
The `apropos-dialog` dialog box appears. Enter any keyword or string that you believe might apply to the action you are planning to perform. For example, if you are looking for a function that deals with paragraphs, you could enter `para` as your search string:



The function dialog box reappears, only this time the scrolling region contains only those functions whose names contain `para`. To learn more about any of these functions, you can select it, then press **Return** or **LEFT click** on **OK**:



You will then see an information box about the function:



Delete the block of text containing the cursor and place it on the clipboard.
Returned result type is: void.
No arguments.

This box contains three pieces of information about the function definition:

- the help message that is included as part of the function definition
- the type of any returned value from the function (void, as in the above example, means that there is no returned value)
- the number of arguments, if any

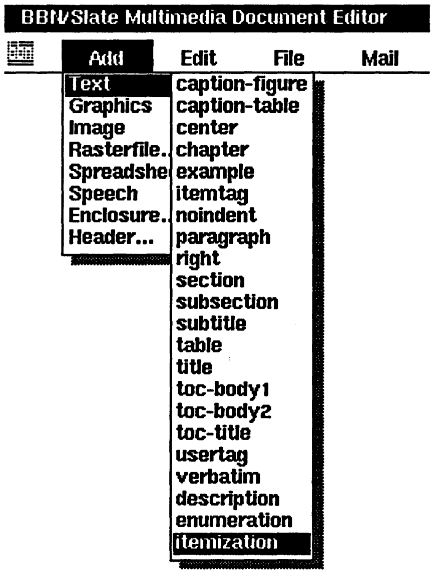
You may find it helpful to repeat `apropos-dialog` several times, using a number of synonyms, to be sure that you have surveyed all the possible candidates for the action you want to perform.

3.4.2 Tutorial example

This section explains how to add two new functions.

Building an itemization function

The first function offers an alternate method for creating an itemization (typically, this action might be requested by users who prefer direct actions from the keyboard). For example, a common menu-directed method of creating an itemization is to select **Add-Text-itemization** from the main menu:



You then begin typing the text of the list item. The objective of the new function is to perform this action using the key binding **Esc-i**. In other words, typing **Esc-i This is a list item** results in the following:

o This is a list item

It might seem you just create a simple key binding for this action, but this is not the case. Searching the listing of available functions reveals that there is no existing function for creating an itemization. This is because BBN/Slate does not provide a separate function for the creation of each text format. (Indeed, this is not practical, since users are free to create their own text formats with unique names.) Instead, it provides a general-purpose function, `text-add-item`, whose description is as follows:

```
text-add-item
Takes two arguments:  the name of a text format,
and a boolean which is true if the format was
automatically generated.  Add an instance of the
specified text format. Returned result type is: void.
Argument types are: string, int
```

Because this function requires arguments, and arguments cannot be provided as part of a `bind` statement, you must use the extension language to supply those arguments. You do this by stating the function name, and providing its arguments enclosed in parentheses:

```
text-add-item("itemization", 0);
```

Additionally, every function begins with a `define` statement and encloses the complete definition within braces (see Section 3.2), so the complete function reads as follows:

```
define public void text-add-itemization()
{
text-add-item("itemization", 0);
}
```

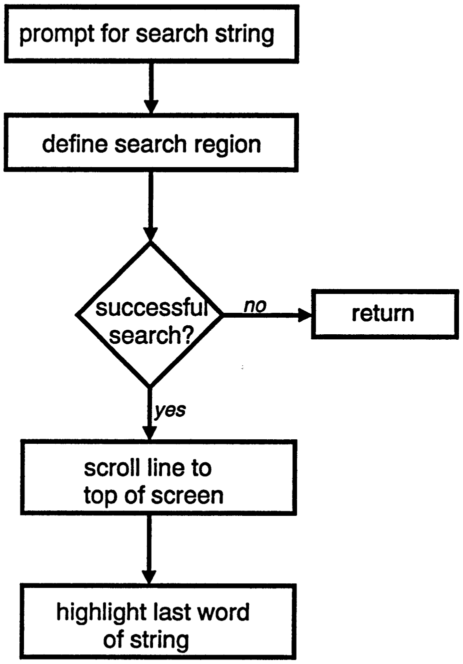
The `public` declaration makes this a function that can be bound to a key sequence or invoked from a menu; `void` specifies that this function returns no arguments. The last part of the `define` statement names the function, `text-add-itemization.` The succeeding sections illustrate how you process and activate it.

Building a new search function

The second (and separate) new function you can add is a new search capability that performs the following tasks (this function was introduced briefly in Section 3.1):

- prompts the user to enter a search string
- searches the document for the specified string
- if the string is found, scrolls the paragraph containing that string to the top of the display area
- highlights the last word of the string

The logical work flow of this function can be represented as follows:



This function is built from several existing lower-level BBN/Slate functions. You can survey potential lower-level functions by locating them with `apropos-dialog` and reading their descriptions. Entering `string`, for example, yields the function `read-string-dialog`:

```
read-string-dialog
Display a dialog for obtaining a string from the
user and return that string. This takes two
arguments, the first is a message that is placed at
the top of the dialog, and the second is a prompt
that is placed in a box within the dialog.  The
dialog also has confirm and cancel buttons.  If the
user hits the cancel button, this function sets the
global variable, global-error.
Returned result type is: string.
Argument types are: string, string.
```

This function could be used to query a user for the search string and store the reply in a variable called `search-string`:

```
search-string = read-string-dialog(
   "Keyword or phrase to search for.",
   "Word or phrase: ");
```

Entering `search` to the `apropos-dialog` prompt yields the `text-search` function, which can be employed to do the actual searching. In turn, the first argument to `text-search` is the search string; the second is the region to search. Entering `region` to the `apropos-dialog` prompt reveals a `create-region` function. You can use `create-region` to define the search area and store it in the variable `search-region`:

```
search-region = create-region(beginning-of-buffer(),
   end-of-buffer());
```

Once you know what to search for and where to search for it, you can then proceed with the actual `text-search` function. Its third argument is a Boolean integer that you set to 1 (TRUE) to indicate that you want the search to be case insensitive:

```
text-search(search-string, search-region, 1)
```

As shown in the logic diagram, you also want to differentiate between successful and unsuccessful searches. Only successful searches should take action. You can do this by placing `text-search` within an `if` expression:

```
if (text-search(search-string, search-region, 1))
    {

    }
```

As shown in the logic diagram, successful searches should result in scrolling the text to the top of the screen and highlighting the last word of the string that is found. You now need to locate functions that correspond to these actions.

Using `apropos-dialog` to search for the topics *scroll* and *highlight* does not yield the functions for which you are looking (for example, the function `scroll-up` moves the display up by one line. You want to proceed to the top of the pane; so, take some time to review a hard copy listing of the many text functions available. Each of these begins with the prefix "`text-`" (you can examine these now by referring to Appendix B). This reveals two useful functions: `text-line-to-top` and `text-select-current-word`.

In the BBN/Slate extension language, function calls must include parentheses following the function name. Because there are no arguments for either of these functions, simply include empty parentheses (and place them within the `if` statement):

```
if (text-search(search-string, search-region, 1))
   {
   text-line-to-top();
   text-select-current-word();
   }
```

If you include no further statements, unsuccessful searches simply "fall out" of the function and it returns. In practice, this means that unsuccessful searches leave the cursor where it was before the search was initiated.

The complete code is as follows:

```
define public void search-word()
"Search for a word, scroll to top, and highlight it."
{
local search-string, search-region;
search-string = read-string-dialog(
   "Keyword or phrase to search for.",
   "Word or phrase: ");
search-region = create-region(beginning-of-buffer(),
   end-of-buffer());
if (text-search(search-string, search-region, 1))
   {
   text-line-to-top();
   text-select-current-word();
   }
}
```

This completes the design shown in the logic diagram.

The next section explains how to create a file that contains both the itemization and searching functions. Creating definition files is an essential part of developing BBN/Slate function definitions.

3.4.3 Edit a definition file

Once you have completed planning the function, you are ready to enter SEL statements into a file.

The code for building these two new functions is now ready to place within a *definition file*. It is important to keep in mind that definition files *must always be text files, not BBN/Slate document (.slt) files.* The reason is evident if you examine document files from the operating-system level instead of within BBN/Slate. Each file contains an introductory section with a considerable amount of BBN/Slate-specific style information, plus special content tags that identify the text (or other media types) that may be present in the file. Definition files, on the other hand, should contain none of this special information — only the extension language text.

Where to store SEL files

Definition files can be located in any convenient directory of your operating system. Simply make sure that these directories are included in a `library-search-path` statement in your profile file (`.slate_editor.init`). This statement should include one or more directory paths, each separated with a colon, and the entire path(s) in quotation marks:

```
library-search-path =
    "/home/smith/Slate/SEL:/usr/slate/lib/SEL";
```

You should place this statement near the beginning of your profile file, before any mention of site-specific functions located in those directories.

A definition file can contain one or more function definitions. The definitions need not be related, but you may find it helpful to group different types of functions in separate files.

3.4.4 Tutorial example

This section explains how you can create definitions in the following ways:

- outside of BBN/Slate (for example, using a text editor such as `emacs`)
- within BBN/Slate (storing the definition as a text file)

If you use an outside editor, it must store the text as a plain, ASCII text file (for example, there can be no control codes present). If you use BBN/Slate, you must use choose **File-Write-As Text** from the main menu when saving the file. Both methods are described here.

Creating text-only files outside of BBN/Slate

The first tutorial example (on itemization) illustrates how you can create files outside of BBN/Slate for use with function definitions.

Step 1

Using a workstation window outside of BBN/Slate, create a subdirectory for SEL functions (if you have not already done so). For example:

```
cd /home/smith/Slate/SEL
```

Although it is not necessary to provide a separate subdirectory for this purpose, it makes it easier to locate SEL functions, and it avoids cluttering up regular BBN/Slate directories with SEL files.

Step 2

Change directories to this new SEL subdirectory and start up your normal text editor. Choose a new filename that is appropriate to the contents, such as `ListItems.sel`. (Later, you can add separate definitions to this file for other BBN/Slate list items, such as enumeration.) For example:

```
cd /home/smith/Slate/SEL
emacs ListItems.sel
```

Step 3

Enter the complete itemization function definition:

```
define public void text-add-itemization()
    {
    text-add-item("itemization", 0);
    }
```

Step 4

Save the file.

Creating text-only files within BBN/Slate

The second tutorial example (on word searching) illustrates how you can create files within BBN/Slate for use with function definitions.

Step 1

Using a workstation window outside of BBN, create a subdirectory for SEL functions (if you have not already done so). For example:

```
mkdir /home/smith/Slate/SEL
```

Step 2

Now switch to your BBN/Slate Document Editor window and begin with an empty buffer. If your main buffer is empty, for example, you can do this by selecting **Editor-Buffer-Switch To-main** from the main menu. Enter the entire text for the search function:

```
define public void search-word()
"Search for a word, scroll to top, and highlight it."
    {
    global search-string, search-region;
    search-string = read-string-dialog(
    "Keyword or phrase to search for.",
    "Word or phrase: ");
    search-region = create-region(beginning-of-buffer(),
        end-of-buffer());
    if (text-search(search-string, search-region, 1))
        {
        text-line-to-top();
        text-select-current-word();
        }
    }
```

Step 3

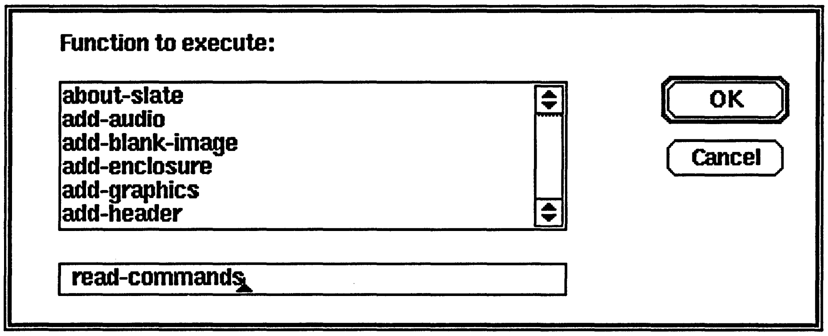
Choose **File-Write-As Text** from the main menu. Because this buffer has not yet been saved, you are prompted for a pathname and filename. Enter the pathname of the SEL directory you created, then enter the filename `SearchWord.sel`.

3.4.5 Test the definition

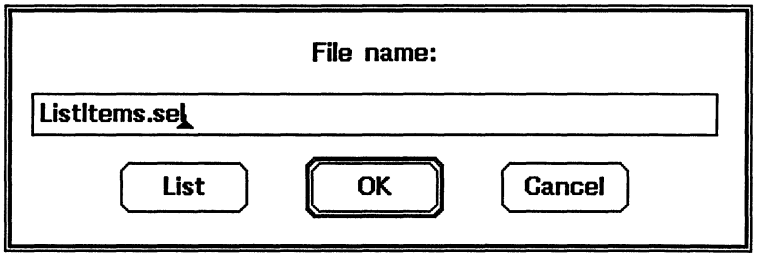
Once you write the definition and save it in a file, you are ready to test it. The first tutorial example that follows deals with the itemization function: this is a short, simple function and it can be tested in a straightforward manner. The second tutorial example deals with the word searching function. This is a more complex function, and several tools and techniques are introduced that you can use. These will be helpful whenever you need to refine and debug more complex functions.

3.4.6 Tutorial example: `ListItems.sel`

You are now ready to explicitly read and interpret the extension language commands in the `ListItem.sel` file. To do this, press **Esc-X**, then enter the function `read-commands` in the dialog box:

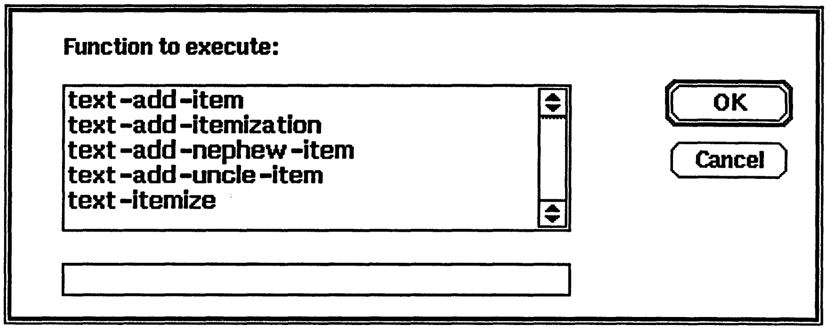


This prompts you for the file name containing the definition (to obtain a listing of available files in the current or other directories, press **Esc**):



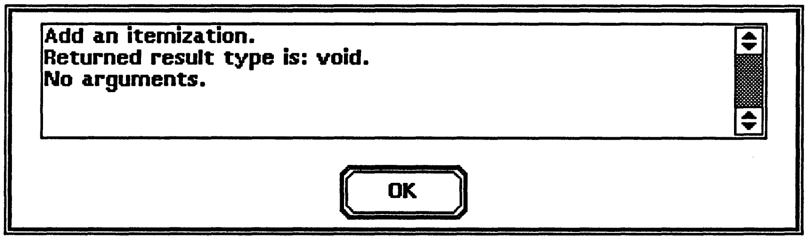
ListItems.sel

The `read-commands` function reads the definition, and adds it to the collection of existing definitions. To make sure it is there, enter an `apropos-dialog` command and search for `item`. You should see `text-add-itemization` among the other functions that include the string `item` in their name:



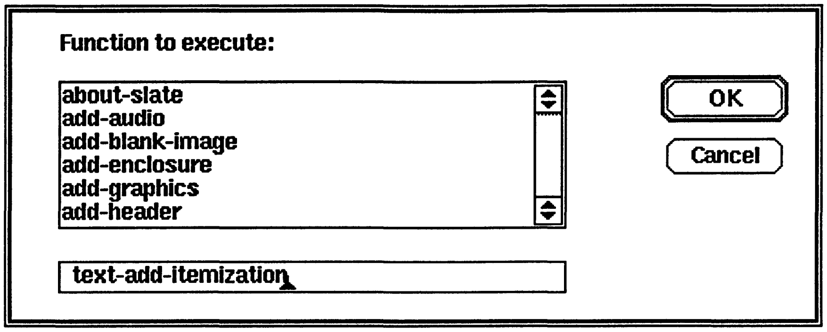
```
text-add-item
text-add-itemization
text-add-nephew-item
text-add-uncle-item
text-itemize
```

Select `text-add-itemization` and **LEFT click** on **OK** to test the help string. You should see the following:



```
Add an itemization.
Returned result type is: void.
No arguments.
```

LEFT click on **OK** to close the help window. Now, test the function by entering **Esc-X** and the new function name; then, press **Return** or **LEFT click** on **OK**:



Function to execute:

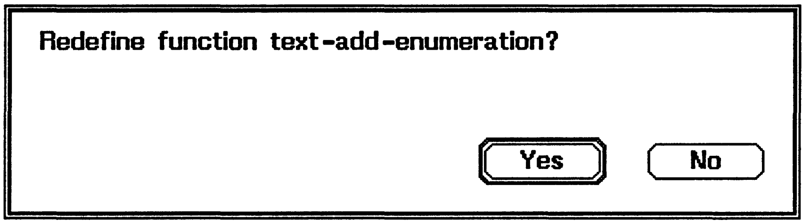
about-slate
add-audio
add-blank-image
add-enclosure
add-graphics
add-header

text-add-itemization

This should create a new itemization at the current cursor position.

Subsequent editing of a function

From time to time, you may need to revise an existing function definition file. You can do this at any time, using the same `read-commands` function as before. It re-reads the file, processes it, and checks to see if there is already a definition of the same name (in the case of the current example, a function named `text-add-itemization`). It will ask you for permission to redefine it:



Redefine function text-add-enumeration?

Press **Return** or **LEFT click** on **OK** to test your changes and redefine the function. If there are other pre-existing functions in the file as well, you are prompted in the same manner for each of them; select **No** if you have not changed the function.

It is possible to suppress this redefinition prompt, if desired, by taking either of the following steps:

1. You can suppress *all* redefinition prompting by placing the following line in your profile file:

```
redefine-silently = 1;
```

2. You can suppress redefinition prompting for selected functions by first undefining them. To do this, place an `undefine` statement at the beginning of the function definition. For example, the first two lines of the word searching function would appear as follows:

```
undefine search-word
define public void search-word()
```

3.4.7 Tutorial example: `SearchWord.sel`

This section presents a development tool that will assist you in designing new functions more quickly and efficiently.

When developing a number of functions, it is convenient to automate the process of reading the definition file and executing the commands to test them. You can do this by inserting the following lines in your `.slate_editor.init` profile file, then restarting the Document Editor:

```
#Execute a buffer of extended commands
define public void execute-buffer()
"Execute a buffer of extended commands."
{
   write-buffer-to-file-as-text(current-buffer(),
       "/tmp/SELtest")
   read-commands-from-file("/tmp/SELtest")
}

bind(default-keymap, "^X^E", execute-buffer)
bind(text-keymap,    "^X^E", execute-buffer)
```

The binding for `default-keymap` applies when the cursor is not active in any media type (for example, when you first open a buffer); the binding for `text-keymap` applies when the cursor is active in a text area. The newly defined function `execute-buffer`:

- writes the current buffer containing your SEL function as a text file named `SELtest` in your `/tmp` directory
- performs the `read-commands-from-file` function on that file

This allows you to use the following strategy:

1. Create a new definition in a BBN/Slate buffer.
2. Enter **^X ^E** to invoke the `execute-buffer` function; this saves the file temporarily in `/tmp`, reads its commands, and makes it available for calling.
3. Test the function by entering `execute-command-dialog` (**Esc-X**), then the function name. (Note that if the function is declared to be `private`, it is not displayed in the scrolling region.) Evaluate how it works.
4. Use BBN/Slate to modify the definition as necessary, and repeat steps 2 through 3 until the function is debugged and tested. If you are modifying the function during a later session, you can read its definition into a new buffer using **File-Insert-Text File** from the main menu.
5. Store the final version as a text file in an appropriate directory (e.g., `/usr/slate/lib/SEL/ListItems.sel),` using **File-Write-As Text** from the main menu. Additionally, make sure that the directory path is listed in the `library-search-path` in your profile file.

Testing the example

With this development tool now in place, you can proceed with the development of the word searching function:

Step 1

Starting from a new, empty buffer in the Document Editor, select **File-Insert-Text File** to read the file `SearchWord.sel` into the buffer.

Step 2

Enter **^X ^E**. (*Note:* the first time you submit a function, it is created. On subsequent occasions, you are prompted whether you want to redefine the function, since it already exists. If you have modified it, answer **Yes.**)

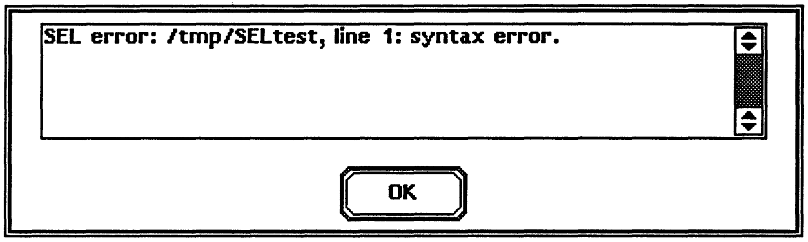
Step 3

Enter **Esc-X**; respond to the dialog box by entering the function name, `search-word`. Test it to make sure you are prompted to enter a search string, and that it performs as expected for existing and nonexisting keyword searches.

Handling errors

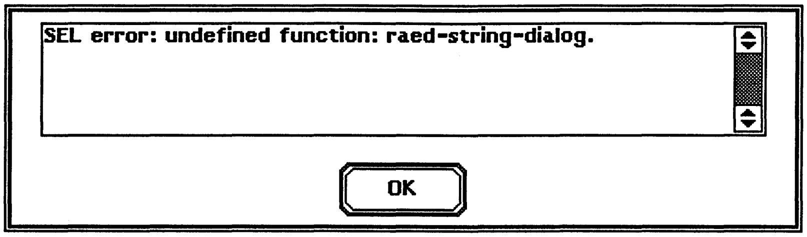
When developing functions, it is normal to experience coding errors from time to time. Errors are reported (via dialog boxes) at different times, depending on the nature of the error.

Syntactical errors are reported by line number at the time the function is parsed. For example, misspelling the reserved word `define` on the first line and then invoking `read-commands` results in the following error report:



```
SEL error: /tmp/SELtest, line 1: syntax error.
```

Some errors are syntactically correct, but incorrect in meaning. For example, a misspelled function name, such as `raed-string-dialog` instead of `read-string-dialog` (line 5 of the `SearchWord` function) results in the following:


```
SEL error: undefined function: raed-string-dialog.
```

Errors may be near rather than actually on the line reported. For example, omitting the parentheses from a function name so that instead of:

```
define public void search-word()
"Search for a word, scroll to top, and highlight it."
```

it reads:

```
define public void search-word
"Search for a word, scroll to top, and highlight it."
```

results in an error reported on line 2. This is because it is syntactically acceptable to place the parentheses on the line after a function name, as in the following:

```
define public void search-word
()
"Search for a word, scroll to top, and highlight it."
```

Only when parentheses were not found at the beginning of line 2 was the error reported.

If there are multiple errors, only the first error encountered is reported, and the remainder of the file is not read.

Refining examples

A worthwhile strategy to employ is to build and test basic features first, making sure they are sound. You can then add additional capabilities one at a time. This permits you to concentrate on debugging new areas of functionality.

Debugging aids

You can also employ the `display-message` function to display pop-up messages at strategic points in your program:

```
   display-message("Now within inner loop.");
```

Additionally, you can use the `global-error` variable (see Section 3.3) to assist with debugging.

3.4.8 Binding and activating functions

Once you have completed the development of a function, you should include a reference to it in the profile file. The profile file is read at Document Editor startup (and at no other time); with a function reference there, you no longer need the `read-commands` and `execute-buffer` functions you used during the development and test phase. In essence, it becomes a permanent part of your function library.

You should also decide if it is appropriate to assign a key binding to the function. You can explicitly invoke any function that does not require arguments by invoking the `execute-command-dialog` function (for which the default key binding is **Esc-x**) and entering the function name in the dialog box; this may be sufficient for functions that will be used infrequently. In many cases, however, it is convenient to bind a function key or key sequence to enable speedy invocation of a new function.

Autoload

It is possible to include the full text of function definitions in the profile file (as with the `execute-buffer` function in the preceding section). However, this creates additional overhead at startup. To address this issue, BBN/Slate provides an *autoload* command.

The `autoload` command allows you to read function definitions from files when the functions are invoked. This command can be used to include a large number of functions without incurring the overhead of directly reading all the function definitions at startup.

The syntax of the `autoload` command is:

```
autoload (function-name, "filename",
          keymap-name, "key-sequence")
```

The *filename* is the name of the file that contains the definition of the function. If you include only a file name, the path to the directory where it is located must be included in the `library-search-path`. Alternately, you may specify an explicit path and file name, in which case the `library-search-path` is ignored.

The *keymap-name* and *key-sequence* are optional. If they are included, the function is bound to the key-sequence in the keymap.

Only functions declared as `public` can be autoloaded.

3.4.9 Tutorial example

The example in this section explains how you can create the reference to the function `text-add-itemization.` The function is located in the file `ListItem.sel;` enter the following in the profile file `.slate_editor.init` (on one line) :

```
autoload (text-add-itemization, "ListItem.sel",
   default-keymap, "Esc-i");
```

Remember, the profile file is also a text-only file. You normally should edit this file using a text editor at the operating-system level. If you edit it within BBN/Slate, read and save it as a text file.

To create the reference to the function `search-word`, located in the file `SearchWord.sel`, enter the following in the profile file `.slate_editor.init` (on one line) :

```
   autoload (search-word, "SearchWord.sel",
      default-keymap, "Esc-s");
```

As a final test, quit the current Document Editor session, start up the Document Editor once more, and make sure that the key bindings correctly invoke the functions.

3.5 Application examples

This section contains several illustrations of function definitions. They vary from simple to more complex examples. These examples provide an overview of using the BBN/Slate extension language, and they can furnish insight into how you might approach similar tasks.

Remember that it is possible to invoke these or any programs you develop from a BBN/Slate menu. See Chapter 2 for a complete description of how you can configure your menuing system to provide access to the programs you create.

3.5.1 Inserting a signature file

If your site is equipped with a scanner, you can use it to create files containing the digitized image of the signature of each of your users. Once you have created the file, you can edit it in BBN/Slate using the **Add-Image-From File...** menu. For example, you may want to crop the image to include just the area immediately around the signature, and scale it to reduce its size. Once it is ready, you can save it in a special location, such as your home directory. For example, this example assumes that the image has been placed in a document named `.slate_sig.slt` and that it is located in `/home/smith`.

The short function below, named `sign-message`, places the image automatically at the end of your current file:

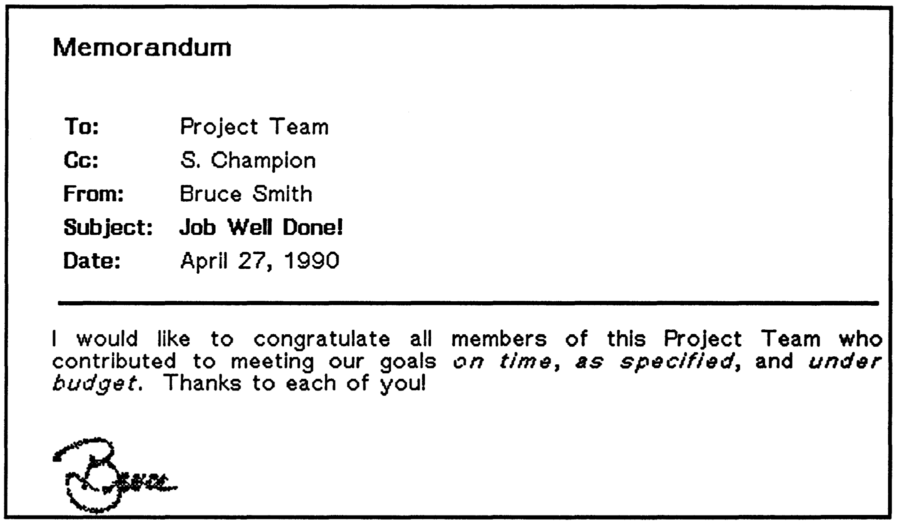
```
DEFINE void sign-message()
# "Sign a message with my digitized signature."
{
    text-end-of-buffer();
    insert-named-file("/home/smith/.slate_sig.slt");
}
```

You must then store the `sign-message` function in a file (remember that it must be a plain text file). This example assumes that you have placed it in a file named `sig.sel`. Note that it is also possible to store many different functions in one common file, perhaps giving it a general name such as `utilities.sel`.

Next, include a key binding in your profile file to associate the `sign-message` function with a key sequence. The `autoload` command binds it to the key sequence **Esc-Esc-S**.

```
# insert signature file in messages
autoload (sign-message, "sig.sel", text-keymap,
   "Esc-Esc-S");
```

Typically, you would invoke this function once you have completed writing a memo, as shown in the following example.


Memorandum

To:	Project Team
Cc:	S. Champion
From:	Bruce Smith
Subject:	**Job Well Done!**
Date:	April 27, 1990

I would like to congratulate all members of this Project Team who contributed to meeting our goals *on time*, *as specified*, and *under budget*. Thanks to each of you!

[illegible]

3.5.2 Loop for `execute-command-dialog`

This is a simple function that allows you to repeat the `execute-command-dialog` function (**Esc-x**) in a while loop, until you select **No** from the dialog box prompt. This allows you to enter `execute-command-dialog` just once, invoke `function-lookup`, and then browse through assorted functions until you are ready to exit. It makes evaluating potential functions for your BBN/Slate programs easier.

```
define public void function-lookup()
"Look at functions repeatedly; interrupt to exit loop."
{
    local s;
    while (1)
        {
        #call the routine
        execute-command-dialog();
        s = display-confirmation(
            "Execute another command? ");
        # if user cancels dialog, exit
        if (s == 0)
            return;
        }
}
```

3.5.3 Changing fonts

This example illustrates how you can redefine and later restore keymap settings and how you can use `invoke` to indirectly call a function. The function asks the user whether to use the roman, bold, italic, or bold italic font face. Then, it rebinds the key bindings for **LEFT hold** and **Esc-Esc-d**. The effect is that each time that the user selects a region of text with the mouse (with **LEFT hold**, dragging over the region, and **Release**), the selected region is changed to the font face chosen. To stop the process and restore the default key bindings, the user types **Esc-Esc-d**. This is a useful procedure for going back over an existing document and highlighting keywords and phrases.

```
local font_func;

define private void @apply-font()
{
    text-set-region();
    invoke font_func();
}

define private void @end-apply-font()
{
    bind(text-keymap, "LEFT-HOLD", text-set-region);
    bind(text-keymap, "ESC-ESC-d", unbound-error);
    bind(default-keymap, "LEFT-HOLD", text-set-region);
    bind(default-keymap, "ESC-ESC-d", unbound-error);
}

define public void apply-font-to-region()
local font;
{
    font = read-string-dialog(
        "Select font: r, b, i or bi.", "Font:");
    if (font == "r")
        font_func = text-romanize
    else if (font == "b")
        font_func = text-boldify
    else if (font == "i")
        font_func = text-italicize
    else if (font == "bi")
        font_func = text-bolditalicize
    else
    {
        display-message("Font face not known.");
        return;
    }
    bind(text-keymap, "LEFT-HOLD", @apply-font);
    bind(text-keymap, "ESC-ESC-d", @end-apply-font);
    bind(default-keymap, "LEFT-HOLD", @apply-font);
    bind(default-keymap, "ESC-ESC-d", @end-apply-font);
}
```

3.5.4 Screen capture

This example is a screen capture program that you can invoke from within the BBN/Slate Document Editor. It determines the windowing system in use and calls the correct screen capture utility. You then select a portion of the screen for capture, and that portion is inserted into the document at the current cursor position.

This example also illustrates the use of `invoke` for indirect function calls.

The most convenient method for users to access this program is to place an `autoload` staement in your public or private profile file. For example:

```
autoload(capture-screen, "snap.sel");
```

Then, add a menu item to an appropriate system menu file that executes `capture-screen`. For example:

```
ITEM Screen capture...
    HELP "Capture a portion of the screen and insert it
        at the current cursor position."
    INVOKES capture-screen
```

See Chapter 2 for details on adding new menu items.

```
DEFINE PUBLIC VOID capture-screen()
    "Capture a portion of the screen using mmsnap
     or xwd (as appropriate to the window system
     in use) and insert it at the current cursor
     position."
{
    local snap-buffer;
    local snap-command;
    local snap-prompt;
    local insert-function;
```

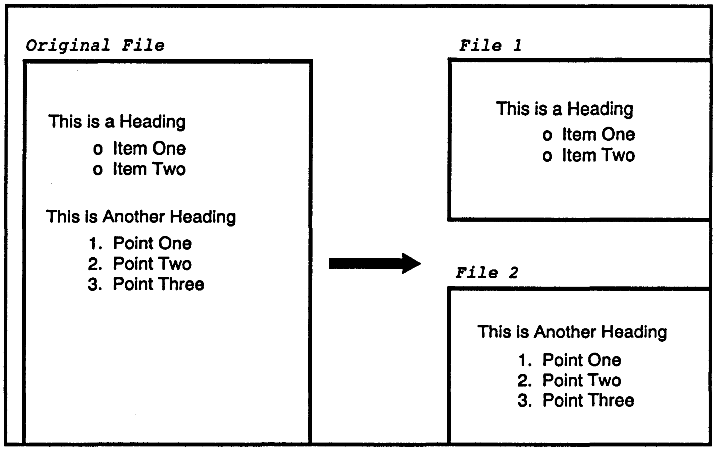
```
# Select a screen capture program and an object
# creation routine based on the window system
# in use.  We should really allow the user to
# set up default commands for SunView and X11,
# to take advantage of optional flags on the
# screen capture programs.  For instance, the
# MIT X11R4 xwd program has a "-frame" argument
# that includes the window manager's decorations
# on a window dump.  Not all versions of xwd
# support this, so we can't include it in a
# default program specification intended to work
# in all environments.  Per-user specifications
# for the screen capture programs would help here.

if (window-system-is-sunview) {
    snap-command = "mmsnap >/tmp/snap.out";
    snap-prompt = "Left Drag to select a region
       of the screen...";
    insert-function = add-named-raster;
}
else {
    snap-command = "xwd >/tmp/snap.out";
    snap-prompt = "Left Click to select a window...";
    insert-function = add-named-xwd;
}

# Create a buffer for process-run to use.  We also
# clear the buffer in case it contains any error
# messages from previous unsuccessful attempts to
# run capture-screen.

set-global-error-code(0);
snap-buffer = find-buffer("%screen-capture");
if (global-error)
    snap-buffer = add-empty-buffer(
        "%screen-capture", "");
clear-buffer(snap-buffer);

# Run the process, check the result, and remove
# the temp file. If we wanted to handle errors
# in a more sophisticated way, we we could try
# to grab the contents of the snap-buffer and
# display it in a dialog instead of just providing
# the user with the buffer name.
```

```
            display-string-in-status-line(snap-prompt);
            set-global-error-code(0);
            process-run(snap-buffer, snap-command);
            display-string-in-status-line("");
            if (global-error) {
                display-message("The screen capture operation
                   seems to have failed.  The buffer
                   named #screen-capture should contain
                   whatever output or error messages the
                   operation may have produced.");
                return;
            }
            invoke insert-function("/tmp/snap.out");
            process-run(snap-buffer, "/bin/rm -f /tmp/snap.out");
        }
```

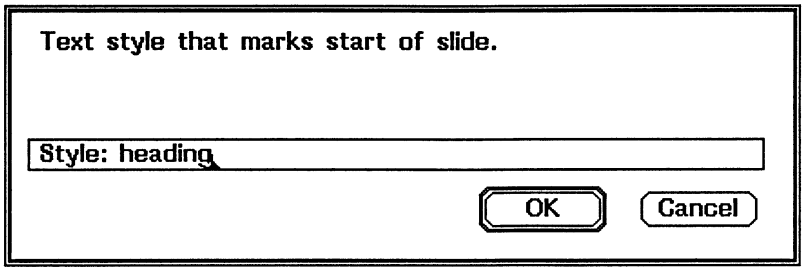
3.5.5 Slide maker

This example is a collection of three function definitions that form a slide maker program. That is, it breaks up one file into a series of smaller component files. For example, you can specify that each time the program encounters a `heading` style, a new *slide* (file) is created.

The program is applicable to files that consist of regularly demarcated material, such as a viewgraph presentation. The illustration below summarizes this process:

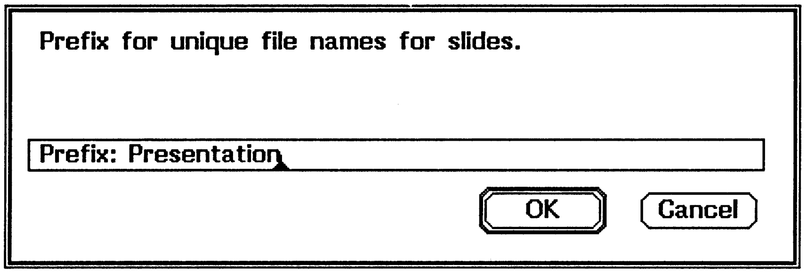


When you invoke the function `make-slides`, you are prompted for the text style, which serves as a dividing point for each file:


Text style that marks start of slide.

Style: heading

You are then prompted for a file prefix. If you enter **Presentation**, the first file becomes `Presentation1.slt`, the second file becomes `Presentation2.slt`, and so on:


Prefix for unique file names for slides.

Prefix: Presentation

The complete set of three function definitions is shown below.

```
# Break a file up into pieces and store each piece as
# a separate file with a name of the form prefixn,
# where n is an incrementing number. Each piece
# begins with a specified text style. The user is
# queried for the name of the text style and for the
# prefix for the file created.  The original file is
# not modified.

global prefix-name, n, done, TRUE, FALSE, next;


define void set-region-to-copy()
{
    local start;
    start = current-point();
    find-media-type-again();
    if (start == current-point())
    {
        done = TRUE;
        document-set-point(end-of-buffer());
    }
    text-set-mark();
    next = current-point();
    document-set-point(start);
}

define void copy-region-to-file()
{
    local this-buffer, s;
    this-buffer = current-buffer();
    copy();
    s = strcat(strcat(prefix-name, n), ".slt");
    n = n + 1;
    add-empty-buffer(s, "");
    change-to-named-buffer(s);
    paste();
    write-buffer-to-file(current-buffer(), s);
    mark-buffer-unmodified(current-buffer());
    set-buffer-of-pane(0, this-buffer);
}
```

```
define void make-slides()
{
    n = 1;
    TRUE = 1;
    FALSE = 0;
    done = FALSE;
    find-direction = 0;
    find-type = 0;
    text-find-name = read-string-dialog(
        "Text style that marks start of slide.",
        "Style:");
    prefix-name = read-string-dialog(
        "Prefix for unique file names for slides.",
        "Prefix:");
    delete-other-panes();
    document-set-point(beginning-of-buffer());
    while (!done)
    {
        set-region-to-copy();
        if (current-region() == 0)
        {
            display-message("No region selected to save."
            return;
        }
        copy-region-to-file();
        document-set-point(next);
    }
}
```

3.5.6 Search utility

This example, while only slightly extending the features available with the conventional search function of BBN/Slate, is instructive in its use of variables for defining and manipulating areas of a document. See Section 3.3 for more information on these special extension language functions (such as `current-point` and `end-of-buffer`).

The functions in this example query the user for a search string and then search the entire document from beginning to end for instances of that search string. When the string is found, the paragraph containing that string is displayed at the top of the display area and is highlighted. Two function keys are used in this example and are arbitrarily chosen to be F2 and F3 -- you may need to change them to reflect your system's use of function keys, the escape sequences they generate, and your preferences. F2 and F3 are bound to the functions `search-first` and `search-next`. The function `search-first` queries the user for a search string and searches for it at starting at the start of the document. The function `search-next` finds the next instance of the search string in the document.

```
local FALSE, TRUE, NULL, F2, F3;

local s_search_string; # the current search string

define private int find-word-in-region(search_string,
    search_region)

"Search forward for search_string in search_region
folding case (i.e. a lower case letter in the search
string matches either an upper or lower case letter
in the document). If the search is successful, then
display the start of the paragraph that contains the
search string at the top of the display, and
highlight the paragraph."
{
  # if no search region, return FALSE
  if (search_region == NULL)
      return(FALSE);
  if (!text-search(search_string, search_region,
         TRUE))
      return(FALSE);

  # save current cursor position
  text-save-excursion();

  # move cursor to start of paragraph
  text-start-of-paragraph();

  # display line containing cursor at top of display
  text-line-to-top();

  # restore cursor position
  text-restore-excursion();

  # select and highlight paragraph
  text-select-current-paragraph();
  return(TRUE));
}
```

```
define public void search-first()
"Query the user for a word or phrase to search for
and call find-word-in-region to search for it from
the start of the document. If the search string is
not found, then display an error message."
{
    local new_search_string, whole_document;
    new_search_string =
    read-string-dialog("Word or phrase to search for.",
                   "Word or phrase: ");

    # if user cancels dialog, do nothing
    if (global-error)
        return;

   # save the search string
    s_search_string = new_search_string;

    # if search string is not empty, do search
    if (s_search_string != "")
    {
        whole_document = create-region(
          beginning-of-buffer(), end-of-buffer());
        if (!find-word-in-region(s_search_string,
           whole_document))
           display-message(strcat(s_search_string,
               " not found."));
    }
}
```

```
define public void search-next()
"Search for the next instance of the search string in
the document. If the search string is not found, then
deselect any selected region."
{
  local region_to_search;
  # if cursor not set, then do nothing
  if (current-point() == NULL)
      return;
  region_to_search = create-region(
    current-point(), end-of-buffer());
  if (!find-word-in-region(s_search_string,
      region_to_search))
      # clear-region any previously selected region
      text-clear-mark();
}

F2 = "ESC-[225z";
F3 = "ESC-[226z";
FALSE = 0;
TRUE = 1;
NULL = 0;
s_search_string = "";

# bind F2 function key to search-first
bind(text-keymap, F2, search-first);

# bind F2 function key to search-next
bind(text-keymap, F3, search-next);
```

3.5.7 Mail merge

This example is a mail merge facility. It is the largest and most complex of the sample programs presented here. The initial six functions are declared as `private` (because users do not need to access them). Only the final function, `merge`, is declared as `public` – this is the function name users will invoke.

There are comments throughout to assist you in understanding the code. The initial comments describe how the program works.

```
#################################################################
#
#  merge(): a mail merge facility
#
#       This package defines a single public routine, merge.
#
#       The merge routine:
#         1)    Prompts for a file of addresses.  The file should
#               be a Slate document with sets of names and
#               addresses, each separated by blank lines.
#         2)    Prompts for a letter template.
#         3)    Prompts for a place to insert the salutation and
#               a place to insert the address into the letter.
#         4)    Generates the merged files with the same base
#               name as the template but with numbers appended.
#               For example, if the template is called form-letter,
#               the generated files will be form-letter-1.slt,
#               form-letter-2.slt, etc.
#
#################################################################

#
# Define some global values
#
global TRUE;
TRUE = 1;
global FALSE;
FALSE = 0;
```

```
#
# Special keys used below
#
local control-g;
local control-left-click;
control-g               = string-to-event-code("^G");
control-left-click      = string-to-event-code("CTRL-LEFT-CLICK");

#
#  These are local variables used within this set of routines.
#  We prefix all these variables with s_ to make it clear that
#  they are local variables.
#
local s_address_file;      # name of address list file
local s_letter_file;       # name of letter template file
local s_address_curpoint;  # saved point in address file
local s_files_written;     # number of letters written
local s_eof;               # end of address list read
local s_newfile;           # initial substring of new file
local s_filename;          # name of file being created
local s_paste_1_mark       # place to paste salutation
local s_paste_2_mark       # place to paste address

#
#  For development purposes, we undefine these functions
#  at the beginning of the file so we can reload new
#  versions without getting any error messages during
#  the development process.
#  These lines can be removed once these routines are stable
#  and will only be loaded once per editing session.
#
undefine        merge-mark-distance,
                merge-is-blank-line,
                merge-get-point-from-user,
                merge-copy-line-group,
                merge-cleanup,
                merge-worker,
                merge
```

```
#################################################################
#
#  merge-mark-distance(): Return the distance between two marks.
#       We do this by converting them to document locations
#       (which are simple integers) and then subtracting.
#
#################################################################

define private int merge-mark-distance(m1, m2)
{
    local l1, l2;

    l1 = mark-convert-to-location(m1);
    l2 = mark-convert-to-location(m2);
    return (l2 - l1);
}

#################################################################
#
#  merge-is-blank-line(): This routine returns TRUE if the
#       point is in a blank line, FALSE otherwise.  An empty line
#       or a line that contains only whitespace is considered to
#       be blank.  This routine assumes the point is at the
#       start of the line.
#       Note, however, a blank line is *not* the whitespace
#       in between two paragraphs (one could argue that this
#       mail merge package should treat it as an empty line,
#       but this routine will not recognize it).
#
#################################################################
```

```
define private int merge-is-blank-line()
{
    local sol, eol, nchars, i;

    #
    # Use save and restore excursion to leave the point at the
    # same place as where you started.
    #
    text-save-excursion();
    sol = current-point();
    text-end-of-line();
    eol = current-point();
    document-set-point(sol);
    nchars = merge-mark-distance(sol, eol);
    i = 0;
    while (i < nchars)
    {
        if (!text-is-whitespace(i))
        {
            text-restore-excursion();
            return(FALSE);
        }
        i = i + 1;
    }
    text-restore-excursion();
    return(TRUE);
}

################################################################
#
#  merge-get-point-from-user(): Allow the user to scroll through
#       the document to identify the position to insert the
#       salutation, address, etc.
#       Actually, since event-process-current() is used, the
#       user can really do anything until s/he confirms.
#       Ctrl-left-click will confirm the point, Control-G
#       will interrupt the process.
#       This routine returns TRUE if the user confirmed, FALSE
#       otherwise.
#
################################################################
```

```
define private int merge-get-point-from-user(msg)
{
    local s, code, buffer;

    buffer = current-buffer();
    s = strcat(msg, " Ctrl-Left-Mouse to confirm, ^G to stop.");
    display-string-in-status-line(s);
    command-loop-start();
    while (TRUE)
    {
        code = event-wait-for-next();
        if (code == control-left-click)
        {
            if (buffer != current-buffer())
            {
                display-message("You must be in the template file.");
                display-string-in-status-line(s);
            }
            else
            {
                display-string-in-status-line("");
                return (TRUE);
            }
        }
        else if (code == control-g)
        {
            if (buffer != current-buffer())
            {
                display-message("You must be in the template file.");
                display-string-in-status-line(s);
            }
            else
            {
                display-string-in-status-line("");
                return (FALSE);
            }
        }
        else
            event-process-current();
    }
}
```

```
#################################################################
#
#  merge-cleanup(): Delete back size chars from the mark specified.
#
#################################################################

define private void merge-cleanup(mark, size)
{
    document-set-point(mark);
    text-set-mark();
    document-set-point(mark-add-offset(mark, -size));
    text-cut-region();
}


#################################################################
#
#  merge-copy-line-group(): Copy the next group of lines
#       from the address file.
#
#################################################################

define private void merge-copy-line-group()
{
    local start, nlines;

    #
    #  Go back to the address file and to our last location.
    #
    visit-document(s_address_file);
    document-set-point(s_address_curpoint);
    #
    #  Skip over blank lines.  Note that text-down-line will
    #  set the global-error flag if we are at the end of the
    #  document.
    #
    while (merge-is-blank-line() && !global-error)
        text-down-line();
    if (global-error)
    {
        s_eof = TRUE;
        set-global-error-code(0);       # RESET!
        return;
    }

    #
    #  Skip down to a blank line or the end of the document.
    #  Make the non-blank lines be the selected region.
    #
    start = current-point();
    nlines = 0;
    while (!merge-is-blank-line() && !global-error)
    {
        text-down-line();
        if (!global-error)
            nlines = nlines + 1;
    }
    set-global-error-code(0);           # RESET!
    if (nlines == 0)
    {
        s_eof = TRUE;
        return;
    }
    text-end-of-line();
    s_address_curpoint = current-point();       # ...for next time around
    text-set-mark();
    document-set-point(start);

    #
    #  Copy those lines to the clipboard.
    #
    copy();
}
```

```
################################################################
#
#  merge-worker(): Do the actual merge.  Copy the
#       salutation from the address list and paste it
#       in and then copy the address and paste it.
#
################################################################
```

```
define private void merge-worker()
{
    local start, end, size1, size2;

    # Get salutation
    merge-copy-line-group();
    # Check if we're done
    if (s_eof)
        return;
    # Now paste it in
    visit-document(s_letter_file);
    if (global-error) return;
    document-set-point(s_paste_1_mark);
    start = mark-convert-to-location(current-point());
    paste();                        # current-point now at end of region
    text-delete-back();                     # get rid of extra newline
    end = mark-convert-to-location(current-point());
    size1 = end - start;

    # Get address
    merge-copy-line-group();
    # This is really an error...
    if (s_eof)
        return;
    # Now paste it in
    visit-document(s_letter_file);
    document-set-point(s_paste_2_mark);
    start = mark-convert-to-location(current-point());
    paste();                        # current-point now at end of region
    text-delete-back();                     # get rid of extra newline
    end = mark-convert-to-location(current-point());
    size2 = end - start;

    # Save the document
    s_filename = strcat(strcat(s_newfile, s_files_written+1), ".slt");
    write-buffer-to-file(current-buffer(), s_filename);
    if (!global-error)
        s_files_written = s_files_written + 1;

    # Clean up pasted-in region
    merge-cleanup(s_paste_1_mark, size1);
    merge-cleanup(s_paste_2_mark, size2);
}
```

```
##################################################################
#
#  merge(): This is the main routine for the mail merge facility.
#       This routine:
#         1)    Prompts for a file of addresses.  The file should
#               be a Slate document with sets of user names and
#               addresses, separated by blank lines.
#         2)    Prompts for a letter template.
#         3)    Prompts for a place to insert the salutation and
#               a place to insert the address.
#         4)    Generates the merged files with the same base
#               name as the template but with numbers appended.
#
##################################################################
```

```
define public void merge()
{
    # Get the file of addresses
    s_address_file = read-string-dialog("Address file:", "File name:");
    if (global-error)
        return;
    visit-document(s_address_file);
    if (global-error)
    {
        display-message("Merge failed:  Can't find address file.");
        return;
    }
    document-set-point(beginning-of-buffer());
    s_address_curpoint = current-point();

    # Get the letter template file
    s_letter_file = read-string-dialog("Letter template file:", "File name:"
    if (global-error)
        return;
    visit-document(s_letter_file);
    if (global-error)
    {
        display-message("Merge failed: Can't find letter template file.");
        return;
    }
    s_files_written = 0;

    s_newfile = strcat(s_letter_file, "-");

    # Get the salutation and address point
    if (!merge-get-point-from-user("Specify salutation point."))
        return;
    s_paste_1_mark = current-point();
    if (!merge-get-point-from-user("Specify address point."))
        return;
    s_paste_2_mark = current-point();

    s_eof = FALSE;
    s_files_written = 0;
    while (!s_eof && !global-error)
        merge-worker();
    display-message(strcat("Files written: ", s_files_written));
}
```

3.6 Token syntax

This section lists the token syntax of the BBN/Slate extension language. It can be used as an aid in understanding the exact structure of the language.

3.6.1 Statements

```
stmt:
    if_stmt |
    while_stmt |
    func_call |
    assign_stmt |
    func_def |
    multi_stmt |
    return_stmt |
    local_stmt |
    global_stmt |
    undef_stmt |
    autoload_stmt |
    ind_func_call

    stmtlist:       stmtlist stmt | /* empty */

    multi_stmt:     LBRACE_TK stmtlist RBRACE_TK

    autoload_stmt:  AUTOLOAD_TK LPAREN_TK SYMBOL_TK SYMBOL_TK RPAREN_TK |
                    AUTOLOAD_TK LPAREN_TK SYMBOL_TK SYMBOL_TK SYMBOL_TK
                       SYMBOL_TK RPAREN_TK

    assign_stmt:    SYMBOL_TK ASSIGN_TK expr

    func_call:      SYMBOL_TK param_list

    ind_func_call:  INVOKE_TK SYMBOL_TK param_list

    if_stmt:        IF_TK LPAREN_TK expr RPAREN_TK stmt |
                    IF_TK LPAREN_TK expr RPAREN_TK stmt ELSE_TK stmt
```

```
while_stmt:      WHILE_TK LPAREN_TK expr RPAREN_TK stmt

return_stmt:     RETURN_TK LPAREN_TK expr RPAREN_TK |
                 RETURN_TK

symbol_list:     /* empty */ | symbol_list1

symbol_list1:    SYMBOL_TK |
                 symbol_list1 COMMA_TK SYMBOL_TK

global_stmt:     GLOBAL_TK symbol_list

param_list:      LPAREN_TK expr RPAREN_TK |
                 LPAREN_TK RPAREN_TK

local_stmt:      LOCAL_TK symbol_list

undef_stmt:      UNDEFINE_TK symbol_list

arg_list:        LPAREN_TK symbol_list RPAREN_TK
```

3.6.2 Functions

```
func_type:
    /* empty */ | PRIVATE_TK | PUBLIC_TK

func_def:
    DEFINE_TK func_type SYMBOL_TK SYMBOL_TK arg_list |

    DEFINE_TK func_type SYMBOL_TK SYMBOL_TK arg_list multi_stmt |

    DEFINE_TK func_type SYMBOL_TK SYMBOL_TK arg_list STRING_TK |

    DEFINE_TK func_type SYMBOL_TK SYMBOL_TK arg_list STRING_TK multi_stmt
```

3.6.3 Expressions

```
expr:
    expr EQ_TK expr |
    expr NE_TK expr |
    expr GE_TK expr |
    expr LE_TK expr |
    expr GT_TK expr |
    expr LT_TK expr |
    expr MUL_TK expr |
    expr DIV_TK expr |
    expr ADD_TK expr |
    expr SUB_TK expr |
    expr OR_TK expr |
    expr AND_TK expr |
    expr COMMA_TK expr |
    NOT_TK expr |
    SUB_TK expr |
    LPAREN_TK expr RPAREN_TK |
    INT_TK |
    REAL_TK |
    SYMBOL_TK |
    STRING_TK |
    func_call |
    ind_func_call |
    NULL_TK

    INT_TK:         One or more unsigned digits.
    REAL_TK:        One or more unsigned digits including a decimal point
    SYMBOL_TK:      lower case letters plus the characters - . @ and _.
                    The dash and period cannot be used as the first
                    character.
    STRING_TK:      Zero or more characters bounded by double quotes.
                    A double quote may be included by preceding it with
                    a backslash.
```

3.6.4 Reserved words

DEFINE_TK:	define
UNDEF_TK:	undefine
GLOBAL_TK:	global
LOCAL_TK:	local
NULL_TK:	null
INVOKE_TK:	invoke
IF_TK:	if
ELSE_TK:	else
WHILE_TK:	while
RETURN_TK:	return
AUTOLOAD_TK:	autoload

3.6.5 Expression operators

ASSIGN_TK:	=
NOT_TK:	!
EQ_TK:	==
NE_TK:	!=
GE_TK:	>=
LE_TK:	<=
GT_TK:	>
LT_TK:	<
MUL_TK:	*
DIV_TK:	/
ADD_TK:	+
SUB_TK:	-
AND_TK:	&&
OR_TK:	||

3.6.6 Other tokens

COMMA_TK:	,
LPAREN_TK:	(
RPAREN_TK:	)
LBRACE_TK:	{
RBRACE_TK:	}

3.6.7 Comments

Text from a # to the end of a line is ignored. A # may be included in a string, and in that case, it is not interpreted as a comment. Semicolons may be used at the end of statements. They are currently ignored.

A Customizing Variables

This appendix describes the variables that may be set to control the behavior of BBN/Slate. There are two basic categories of BBN/Slate variables: *document manager variables* and *document editor variables*. The document manager variables control the appearance and behavior of the Document Manager. The document editor variables control the appearance and behavior of the Document Editor.

The document editor variables can be further separated into three categories:

- *Editor variables*. These variables control the operation of the Document Editor, and should be set in the `.slate_editor.init` profile file. An example of this type of variable is `classify-names`, which controls the classification markings used to label document elements.
- *Document variables*. These variables control the attributes of a document that should persist from one editing session to another, and are set by defining document attributes in the Document Editor. An example of this type of variable is `page-footer`, which you set in the **Preview/Print...** dialog box of the **Print** menu.
- *Buffer variables*. These variables control the attributes of a document that should persist during an editing session, but should not persist across editing sessions. These are internal variables set and used the Document Editor.

All variables are stored in one of three *variables tables*, depending upon the scope of the variable. Variables associated with the current buffer are stored in the *buffer variables table.* Variables associated with the document are stored in the *document variables table.* Variables associated with the Document Editor in general are stored in the *global variables table* (for example, `page-left-pixels`, which controls the margin between the left edge of the Document Editor window and the beginning of the text area, is a global variable).

Setting variables

In addition to setting variables in your `.slate_editor.init` profile file, you can specify a new value for a variable interactively. You can do this in several ways:

- Choose **Set Variable** from the BBN menu item (located at the left side of the main menu).
- Enter `execute-extended-command` (for which the default key binding is **Esc-x**) and select set from the scroll box. You can also select `set-buffer-variable-dialog`, `set-document-variable-dialog`, or `set-global-variable-dialog` to review and reset specific variables in the variables tables.

If the variable you choose has a current value, the dialog box will display it. If you want, you can simply exit from the dialog box without changing the value. This is useful for reviewing or noting the value of the variables in case you want to revert to a previous value. The values you assign in this interactive manner last for the duration of your editing session. To make variable assignments permanent, you should set them in your profile file.

Organization of this appendix

This appendix sorts the variables into the two basic categories of Document Manager variables (of which there are only a few) and Document Editor variables. It further groups the Document Editor variables by function: top-level Document Editor variables, text variables, graphics variables, and so on (note that there are no variables associated with conferencing). This organization helps you to associate each variable with the context in which you use it.

A.1 Document Manager variables

The Document Manager reads a file named `.slate_tool.init` from your home directory to customize its operation. This file may include any of the variables included in this section.

editor-args

A string that the Document Manager passes on startup to the Document Editor. Since the Document Editor obeys the standard operating-system flags to set its window size and position, you can use this variable to control the initial position and size of the Document Editor. For example:

```
editor-args = "-Wp 550 0 -Ws 600 900 -WP 1088 75";
```

display-document-size

If the value of this variable is 1, then the size of a document is listed at the end of the document summary line. The default assignment is:

```
display-document-size = 0;
```

default-menu-font

You can control the default font used for menus in the Document Manager by setting this variable to the name of the font. The default assignment is:

```
default-menu-font = "helvetica10b";
```

A.2 Document Editor variables

A.2.1 Top-level Document Editor variables

auto-backup	Controls whether a file `filename.BAK` is written the first time a file is saved. This backup file saves the initial version of the file at the start of the editing session. It may be overwritten during subsequent editing sessions.
checkpoint-frequency	Controls how often checkpoint files are written out. Default value is 200.
classify	An integer which gives the current top classification level for all objects in the document. This variable is only guaranteed to be valid after the document is written or printed.
classify-names	The classification names. The value of this variable should be a string of semicolon separated pairs. Each pair is a classification level from lowest to highest. The elements of the pair are separated by a comma. The first element of the pair is the long form of the classification level; the second element is the short form. The default assignment is:

```
classify-names =  "(Unclassified), (U) ; (Confidential),
   (C) ; (Secret), (S) ; (Top Secret), (TS) ;"
```

default-classification	The default classification level for new objects. The value of this variable should be the long form of one of the classification levels, e.g. "Secret".
default-document-template	The default document template to use for initializing new buffers. If the name is an absolute path name, that file is used. If it begins with a plus sign (+), it is looked up as a file in a folder. Otherwise, the user and then system template directories are searched. If this variable is unset, the editor will look for a file of the name `.slate_template` in the current directory, user's home directory, or the Slate system data directory, in that order.

default-font	The default font for new documents if no document template is explicitely set. If a document template is set (using `default-document- template`), this variable has no effect.
default-keymap	The top-level keymap that is used when outside any media editor.
default-menu-font	Font used for menus in the editor.
display-classify-menu	If non-zero, the classification menu choice is displayed in the top-level editor menu.
default-ui-font	The font used for buffer status lines and the editor message area.
find-direction	Direction in which to search to find an instance of a media type or text style. Values are 0 (forward), 1 (reverse), and 2 (anywhere). The default is forward.
find-type	Type of media object or text style to search for. Values are: 0 (text), 1 (image), 2 (speech), 3 (sheet), 4 (graphics), 5 (headers), and 6 (raster). The default is text.
global-error	An error code that is set by the editor functions.
global-last-function	The name of the last editdoc or user defined function executed.
global-repeat-count	The number of times to repeat an operation.
global-linestyle-table	The global line style table.
library-search-path	Path names separated by colons to search for `autoload` files.
menubar-suppress	If `menubar-suppress` is set to non-zero, the pull-down menus across the top of the editor window will not be shown. If you

	suppress display of the menu bar, the only way to access the top level editor menu is by using **Control-Right-Hold**. This variable is only effective when set in the `.slate_editor.init` profile file -- changing it while the editor is running has no effect.
menubar-distribute-evenly	If set to non-zero, the pull-down menubar items will be evenly distributed across the top menubar. Otherwise, the items will be left justified.
menubar-icon-file	The file name where icon for menubar is found.
menu-directory	The directory in which to search for menu files. If this variable is set, the editor will look in the specified directory when loading menu definition files, and use the menu definitions there to modify the default menus. This variable only takes effect when set from the `.slate_editor.init` file; changing its value interactively has no effect.
page	The current page while printing. This variable may be modified in the Print Values attribute of a text style in order to set the page number during printing.
page-even-footer	The text to place at the bottom of even pages. Tildes are used to separate the text for the left side, center, and right side of the page. A "$" indicates the end of a line (for multi-line text). The characters "@1" are replaced with the current page number. The default is the empty string. This variable may be modified in the Print Values attribute of a text style in order to change the even footer style during printing.
page-even-header	The text to place at the top of even pages.
page-footer	The text to place at the bottom of even and odd pages.
page-header	The text to place at the top of even and odd pages.

page-left-pixels	The number of pixels of white space at the left of a document when it is displayed. The default is 12.
page-odd-footer	The text to place at the bottom of odd pages.
page-odd-header	The text to place at the top of odd pages.
page-scroll-percentage	The percentage of the new page (height or width) to show when scrolling by pages. The default is .8 to show 80% of the new page.
page-top-pixels	A buffer-specific variable. The number of pixels of white space at the top of a document when it is displayed. The default is 12.
page-width	A document-specific variable. The width of a page displayed on the screen. The default is 6.5i.
quick-menus	This variable determines whether Slate uses the full set of command menus, or an abbreviated set of menus that includes only the most commonly used subset of editor commands. This variable only takes effect when set from the `.slate_editor.init` file; changing its value interactively has no effect.
read-only	A buffer specific variable. If non-zero, the buffer cannot be modified; if 0, the buffer can be modified. The default is to allow the buffer to be modified.
redefine-silently	If non-zero, the user can redefine user-defined functions in the extension language without warning. If 0, the user is asked to confirm the redefinition of a function. The default is to ask the user to confirm redefinitions.
spool	If non-zero, send the document to a printer rather than a file. The default is to send the document to a printer.

suppress-startup-screen	If non-zero, then don't display the logo; if 0, then display the logo. The logo is displayed momentarily at startup and covers the entire editor window. The default is to display the logo.
use-color	If non-zero, use color; if 0, do not use color. The default is to not use color.
visit-splits-pane	If non-zero, the visit-file command (invoked using the menu commands **File-Read-In New Buffer** or the default key binding **^X^V**) splits the existing pane and displays the file in the second pane. If 0, the visit-file command reads a new file into the existing pane. The default is to read a new file into the existing pane.
window-system-name	The name of the window system in use (either `sunview` or `x11`). The default setting is `sunview`.
window-system-is-sunview	If non-zero, the window system in use is SunView™. If 0, the window system in use is X11.
window-system-is-x11	If non-zero, the window system in use is X11. If 0, the window system in use is SunView.
write-current-version	If non-zero, the editor uses the version 3.0 format for sending documents. If 0, the current version format is used. The default is to use the current version format.

A.2.2 Text variables

text-default-format The format used for inserting text into a document when no specific format is requested. Default is:

```
text-default-format = "paragraph"
```

text-autotag-content	If 1, the three letters "tag" appear in the tag field of a description list as it is created. If 0, no text appears. The default assignment is: `text-autotag-content = 1;`
text-keymap	The keymap for the text media type.
text-find-name	The name of the text style to find. The default is the empty string.
text-plain-linewidth	The default width of a line when writing a multimedia document as text. The default is 79 characters.

A.2.3 Graphics variables

additional-textures	Additional textures to enter in the global texture table.
global-bg-texture	Index in global texture table for the default background texture for images.
global-color-texture-table	The global color texture table.
global-fill-texture	Index in global texture table for the default fill texture for graphics and images.
global-pen-texture	Index in global texture table for the default pen texture for graphics and images.
global-texture-table	The global texture table.
graphics-keymap	The keymap for the graphics media type.
graphics-text-justification	Justification to use when adding text to a graphics object. Valid values: left, center, right.

graphics-show-grid	If set to 1, show the grid when a new graphics object is created. If set to 0, do not show it.
graphics-align-to-grid	If set to 1, grid mode (align objects to grid) is ON. If set to 0, it is OFF.
graphics-show-size	If set to 1, show the size when reshaping. If set to 0, do not show it.
graphics-show-rulers	If set to 1, show the horizontal and vertical rulers when a new graphics object is created. If set to 0, do not show them.
graphics-show-mouse-position	If set to 1, show the mouse position in rulers (when visible). If set to 0, do not show position.
graphics-grid-size	Size to make grid when new graphics object is created.
graphics-font	The font to use for text in a graphics object. The default is the default document font.

A.2.4 Image variables

editor-colormap-size

Sets the number of colors BBN/Slate will use when displaying a color rasterfile. If a color image has more than this number of colors, BBN/Slate will discard the least frequently used colors and replace them by other, similar colors.

The default is to use no more than 128 colors, leaving the other 128 colors for use by other color applications running on the same workstation display. If you are not using any other color applications, you may want to increase the editor's colormap size to 256, by including a line such as:

```
editor-colormap-size = 256;
```

in your `.slate_editor.init` file.

The editor's colormap is created the first time a color image is displayed. Once the colormap has been created, changes in `editor-colormap-size` are ignored.

global-brush-texture Index in global texture table for the default brush texture for images.

image-keymap The keymap for the image media type.

picture-keymap The keymap for the picture media type.

A.2.5 Spreadsheet variables

spreadsheet-font Sets the default font for the spreadsheet editor. For example,

```
spreadsheet-font = "helvetica12"
```

sets the default font for new spreadsheets to a 12-point Helvetica font. The font may be changed for each spreadsheet by selecting the spreadsheet, then choosing **Font** from the spreadsheet menu.

A.2.6 Speech (audio) variables

audio-caption-position Indicates the default positioning for the caption of new speech elements. Legal values are `"above"`, `"below"`, `"left"`, and `"right"`. The default assignment is:

```
speech-caption-position = "below"
```

audio-caption-justification A string that indicates how the caption is to be justified with respect to the icons for new speech elements. The editor only uses this variable when the caption is above or below the icon. Legal values are `"left"`, `"right"`, and `"center"`. The default assignment is:

```
audio-caption-justification = "center"
```

audio-caption-font	A string that indicates the font to be used for captions on new speech elements. You may use a BBN/Slate font name. For example: `audio-caption-font = "helvetica10b"` The default value is the same as the default font for the document.
audio-caption-text	The text that labels each new speech element. You can embed various codes in this string to insert information such as your name, the date, the time, etc. The codes that may be included in this string are:
@logname	Your operating system login name.
@fullname	Your full name, as it appears in the operating system password file.
@hostname	The short host name (without domain qualifiers) of the host where you are logged in.
@Hostname	The full host name (including domain qualifiers) of the host where you are logged in.
@date	The current date in the form "3 Jun 1990".
@date(format)	A date whose format is determined by the date formatting codes in the **format** string. These codes are the same codes used in headers and footers. The default assignment for this string is:

```
speech-caption-text = "Remark by @fullname on
    @date(%D, %G:%M %A)"
```

The length of each speech passage in seconds is automatically appended to its caption string.

audio-channels	Number of audio channels to record. Monophonic is 1 channel and stereophonic is 2. Not all devices support stereo.

audio-input	The input source for recording speech. The value is a string whose interpretation is device-specific. For instance, for a device that can record directly or take input from another audio source, the legal values might be "microphone" or "aux". Using the value "default" will select some default audio input source appropriate to the device in use.
audio-output	The output device for playing speech. The value is a string whose interpretation is device-specific. For instance, for a device that can play to either an internal speaker or an external headphone jack, the legal values might be "speaker" or "headphones". Using the value "default" will select some default output appropriate to the audio device in use.
audio-playback-level	The relative volume at which to play speech back. The value is usually an integer between 0 and 100, and specifies the percentage of the audio device's maximum playback level to use when playing speech. Setting `audio-playback-level` to a negative value uses a default playback level appropriate to the device. The actual effect of any given value will be device-specific. Some devices may not provide control over the playback level, in which case setting the variable will have no effect at all.
audio-record-level	The relative volume at which to record speech. The value is usually an integer between 0 and 100, and specifies the percentage of the audio device's maximum recording level to use when recording speech. Setting audio-record-level to a negative value uses a default recording level appropriate to the device. The actual effect of any given value will be device-specific. Some devices may not provide control over the recording level, in which case setting the variable will have no effect at all.
audio-sampling-rate	Number of audio samples per second for each channel. This variable is only used when creating an audio object.
vocoder	The hostname of the voice annotation server for this editor to use.

A.2.7 Enclosure variables

enclosure-internal-top-padding	The amount of whitespace to leave between the top of an enclosure's icon or caption and its bounding box.
enclosure-internal-left-padding	The amount of whitespace to leave between the left edge of an enclosure's icon its bounding box.
enclosure-internal-bottom-padding	The amount of whitespace to leave between the bottom of an enclosure's icon or caption and its bounding box.
enclosure-internal-right-padding	The amount of whitespace to leave between the right edge of an enclosure's icon or caption and its bounding box.
enclosure-external-top-padding	The amount of whitespace to leave above an enclosure in the document.
enclosure-external-left-padding	The amount of whitespace to leave to the left of an enclosure in the document.
enclosure-external-bottom-padding	The amount of whitespace to leave below an enclosure in the document.
enclosure-external-right-padding	The amount of whitespace to leave to the right of an enclosure in the document.
enclosure-ignore-exitval	If the value of `enclosure-ignore-exitval` is nonzero, the enclosure media type will ignore non-zero termination codes by enclosure commands. Otherwise, non-zero termination codes are assumed to imply an error in the execution of the command and an error message is presented.
enclosure-temp-directory	Directory to use when creating temp files for editing copies of enclosed data. If undefined, use "/tmp".
enclosure-font	The font to use for enclosure captions. The default is to use the default document font when the enclosure is created.

enclosure-icon-file	The name of a file containing that icon to use for newly created enclosures. The data in the file should be in the format produced by "iconedit". If the variable is unset, a default icon is used.
enclosure-icon-width	If the `enclosure-icon-file` variable is set, this variable tells how many pixes of icon data to use in the enclosure's icon. Setting its value to 0 means use the entire width of the icon data.
enclosure-icon-height	If the `enclosure-icon-file` variable is set, this variable tells how many pixels of icon data to use in the enclosure's icon. Setting its value to 0 means use the entire height of the icon data.
enclosure-copy-data	If set to a non-zero value, the default for new enclosures is to use enclose-by-copying mode. If set to 0, the default is to use enclose-by-reference mode.
enclosure-show-size	If set to a non-zero value, the default for new enclosures is to set the show-size attribute. If set to 0, the default is not to show the size.
enclosure-copy-when-mailed	If set to a non-zero value, the default for new enclosures that are references is to set the copy-when-mailed attribute. If set to 0, the default is not to set copy enclosure data when mailed.

A.2.8 Printing variables

print-bottom-margin	The margin to reserve at the bottom of the page when printing. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is .75 inches.
print-gutter	The horizontal space between columns when printing multiple columns on a single page. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is ".5i".

print-header-font	The font to use for headers and footers. The value of this variable is an index into the font delta table, so it should be changed with care.
print-headers	If this is non-zero, print headers and if this is 0, do not print headers. The default is to print headers.
print-landscape	If this is non-zero, print in landscape mode (horizontally) on the page and if this is 0, print vertically on the page. The default is to print vertically on the page.
print-left-margin	The margin to reserve at the left edge of the page when printing. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is ".75i".
print-numcolumns	The number of columns to print. The default is 1.
print-oddeven	If this is non-zero, print odd and even pages with different headers and footers. If this is 0, then the values for even headers and footers are used for both odd and even pages. The default is to use the even headers and footers for both odd and even pages.
print-pageheight	The height of the page before margins are subtracted. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is "11i".
print-pagewidth	The width of the page before margins are subtracted. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is "8.5i".
print-right-margin	The margin to reserve at the right edge of the page when printing. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is ".75i".

print-spooler

The command line used to print the document. The program invoked is assumed to read PostScript® on its standard input and generate a printed document as its output. In the command line, the notation `%s` is converted to the name of the document. The default is "`lpr %s`".

print-startpage

The page number for the first page of the document. The default is 1.

print-titlepages

The number of pages that should be printed at the start of the document before page numbering begins. The default is 0.

print-toc

If this is non-zero, print the table of contents and if this is 0, do not print the table of contents. The default is to not print the table of contents.

print-top-margin

The margin to reserve at the top of the page when printing. Units are: i (inches), l (lines), c (centimeters), s (spaces in the default font), and p (points - 1/72 of an inch). The default is ".75i".

printfile

A buffer specific variable. If the spool variable is 0, this variable specifies the name of the file in which the print command will place its output. The default assignment is:

```
printfile = "printfile.ps";
```

A.2.9 Mail variables

Header variables

header-display-fields

The set of header fields displayed when you add new headers to a document. List the names of the fields, separated by spaces. The default assignment is:

```
header-display = "to subject";
```

The header editing dialog, however, will *always* contain the three standard headers, **To**, **Subject**, and **Cc**.

header-edit-fields	An additional set of header field names, separated by spaces, that appear when you add headers to a document. The default **To**, **Subject**, and **Cc** fields always appear. You can use this variable to specify additional headers for your messages. For example, if you always want to be prompted for fields named **Bcc** and **Priority**, you could include: `header-edit-fields = "Bcc Priority";` in your `.slate_editor.init` profile file.
header-keymap	The keymap for the header media type.
header-name-font	The font used to display the header field names. It should be of the form: `header-name-font = "helvetica10b";`
header-nodisplay-fields	The set of header fields you do not want displayed in the editor. List the names of the fields, separated by spaces. For example: `header-nodisplay-fields = "received message-id";`
header-noedit-fields	The set of header fields that cannot be edited. When you edit an existing set of document headers, these fields will not be displayed in the editing dialog. List the field names, separated by spaces. The default assignment is: `header-noedit-fields = "date from";` The header editing dialog will *always* contain the three standard headers, **To**, **Subject**, and **Cc**.

header-value-font The font used to display the header field values. It should be of the form:

```
header-value-font = "helvetica10";
```

mail-record The name of a folder to place copies of all out-going mail, e.g., `"+outbox"`. If this variable is unset, no copy of outgoing mail is made.

Mail variables for the system administrator

In addition to the variables that control display and editing of document headers, a number of variables control the encoding and sending of multimedia mail. These variables are normally set by the system administrator when BBN/Slate is first installed. These variables enable you to configure BBN/Slate so that it can send multimedia mail using mail transport systems not directly supported by the installation program described in the *Installation Instructions.*

If you use Sendmail or MMDF as your mail transport system, you probably do not need to be familiar with the variables in this section; the BBN/Slate installation program knows how to configure the system correctly for both Sendmail and MMDF. If you use some *other* maii transport system, you must use these variables to configure the editor to send mail correctly. You will also need to refer to *System Topics* to learn how to arrange for multimedia mail to be received from other BBN/Slate users properly.

Encoding and delivery of multimedia mail are controlled by the following editor configuration variables. The default values have been chosen so that they are suitable for use with `/usr/lib/sendmail` as the mail delivery agent.

fax-mail-address The address of the fax spooler mailbox.

mail-content-name	A string indicating the name of the field used to encode the message content type.

This variable is only used if mail-content-type is not an empty string. It sets the *name* of the field whose value will be the value of `mail-content-type`. The default assignment is:

```
mail-content-name = "X-Content-Type";
```

You might want to change it to `Content-Type` if you have a mail system that fully implements Internet RFC 1049 message headers. The MMDeliver program supplied with BBN/Slate will recognize either form.

mail-content-type	A string indicating the content type of the mail message.

If the value is not an empty string, then the file submitted to the mail spooler will contain a header of the form:

```
X-Content-Type: value of mail-content-type
```

This variable is normally used when multimedia messages are being encoded prior to delivery. It indicates the encoding type used, so that the mail system on the recipient's host will be able to decode the message when it arrives. To disable the generation of the `X-Content-Type` header, set the variable's value to an empty string, like this:

```
mail-content-type = "";
```

The default is:

```
mail-content-type = "X-BBN-Encoded-Multimedia;1.0";
```

The value has been chosen to conform to the syntax specified in

Internet Request For Comments number 1049 (RFC 1049, "Content-type header field for Internet messages"). It should not be changed lightly, since MMDeliver looks for this magic value to indicate encoded multimedia mail.

mail-delete-when-sent

If non-zero, the editor will delete automatically generated composition, reply, and forward buffers after they are sent as electronic mail. If this variable is set to 0, the buffers will not be automatically deleted. The default value is 1. It should not be changed at the moment, because the implementation of nontransient buffers is currently incomplete.

mail-encoder

If mail-spooler-requires-encoding is true, this program will be used to convert the 8-bit multimedia message body to a 7-bit encoding. The program is expected to read from its standard input and write to its standard output. The default value is `mmencode`, a program supplied with BBN/Slate. It should not be altered lightly, since the `mmdeliver` mail delivery program is designed to work with MMEncode's encoding format.

mail-encoder-ignore-exitval

An integer indicating that the mail-encoder program's exit status is not to be relied upon.

When called upon to encode mail messages prior to delivery, the editor will test the exit status of the mail-encoder program to make sure it succeeded. It assumes that an exit status matching the value of `mail-encoder-successful-exitval` indicates success; it also assumes that any other status indicates failure, in which case it will issue a lengthy diagnostic message.

However, it is possible that some user-supplied mail encoders will return unpredictable results even when they succeed in encoding the message. Setting `mail-encoder-ignore-exitval` to a nonzero value will suppress the error checking and diagnostic messages in such cases.

The default value for `mail-encoder-ignore-exitval` is 0.

mail-encoder-successful-exitval	An integer indicating the exit value the mail encoder uses to signal successful encoding. Most UNIX programs will exit with a zero termination status to indicate success. However, there are exceptions, and BBN/Slate is prepared to cope with them. The default `mail-encoder-successful-exitval` is 0.
mail-spooler	A string that specifies the name of the program used to enqueue mail for delivery. The name may contain shell wildcard and redirection characters, in which case it will be evaluated by `/bin/sh`. You may include additional information such as header field values and spool file names by embedding the following special sequences in the spooler name:
%F	The value of the **From:** field
%T	The value of the **To:** field
%C	The value of the **Cc:** field, if any
%B	The value of the **Bcc:** field, if any
%S	The value of the **Subject:** field, if any
%D	The value of the **Date:** field
%s	The name of the temporary file in which the message may be found

When header fields are inserted into the spooler string, *only* the field values are included; the field names are not inserted unless you include them explicitly in the spooler string.

The field values are inserted exactly as typed by the user, with no automatic quoting of multi-word values or escaping of shell wildcards. If your mail spooler requires such quoting, it is your responsibility to include the appropriate quote characters in the mail spooler string.

The default is:

```
mail-spooler = "/usr/lib/sendmail -t -Oeq < %s"
```

mail-spooler-extracts-headers

An integer that indicates whether the mail-spooler can extract message headers from the mail message.

If the value is non-zero, the spool file will consist of the message headers, followed by a blank line, followed by the (optionally encoded) message contents.

If the value is 0, the spool file will consist of the message contents alone; any required header information is presumed to be included in the mail-spooler variable.

The default value for `mail-spooler-extracts-headers` is 1.

mail-spooler-ignore-exitval

An integer indicating that the mail-spooler program's exit status can not be relied on.

When it calls the local mail transport program to deliver a multimedia message, the editor will test the exit status of the mail-spooler program to make sure it succeeded. It assumes that an exit status matching the value of `mail-spooler-successful-exitval` indicates success; it also assumes that any other status indicates failure, in which case it will issue a lengthy diagnostic message.

However, it is possible that some user-supplied mail spoolers will return unpredictable results even when they succeed in enqueuing the message. Setting `mail-spooler-ignore-exitval` to a non-zero value will suppress the error checking and diagnostic messages in such cases.

The default value for `mail-spooler-ignore-exitval` is 0.

mail-spooler-requires-encoding

An integer that indicates whether the mail delivery program requires encoding of non-ASCII messages.

If the value is non-zero, the mail transport program requires that messages be composed of 7-bit ASCII characters. Since BBN/Slate documents are binary files using 8-bit characters, they must be encoded into an ASCII-only format before they may be submitted to such a mail transport program.

If the value is 0, however, the mail transport program is assumed to be capable of dealing with binary files directly, and the multimedia message is not encoded.

The default value for `mail-spooler-requires-encoding` is 1.

mail-spooler-successful-exitval

An integer indicating the exit value the mail spooler uses to signal that it has successfully enqueued the mail.

Many UNIX mail-spooling programs will exit with a zero termination status to indicate success. However, there are exceptions, most notably MMDF (whose mail submission program indicates success with exit status 9), and BBN/Slate is prepared to cope with them.

The default `mail-spooler-successful-exitval` is 0.

Mail variables for sending text mail

When delivering text mail, the BBN/Slate editor converts the multimedia message to a text message (replacing non-text objects with legends to mark where they occurred in the original document), then writes the message headers and text message to a spool file, and calls on the local mail delivery agent to submit the file for delivery.

The exact behavior of each of these steps is controlled by variables that you can set in the sitewide profile file (`slate_editor.init`) or in a user's private profile file (`.slate_editor.init`). See Chapter 1 for more information on profile files. Variables are set by including a line of the form

```
variable-name = numeric value;
```

or

```
variable-name = "string value";
```

in the profile file. Double quote marks may be embedded in string values by preceding them with a backslash character, exactly as in the shell.

Encoding and delivery of multimedia mail is controlled by the following three editor configuration variables. The default values have been chosen so that they are suitable for use with `/usr/lib/sendmail` as the mail delivery agent.

text-mail-spooler A string that specifies the name of the program used to enqueue text mail for delivery. This will normally be the same as the `mail-spooler` used for encoded multimedia messages. The name may contain shell wildcard and redirection characters, in which case it will be evaluated by `/bin/sh`.

You may include additional information such as header field values and spool file names by including the special character sequences mentioned above, in the description of the `mail-spooler` variable.

The text mail delivery program is assumed to extract headers and delivery information from its input. The spool file consists of the message headers, followed by a blank line, followed by the text of the message. No special encoding is supported.

The default assignment is:

```
text-mail-spooler = "/usr/lib/sendmail -t -Oeq < %s";
```

text-mail-spooler-successful-exitval

An integer indicating the exit value the text mail spooler uses to signal that it has successfully enqueued the text mail.

Many UNIX mail spooling programs will exit with a zero termination status to indicate success. However, there are exceptions, most notably MMDF (whose mail submission program indicates success with exit status 9), and BBN/Slate can work with them if you set value for `text-mail-spooler-successful-exitval` accordingly.

The default `text-mail-spooler-successful-exitval` is 0.

text-mail-spooler-ignore-exitval

An integer indicating that the text-mail-spooler program's exit status can not be relied on.

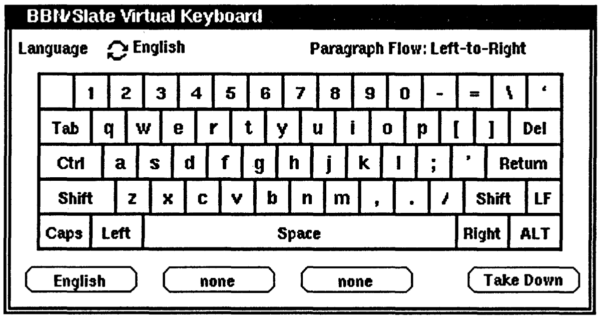
When it calls the local mail transport program to deliver a multimedia message, the editor will test the exit status of the `text-mail-spooler` program to make sure it succeeded. It assumes that an exit status matching the value of `text-mail-spooler-successful-exitval` indicates success; it also assumes that any other status indicates failure, in which case it will issue a lengthy diagnostic message.

However, it is possible that some user-supplied mail spoolers will return unpredictable results even when they succeed in enqueuing the message. Setting `text-mail-spooler-ignore-exitval` to a non-zero value will suppress the error checking and diagnostic messages in such cases.

The default value for `text-mail-spooler-ignore-exitval` is 0.

A.2.10 Multilingual variables

The keyboard display (which you can access by choosing **Keycaps** from the main menu or by invoking the `display-keyboard` function) includes three buttons at the bottom, two of which, by default, are set to display `none`:


BBN/Slate Virtual Keyboard
Language English
Paragraph Flow: Left-to-Right
1 2 3 4 5 6 7 8 9 0 - = \ '
Tab q w e r t y u i o p [ ] Del
Ctrl a s d f g h j k l ; ' Return
Shift z x c v b n m , . / Shift LF
Caps Left Space Right ALT
English
none
none
Take Down

If your site is using the multilingual version of BBN/Slate, you can shift the keyboard to other languages by simply selecting one of these three "fast keyboard" buttons, once you define them. You may find this more convenient than switching languages by using the rotary selection switch at the upper left of the keyboard display.

keyboard_button1 *Foreign language versions only.* The keyboard to place on the first "fast keyboard" button on the keycaps dialog. For example:

```
keyboard_button1 = "Russian";
```

keyboard_button2	*Foreign language versions only.* The keyboard to place on the second "fast keyboard" button on the keycaps dialog.
keyboard_button3	*Foreign language versions only.* The keyboard to place on the third "fast keyboard" button on the keycaps dialog.

B Listing of Functions

This appendix lists all user-callable BBN/Slate functions. These are the default functions provided with this release of the software. It is intended as a general-purpose reference tool for functions that you can employ in your own BBN/Slate extension language functions. You can scan this appendix to gain an overview of the breadth of functionality available.

You can also use the `apropos` function to assist you in locating functions having to do with a particular topic area. There are two ways to invoke `apropos`: by selecting **BBN-Help-Apropos** from the main menu, or by invoking `execute-command-dialog` (for which the default key binding is **Esc-X**) and typing `apropos`. In response to the `apropos` dialog box, enter any potential topic or string of characters that you believe might be part of a function name (examples are `char`, `string`, `text`, `buf`, and so on). `Apropos` then displays a scrolling box containing any functions that include your entry. Select any of the functions to display its description in a dialog box.

This alphabetical listing was created using the function `display-functions-in-buffer` (see Chapter 3). As you add your own user-callable functions, they will be added to it whenever you create another list. Note that once you create a listings file in this manner, you may also find it convenient to use normal BBN/Slate search commands on the file to locate topics of interest.

about-slate	Displays a dialog that tells you what version of BBN/Slate you are using, and when it was created. Returned result type is: void. No arguments.
add-audio	Add a speech object at the current cursor position. Returned result type is: void. No arguments.
add-blank-image	Add a blank image at the current cursor position. Returned result type is: void. No arguments.
add-empty-buffer	Open an empty buffer with the specified buffer name and file name. The buffer name must be unique and may not be empty. This routine returns the newly created buffer, or NULL in case of errors. Note that add-empty-buffer does NOT switch to the new buffer. Use change-to-named-buffer to make the new buffer become the current buffer. Returned result type is: buffer. Argument types are: string, string.
add-enclosure	Add an enclosure at the current cursor position. Returned result type is: void. No arguments.
add-generic-image	Add some type of monochrome or color bitmap image to the document. This command prompts for a file name, then tries to derive the image type by examining the file contents. Returned result type is: void. No arguments.
add-graphics	Add a graphics object at the current cursor position. Returned result type is: void. No arguments.
add-header	Add a header to the start of the document if none exists and display a dialog for the user to fill in the header fields. Returned result type is: void. No arguments.
add-image-from-file	Prompt for the name of a BBN/Slate image file and add its contents at the current cursor position. Returned result type is: void. No arguments.

add-image-from-named-file	Add the contents of the named BBN/Slate image file at the current cursor position. Returned result type is: void. Argument types are: string.
add-named-raster	Add the contents of the named rasterfile at the current cursor position. Returned result type is: void. Argument types are: string.
add-named-xwd	Add the contents of the named X Window Dump file at the current cursor position. Returned result type is: void. Argument types are: string.
add-object	Displays a scrolling list of objects which may be added to the document and adds the selected object at the current cursor position. Returned result type is: void. No arguments.
add-raster	Prompt for the name of a rasterfile and add its contents at the current cursor position. Returned result type is: void. No arguments.
add-spreadsheet	Add a spreadsheet at the current cursor position. Returned result type is: void. No arguments.
add-xwd	Prompt for the name of an X Window Dump file and add its contents at the current cursor position. Returned result type is: void. No arguments.
alias	Bind an alias name to its value. Returned result type is: void. Argument types are: string, string.
any-buffer-is-modified	Returns true if any buffer is modified. Returned result type is: int. No arguments.
append-buffer-to-document	Append the specified buffer to the specified file. Returned result type is: void. Argument types are: buffer, string.
append-buffer-to-file-as-text	Append the specified buffer to the specified file as text. Returned result type is: void. Argument types are: buffer, string.

apropos	Display a list of published functions that contain the specified word. Selecting one of the functions describes that function. Returned result type is: void. Argument types are: string.
apropos-dialog	Query the user for a word and then display a list of published functions that contain that word. Selecting one of the functions describes that function. Returned result type is: void. No arguments.
audio-edit	Pop up a dialog to edit (record and play) the selected audio object. Returned result type is: void. No arguments.
audio-edit-caption	Edit the caption you see in the audio object on the screen. Returned result type is: void. No arguments.
audio-edit-font	Change the font of the caption on the screen. Returned result type is: void. No arguments.
audio-listen	Plays the current audio object without a control panel. Interrupted by any mouse button. Returned result type is: void. No arguments.
audio-top-menu	Put up menu to manipulate the audio object. Returned result type is: void. No arguments.
audio-use-clipboard-icon	Use icon in the clipboard for the icon of the caption on the screen. Returned result type is: void. No arguments.
audio-use-default-icon	Change the icon of the caption on the screen. Returned result type is: void. No arguments.
audio-use-file-icon	Use the default icon for the icon of the caption on the screen. Returned result type is: void. No arguments.
beginning-of-buffer	Returns the start of the document. The returned type is marker. No arguments.

beginning-of-region	Returns the start of the specified region. The returned type is marker.
bind	Bind a key sequence in a keymap to a function. When the key sequence is typed, the function is executed. The first argument is a keymap name; the second argument is a key sequence and the third argument is a function name. The function must not be one that takes arguments or returns a result. Returned result type is: void. Argument types are: string, string, string.
bound	Return the function bound to a key sequence in a keymap. The first argument is the keymap name, the second is the key sequence. Returned result type is: string. Argument types are: string, string.
buffer-is-modified	Return true if the specified buffer is modified, otherwise return false. Returned result type is: int. Argument types are: buffer.
buffer-of-pane	Takes a pane number and returns the buffer currently contained in that pane. Returns NULL and sets the global error flag if the pane number is out of legal range. Returned result type is: buffer. Argument types are: int.
cd	Prompt for a directory name and make it the editor's working directory. Returned result type is: void. No arguments.
change-directory	Change the editor's working directory to the named directory. Returned result type is: void. Argument types are: string.
change-directory-dialog	Prompt for a directory name and make it the editor's working directory. Returned result type is: void. No arguments.
change-to-buffer	Query the user for the name of the buffer to make the current buffer. Returned result type is: void. No arguments.
change-to-named-buffer	Make the named buffer be the current buffer. Returned result type is: void. Argument types are: string.

char-to-int	Convert character to integer. Returned result type is: int. Argument types are: char.
char-to-string	Convert character to string. Returned result type is: string. Argument types are: char.
check-spelling	Check the spelling in the document. A dialog box will be displayed allowing you to select which parts of the document will be checked. You may check the entire document, the currently selected region, or from the cursor to the end of the document. Returned result type is: void. No arguments.
check-spelling-of-document	Check the spelling of all text in the document. Text within other media types (spreadsheets, graphics, headers, voice captions) will also be checked. Returned result type is: void. No arguments.
check-spelling-of-region	Check the spelling of all text in the selected region. Text within other media types (spreadsheets, graphics, headers, voice captions) will be checked wherever possible. Returned result type is: void. No arguments.
checkpoint-modified-buffers	Checkpoint all modified buffers. Returned result type is: void. No arguments.
clear	Clear the selected region. Returned result type is: void. No arguments.
clear-buffer	Clear the specified buffer. Returned result type is: void. Argument types are: buffer.
clear-current-buffer	Clear the current buffer. Returned result type is: void. No arguments.
command-loop	Loop receiving and processing user events. Loop can be exited by call `command-loop-exit()`. Returned result type is: void. No arguments.

command-loop-exit	Exit closest enclosing command loop. Returned result type is: void. No arguments.
command-loop-start	If a user-written routine is going to be used to receive and process user events, this routine should be called at the start to clean up various internal state variables to ensure proper behavior. Returned result type is: void. No arguments.
copy	Copy the selected region. Returned result type is: void. No arguments.
counter-add-occurrence	Add an occurrence of a counter. The arguments are counter name, optional tag, counter level, style (0 = increment, 1 = init, 2 = noincr), and initial value (if style = 1). Returned result type is: void. Argument types are: string, string, int, int, int.
counter-add-occurrence-dialog	Add an occurrence of a counter. Display a dialog to specify which counter. Returned result type is: void. No arguments.
counter-add-reference	Add a reference to a counter. Arguments are counter name, counter tag, and style (0 = counter value, 1 = counter page). Returned result type is: void. Argument types are: string, string, int.
counter-add-reference-dialog	Add a reference to a counter. Display a dialog to specify which counter. Returned result type is: void. No arguments.
counter-define-style-dialog	Display a dialog for creating a new counter. Returned result type is: void. No arguments.
counter-delete-named	Delete the named counter. Returned result type is: void. Argument types are: string.
counter-delete-unused	Delete all the unused counters in the document. Returned result type is: void. No arguments.

counter-describe	Return a string describing the counter near the mark. Returned result type is: string. No arguments.
counter-find	Find an instance of a counter and set the point there. Arguments are the counter name, the tag, and 0 if looking for the actual counter and 1 if looking for a reference to it. Returned result type is: void. Argument types are: string, string, int.
counter-find-tag-dialog	Find an instance of a tag. Displays a dialog to specify the tag. Returned result type is: void. No arguments.
counter-find-tag-ref-dialog	Find a reference to a tag. Displays a dialog to specify the tag. Returned result type is: void. No arguments.
counter-tag-occurrence	Modify the tag or style of the counter near the cursor. The arguments are the tag, the style (0 = increment, 1 = init), and the initial value if the style = init. Returned result type is: void. Argument types are: string, int, int.
counter-tag-occurrence-dialog	Display a dialog to modify the tag or style of the counter near the cursor. Returned result type is: void. No arguments.
create-new-buffer	Create an empty document buffer using the default document style. The user is prompted for the name of the buffer. The buffer is not associated with any file until it is saved for the first time. Returned result type is: void. No arguments.
create-region	Creates a region of the document and returns that region. The returned type is region. Argument types are: marker, marker.
current-buffer	Return the currently selected document buffer. Returned result type is: buffer. No arguments.
current-document	Return the current document. Returned result type is: document. No arguments.

current-file	Return the name of the file that is in the current buffer. Returned result type is: string. No arguments.
current-mark	Returns the current mark (the end of the currently selected region). The returned type is marker. No arguments.
current-object-type	Return the type name of the currently selected object. Returned result type is: string. No arguments.
current-pane	Return the pane number of the current pane. Returned result type is: int. Argument types are: void.
current-point	Returns the current point (cursor location). The returned type is marker. No arguments.
current-region	Returns the currently selected region of the document. The returned type is region. No arguments.
cut	Cut the selected region. Returned result type is: void. No arguments.
deiconify	De-iconify the editor window. Returned result type is: void. No arguments.
delete-all-checkpoints	Delete checkpoint files in preparation for exiting. Returned result type is: void. No arguments.
delete-buffer	Delete the given document buffer. Returned result type is: void. Argument types are: buffer.
delete-buffer-checkpoint	Delete the checkpoint file associated with a buffer. Returned result type is: void. Argument types are: buffer.
delete-buffer-with-filename	Delete the buffer associated with the named file. Returned result type is: void. Argument types are: string.

delete-current-buffer	Delete the current buffer. Returned result type is: void. No arguments.
delete-current-pane	Delete the current pane. Returned result type is: void. No arguments.
delete-named-buffer	Delete the buffer with the specified name. Returned result type is: void. Argument types are: string.
delete-other-panes	Delete all but the current pane. Returned result type is: void. No arguments.
describe-buffer-variable	Describe the given buffer-specific variable. Returned result type is: void. Argument types are: string.
describe-buffer-variable-dialog	Describe a buffer-specific variable. Returned result type is: void. No arguments.
describe-command	Describe an user callable editdoc function or an user defined function. Returned result type is: void. No arguments.
describe-document-variable	Describe the given document-specific variable. Returned result type is: void. Argument types are: string.
describe-document-variable-dialog	Describe a document-specific variable. Returned result type is: void. No arguments.
describe-global-variable	Describe the given global variable. Returned result type is: void. Argument types are: string.
describe-global-variable-dialog	Describe a global variable. Returned result type is: void. No arguments.
describe-key	Give the name of the function to which a sequence of keys is bound. Returned result type is: void. No arguments.

describe-variable	Describe a variable. Searches the buffer specific, document specific and global variable tables in that order. Returned result type is: void. No arguments.
deselect-all-objects	Deselect any objects which are currently selected. Returned result type is: void. No arguments.
designate-variable-buffer-specific	Make the specified variable buffer specific. Returned result type is: void. Argument types are: string.
designate-variable-document-specific	Make the specified variable document specific. Returned result type is: void. Argument types are: string.
display-buffer-status	Display the status of all buffers. Returned result type is: void. No arguments.
display-compile-date	Display the compile date at the bottom of the display. Returned result type is: void. No arguments.
display-confirmation	Displays the specified message in a dialog box with buttons labelled yes and no and returns 1 if the user selects yes and 0 if the user selects no. Returned result type is: int. Argument types are: string.
display-copyright	Display the copyright at the bottom of the display. Returned result type is: void. No arguments.
display-current-directory	Display the editor's current working directory in the message area. Returned result type is: void. No arguments.
display-current-directory-dialog	Display the editor's current working directory in a popup dialog box. Returned result type is: void. No arguments.
display-current-keymap-in-buffer	Display the current key mappings. Returned result type is: void. No arguments.

display-error	Display the specified message in a dialog box. Returned result type is: void. Argument types are: string.
display-functions-in-buffer	Display descriptions of the published functions. Returned result type is: void. No arguments.
display-keyboard	Put up the graphical keyboard in a dialog window. Returned result type is: void. No arguments.
display-message	Display the specified message in a dialog box. Returned result type is: void. Argument types are: string.
display-print-dialog	Display printing dialog and dispatch print job based on the user action. Returned result type is: void. No arguments.
display-string-in-status-line	Display the specified message in the bottom line of the display (below the buffer title banner). Returned result type is: void. Argument types are: string.
display-system-error	Display the specified string in a dialog box followed by the system error message derived from errno. Returned result type is: void. Argument types are: string.
display-unbound-error	Print a message at the bottom of the display saying that a key is not bound to any function. Returned result type is: void. No arguments.
do-nothing	Routine which does nothing. Placeholder in keymaps. Returned result type is: void. No arguments.
document-move-point	Move the cursor location by the number of characters specified in the argument. Returned result type is: void. Argument types are: int.
document-set-object	Set the current object to the object at the given location. Returned result type is: void. Argument types are: marker.

document-set-point	Set the cursor to the specified location in the document. Returned result type is: void. Argument types are: marker.
edit-object-attributes	Change the display or printing attributes of the currently selected object. Attributes that may be changed include whether or not the outline is displayed around this object, whether this object forces a page break, the object's justification on the page, classification level, and print resolution. Returned result type is: void. No arguments.
enclosure-change-font	Change the font used for this enclosure's caption. Returned result type is: void. No arguments.
enclosure-deselect-and-insert	Deselect the current enclosure object and insert the character just typed immediately after the enclosure. This command is normally bound to all printing characters while an enclosure is selected. It should not be bound to non-printing characters, as unpredictable results may occur. Returned result type is: void. No arguments.
enclosure-edit-data	Performs the "Edit" operation defined for an enclosure. Returned result type is: void. No arguments.
enclosure-edit-description	Modify the file, type, or commands defined for an enclosure. Returned result type is: void. No arguments.
enclosure-edit-layout	Modify the icon or margins for this enclosure. Returned result type is: void. No arguments.
enclosure-execute	Performs the "Execute" operation defined for an enclosure. Returned result type is: void. No arguments.
enclosure-execute-command	Searches the current enclosure for a command whose name matches its argument, and executes that command if it exists. Returned result type is: void. Argument types are: string.
enclosure-print	Performs the "Print" operation defined for an enclosure. Returned result type is: void. No arguments.

enclosure-reload-data	Reloads the data for an enclosure from the named file. This command operates on enclosures that include data by copying, not on those that include data by reference. Returned result type is: void. No arguments.
enclosure-show-filename	Display the filename from which this enclosure was created. Returned result type is: void. No arguments.
enclosure-top-menu	Put up the top-level menu for the current enclosure object. If the current object is not an enclosure, do nothing. Returned result type is: void. No arguments.
enclosure-user-command-1	Performs the first user-defined operation for an enclosure. Returned result type is: void. No arguments.
enclosure-user-command-2	Performs the first user-defined operation for an enclosure. Returned result type is: void. No arguments.
enclosure-user-command-3	Performs the first user-defined operation for an enclosure. Returned result type is: void. No arguments.
enclosure-user-command-4	Performs the first user-defined operation for an enclosure. Returned result type is: void. No arguments.
enclosure-write-data	Prompt for a filename, and write the data from an enclosure to the file specified. This command operates on enclosures that include data by copying, not on those that include data by reference. Returned result type is: void. No arguments.
enclosure-write-data-to-file	Write the data from an enclosure to a named file. This command operates on enclosures that include data by copying, not on those that include data by reference. Returned result type is: void. Argument types are: string.
end-of-buffer	Returns the end of the document. The returned type is marker. No arguments.

end-of-region	Returns the end of the specified region. The returned type is marker. Argument types are: region.
event-current-code	Return the character code of the last event received. Returned result type is: int. No arguments.
event-current-x	Return the x coordinate of the last event received. Returned result type is: int. No arguments.
event-current-y	Return the y coordinate of the last event received. Returned result type is: int. No arguments.
event-get-next-code	Receive the next event and return its code. Returned result type is: int. No arguments.
event-is-pending	Return non-zero if an event is pending. Returned result type is: int. No arguments.
event-process-current	Process the event just received. This involves handling events in scrollbars, borders, keepup dialogs, as well as standard events in the pane. Returned result type is: void. No arguments.
event-wait-for-next	Receive the next event and return its code. Handle any pending operations before receiving the event. Returned result type is: int. No arguments.
execute-command-dialog	Execute an user callable editdoc function or an user defined function. Returned result type is: void. No arguments.
execute-string	Execute the string as extension language commands. Returned result type is: void. Argument types are: string.
exit	Exit the editor. If there are any modified buffers, query the user to ask whether the buffers should be saved. Returned result type is: void. No arguments.

exit-without-saving-changes	Exit the editor. This does not save modified buffers. Returned result type is: void. No arguments.
filename-of-buffer	Return the name of the file that is in the specified buffer. Returned result type is: string. Argument types are: buffer.
find-buffer	Return the buffer with the specified name. Returned result type is: buffer. Argument types are: string.
find-buffer-containing-filename	Return the buffer containing the specified file. Returned result type is: buffer. Argument types are: string.
find-media-type	Search for an instance of a media type or text style. A dialog is displayed prompting for the media type to find, and the direction in which to search. Returned result type is: void. No arguments.
find-media-type-again	Find another instance of the media type last specified in a find command. Returned result type is: void. No arguments.
gather-repeat-count	Collect characters for specifying repeat count and set global repeat count. Returned result type is: void. No arguments.
generate-tables	Generate the tables of contents, figures, etc. This function will repaginate the document before generating the tables to assure all page references are correct. Returned result type is: void. No arguments.
get-menu	Returned result type is: menu. Argument types are: string.
graphics-add-arc	Add an Arc feature to the drawing. Returned result type is: void. No arguments.
graphics-add-box	Add a Box feature to the drawing. Returned result type is: void. No arguments.
graphics-add-circle	Add a Circle feature to the drawing. Returned result type is: void. No arguments.

graphics-add-ellipse	Add an Ellipse feature to the drawing. Returned result type is: void. No arguments.
graphics-add-freehand	Add a Freehand feature to the drawing. Returned result type is: void. No arguments.
graphics-add-freehand-closed	Add a Closed Freehand feature to the drawing. Returned result type is: void. No arguments.
graphics-add-line	Add a Line feature to the drawing. Returned result type is: void. No arguments.
graphics-add-lineseq	Add a Line Sequence feature to the drawing. Returned result type is: void. No arguments.
graphics-add-polygon	Add a Polygon feature to the drawing. Returned result type is: void. No arguments.
graphics-add-rounded-box	Add a Rounded Box feature to the drawing. Returned result type is: void. No arguments.
graphics-add-spline	Add a Spline feature to the drawing. Returned result type is: void. No arguments.
graphics-add-text	Add a Text feature to the drawing. Returned result type is: void. No arguments.
graphics-add-wedge	Add a Wedge feature to the drawing. Returned result type is: void. No arguments.
graphics-align-all-features-to-grid	Align the selected features to the grid as a group. Returned result type is: void. No arguments.
graphics-align-each-feature-to-grid	Align each selected feature to the grid separately. Returned result type is: void. No arguments.

graphics-align-features	Align selected features according to Align dialog request. Returned result type is: void. No arguments.
graphics-align-features-dialog	Put up Align Features dialog and execute it. Returned result type is: void. No arguments.
graphics-align-to-grid	Align the selected features to the grid. The user is asked whether they should be aligned separately or as a group. Returned result type is: void. No arguments.
graphics-change-arrows	Add arrow if none or remove arrow if there was one. Returned result type is: void. No arguments.
graphics-clear	Cut the selected features from the drawing and discard them. Returned result type is: void. No arguments.
graphics-copy	Copy the selected features onto the clipboard. Returned result type is: void. No arguments.
graphics-cut	Cut the selected features from the drawing and put them on the clipboard. Returned result type is: void. No arguments.
graphics-deselect-all-features	Turn off selection of all selected features. Returned result type is: void. No arguments.
graphics-deselect-feature	Deselect the feature near the mouse. Returned result type is: void. No arguments.
graphics-duplicate	Add duplicate copies of the selected features to drawing. Returned result type is: void. No arguments.
graphics-font-dialog	Offer dialog to change font of selected text features or default. Returned result type is: void. No arguments.
graphics-group-objects	Create group of all selected features and add it to the drawing. Returned result type is: void. No arguments.

graphics-lock-object	Lock the selected features so they cannot be modified. Returned result type is: void. No arguments.
graphics-move	Try to interpret mouse click as move operation. Returned result type is: void. No arguments.
graphics-move-to-back	Push the selected features to the background of the drawing. Returned result type is: void. No arguments.
graphics-move-to-front	Pull the selected features to the foreground of the drawing. Returned result type is: void. No arguments.
graphics-palette	Display the palette used for creating and editing graphics features. Returned result type is: void. No arguments.
graphics-pan	Select a point with the mouse and make this the new center point in the viewport. Returned result type is: void. No arguments.
graphics-paste	Add the contents of the clipboard to the drawing. Returned result type is: void. No arguments.
graphics-read-from-file	Replace current drawng with the contents of a given file. Returned result type is: void. No arguments.
graphics-repeat-add-feature	Add another of the last type of feature added. Returned result type is: void. No arguments.
graphics-reshape	Evaluate mouse hold as reshape operation. Returned result type is: void. No arguments.
graphics-reshape-move-select	Evaluate mouse click as reshape, move, or select operation. Returned result type is: void. No arguments.
graphics-restore-original	Restore the original view of the drawing. Returned result type is: void. No arguments.

graphics-round-corners	Returned result type is: void. Argument types are: int.
graphics-save-to-file	Save currently selected drawing in a file. Returned result type is: void. No arguments.
graphics-scale	Reduce or enlarge the selected features. Returned result type is: void. No arguments.
graphics-select-additional-feature	Select a feature, preserving other selected features. Returned result type is: void. No arguments.
graphics-select-all	Select all features in the drawing Returned result type is: void. No arguments.
graphics-select-new-feature	Select a new feature, deselecting all other features. Returned result type is: void. No arguments.
graphics-select-with-box	Try to interpret mouse click as Select operation. Returned result type is: void. No arguments.
graphics-set-arrow-style	Change default arrow style for new features. Returned result type is: void. Argument types are: int.
graphics-set-grid-size	Returned result type is: void. No arguments.
graphics-set-resolution	Display dialog to change resolution. Returned result type is: void. No arguments.
graphics-set-ruler-units	Returned result type is: void. No arguments.
graphics-set-text-justification	Change text justification of selected features (or default). Returned result type is: void. Argument types are: int.
graphics-smooth-objects	Smooth line sequences into splines or unsmooth splines into line sequences. Returned result type is: void. No arguments.

graphics-toggle-griddisplay	Toggle whether the grid is displayed. Returned result type is: void. No arguments.
graphics-toggle-gridmode	Toggle whether the grid is turned on. Returned result type is: void. No arguments.
graphics-toggle-rulersdisplay	Toggle whether rulers are displayed. Returned result type is: void. No arguments.
graphics-toggle-showsize	Toggle whether the size of features is shown as they are moved and resized. Returned result type is: void. No arguments.
graphics-top-menu	Display Graphics Editor top-level menu. Returned result type is: void. No arguments.
graphics-undo	Undo the last operation, if possible. Returned result type is: void. No arguments.
graphics-ungroup-object	Ungroup all selected group features. Returned result type is: void. No arguments.
graphics-unlock-object	Unlock the selected locked features. Returned result type is: void. No arguments.
graphics-zoom-in	Draw a box outlining the area to zoom in on to see more detail. Returned result type is: void. No arguments.
graphics-zoom-out	Draw a box outlining the area to zoom out to see less detail. Returned result type is: void. No arguments.
handle-move-button	Move the selected multimedia object. Returned result type is: void. No arguments.
header-add-empty	Add headers to the document if they do not already exist. Returned result type is: void. No arguments.

header-check-fields	Add values for the From and Date fields to the header and ensure that there is a To and Cc list. If there are no addressees specified, allow the user to edit the headers to add them. If, after editing, there are still no addressees, return FALSE. Returned result type is: int. No arguments.
header-delete-field	Remove the specified field from the headers. If there are no headers in the current document or the named field is one that cannot be edited, then no action is taken. If the named field is one that cannot be removed, then only its value is deleted. Returned result type is: void. Argument types are: string.
header-deselect-text-appear	Deselect the header object, insert a text object and insert the character typed into that object. Returned result type is: void. No arguments.
header-edit	Edit the header object. Returned result type is: void. No arguments.
header-edit-name-font	Display a dialog to change the font of the names in the header display. Returned result type is: void. No arguments.
header-edit-value-font	Display a dialog to change the font of the values in the header display. Returned result type is: void. No arguments.
header-get-field	Return the value of the specified field in the header. If there are no headers in the current document or if the field does not exist, then an empty string is returned. Returned result type is: string. Argument types are: string.
header-set-field	Add the specified name/value pair to the document's headers. If the named field already exists, then it is replaced. If there are no headers in the document, then headers are added. If headers cannot be added or the named field is one that cannot be edited, then no action is taken. Returned result type is: void. Argument types are: string, string.

header-stamp-obj	Stamp the date and sender's name on the headers. Returned result type is: void. No arguments.
header-top-menu	Display the top-level header menu. Returned result type is: void. No arguments.
iconify	Iconify the editor window. Returned result type is: void. No arguments.
image-add-object	Displays the palette for adding geometric objects to an image. Returned result type is: void. No arguments.
image-add-text	Allows the user to type text on an image. Returned result type is: void. No arguments.
image-clear	Deletes the currently selected region of an image and replaces it with the background texture (but does not place the deleted region on the clipboard). Returned result type is: void. No arguments.
image-copy	Copies the current selected region of an image onto the clipboard. Returned result type is: void. No arguments.
image-crop	If there is a currently selected region of an image, this reduces the image to only contain that region. If no region is selected, this allows the user to interactively crop the image. Returned result type is: void. No arguments.
image-crop-to-viewport	Discard the image data that is not visible within the image viewport. This involves a loss of data. Returned result type is: void. No arguments.
image-cut	Deletes the currently selected region of an image and replaces it with the background texture. Returned result type is: void. No arguments.
image-deselect-region	Clears the image selection. Returned result type is: void. No arguments.

image-extend-select-region	Extends one corner of the selected region. Returned result type is: void. No arguments.
image-install-colormap	Install the colormap of the currently selected image. Returned result type is: void. No arguments.
image-invert	Inverts the selected region of an image (makes white black and black white). Returned result type is: void. No arguments.
image-paint	Allows the user to paint on the image using the current brush size and texture. Left mouse hold draws and middle mouse hold erases. Middle mouse click confirms the painting and right mouse click cancels it. Returned result type is: void. No arguments.
image-palette	Display the palette used for creating and editing images. Returned result type is: void. No arguments.
image-paste	Paste into the image from the clipboard. The user positions the clipboard image with the mouse and then confirms the position by pressing one of the mouse buttons. Returned result type is: void. No arguments.
image-read-into-image	Queries the user for a file name containing an image in BBN/Slate format. The new image overwrites and destroys the current contents of the image. Returned result type is: void. No arguments.
image-reflect	Reflects the selected region of the image around some axis. This takes an integer argument which determines which axis to reflect around (0 - vertical, 1 - horizontal, 2 - upward diagonal, 3 - downward diagonal). Returned result type is: void. Argument types are: int.
image-scale	Displays a dialog for specifying how to scale the image. Returned result type is: void. No arguments.
image-scale-interactively	Allows the user to interactively scale an image. Returned result type is: void. No arguments.

image-select-region	Interacts with the user to select a rectangular region of the image with the mouse. Returned result type is: void. No arguments.
image-set-brush	Displays the dialog for setting size of the current paint brush. Returned result type is: void. No arguments.
image-set-font	Displays the dialog for setting the font used for text added to the image. Returned result type is: void. No arguments.
image-set-linewidth	Displays the dialog for setting the line width of the paint brush. Returned result type is: void. No arguments.
image-set-texture	Displays the dialog for adding objects to the image and setting line and fill textures. Returned result type is: void. No arguments.
image-tilt	Displays the dialog for specifying how to tilt the image. Returned result type is: void. No arguments.
image-tilt-interactively	Allows the user to interactively tilt an image (not implemented). Returned result type is: void. No arguments.
image-top-menu	Displays the top-level image menu. Returned result type is: void. No arguments.
image-undo	Undo the last image operation. Returned result type is: void. No arguments.
image-user-crop	Interacts with the user to crop the size of the image. Returned result type is: void. No arguments.
image-write-to-file	Query the user for the name of a file. The current image is written out to this file in BBN/Slate image format. Returned result type is: void. No arguments.
index-cleanup	Removes any index entries in the current region. Returned result type is: void. No arguments.

index-delete-specific	Delete the index entry with the given identifier. Returned result type is: void. Argument types are: int.
index-edit	Allows the user to edit the index entry at the cursor. Returned result type is: void. No arguments.
index-edit-specific	Edit the index value with this index identifier. Returned result type is: void. Argument types are: int.
index-find	Find the index entry with the given label and return its identifier. If a region is selected it is limited to that region, otherwise it proceeds from current point and wraps around document. The label specified may be a regular expression. Returned result type is: int. Argument types are: string.
index-find-dialog	Find the index entry with the label the user specifies and make that the selected region. Returned result type is: void. No arguments.
index-find-next	Find the next index entry in the document. Returned result type is: void. No arguments.
index-from-point	Return the identifier for an index entry at or after the current point. Returned result type is: int. No arguments.
index-generate	Generates the index to the file /tmp/index.out. Returned result type is: void. No arguments.
index-get	Return the contents of the index entry at this identifier. Returned result type is: string. Argument types are: int.
index-get-next	Find the next index entry in the document and return the label. Returned result type is: string. No arguments.
index-near-point	Return the identifier of an index entry near the current location of point. Return -1 if there is none. Returned result type is: int. No arguments.

index-next	Return the index entry after the given identifier; sets the error code and returns -1 if there are no more. Returned result type is: int. Argument types are: int.
index-region	Index the current region with the given label. Returned result type is: void. Argument types are: string.
index-region-automatically	Adds an index entry for the current region. The region is indexed under the term in the region. If no region is selected, the current word is indexed. Returned result type is: void. No arguments.
index-region-dialog	Allow the user to specify an index entry which is bound to the current region. Returned result type is: void. No arguments.
index-select	Select the region covered by the index entry specified by the given identifier. Returned result type is: void. Argument types are: int.
index-set	Set the index value with this index identifier to the given new value. Returned result type is: void. Argument types are: int, string.
initialize-error-buffer	Set up the buffer which is used to report errors. Returned result type is: void. No arguments.
initialize-help-buffer	Set up the help buffer for displaying information. Returned result type is: void. No arguments.
insert-document-dialog	Query the user for a document to insert at the cursor location. Returned result type is: void. No arguments.
insert-file-dialog	See insert-document-dialog. Returned result type is: void. No arguments.
insert-named-document	Insert the specified file at the cursor location. Returned result type is: void. Argument types are: string.

insert-named-file	See insert-named-document. Returned result type is: void. Argument types are: string.
insert-named-text-file	Insert the named text file at the current location. Returned result type is: void. Argument types are: string.
insert-text-file-dialog	Insert an ASCII text file at the cursor position. Returned result type is: void. No arguments.
int-to-char	Convert integer to character. Returned result type is: char. Argument types are: int.
keymacro-done-remembering	Stop saving key strokes to define a keyboard macro. Returned result type is: void. No arguments.
keymacro-playback	Execute a keyboard macro. Returned result type is: void. No arguments.
keymacro-start-remembering	Start saving key strokes to define a keyboard macro. Returned result type is: void. No arguments.
last-pane	Return the pane number of the highest-numbered pane that exists. The panes that exist at any given time are numbered from 0 to last-pane(), inclusive. Returned result type is: int. Argument types are: void.
location-convert-to-mark	Converts a location in a document to a marker. The returned type is marker. Argument types are: int.
mail-compose	Compose a new message. This creates a buffer named #compose and displays a dialog for the user to write a header in that buffer. Returned result type is: void. No arguments.
mail-forward	Forward the message in the current buffer. This creates a buffer named #forward and creates a header in that buffer. The original message is included in the buffer. Returned result type is: void. No arguments.

mail-reply	Reply to the message in the current buffer. This creates a buffer named #reply and creates a header in that buffer. The header subject contains the text Re: followed by the subject of the original message. Returned result type is: void. No arguments.
mail-reply-include	Reply to the message in the current buffer. This creates a buffer named #reply and creates a header in that buffer. The header subject contains the text Re: followed by the subject of the original message. The original message is copied into the reply. Returned result type is: void. No arguments.
mail-replycc	Reply to the message in the current buffer. This creates a buffer named #reply and creates a header in that buffer. The header subject field contains the text Re: followed by the subject of the original message. The header Cc field contains all the addresses on the To and Cc lists of the original message. Returned result type is: void. No arguments.
mail-replycc-include	Reply to the message in the current buffer. This creates a buffer named #reply and creates a header in that buffer. The header subject contains the text Re: followed by the subject of the original message. The original message is copied into the reply. Returned result type is: void. No arguments.
mail-send	Send the message in the current buffer. The system determines whether the message should be sent as text or multimedia. Returned result type is: void. No arguments.
mail-send-as-text	Send the message in the current buffer as a text message. Returned result type is: void. No arguments.
mail-send-multimedia	Send the message in the current buffer as a multimedia message. Returned result type is: void. No arguments.
main-top-menu	Display the top level menu. Returned result type is: void. No arguments.

mark-add-offset	Add an integer offset to a marker and return a marker pointing to the new location in the document. The returned type is marker. Argument types are: marker, int.
mark-buffer-modified	Mark the current document as modified. Returned result type is: void. No arguments.
mark-buffer-unmodified	Mark the specified document as not modified. Returned result type is: void. Argument types are: buffer.
mark-convert-to-location	Converts a marker to a location in a document. Returned result type is: int. Argument types are: marker.
mark-is-equal	Compare two markers in the document and return true if they are the same. Returned result type is: int. Argument types are: marker, marker.
mark-is-in-region	Returns true if the specified location is in the specified region. The ends of the region are considered in the region. Returned result type is: int. Argument types are: marker, region.
menu-build	Add a command to a menu. The arguments are the menu, the index of the command in the menu (starting with zero), the name to display in the menu, the help message, the submenu, and the name of the user callable function to execute. Returned result type is: void. Argument types are: menu, int, string, string, menu, string.
menu-create	Create a menu. Called with the menu name, the number of items that will be in the menu, the name of the function to invoke, and the parent menu. Returned result type is: menu. Argument types are: string, int, string, menu.
menu-display	Returned result type is: int. Argument types are: menu.

menu-initialize	Initialize a menu. Call after creating the menu and adding all the commands to it and before displaying the menu. Returned result type is: void. Argument types are: menu.
menu-invoke-function	Invoke the function of the given menuitem of the given menu. Returned result type is: void. Argument types are: int, string.
move-current-object-to-bottom	Move the bottom of the current object to the bottom of the pane. Returned result type is: void. No arguments.
move-current-object-to-top	Move the top of the current object to the top of the pane. Returned result type is: void. No arguments.
next-pane	Make the next pane be the current pane. Returned result type is: void. No arguments.
page-down	Display the next page of the document. Returned result type is: void. No arguments.
page-left	Move the viewport one page to the left. Returned result type is: void. No arguments.
page-right	Move the viewport one page to the right. Returned result type is: void. No arguments.
page-up	Display the previous page of the document. Returned result type is: void. No arguments.
paginate-current-buffer	Ensure that the current buffer is paginated and page references are up-to-date. Returned result type is: void. No arguments.
paste	Paste the selected region. Returned result type is: void. No arguments.
picture-install-colormap	Ensure that the currently selected picture's colormap is installed. Returned result type is: void. No arguments.

preview-buffer	Display a dialog to specify the parameters for the previewer and then preview the specified buffer. Returned result type is: void. Argument types are: buffer.
preview-current-buffer	Preview the printed form of the current buffer on the screen. Returned result type is: void. No arguments.
previous-pane	Make the previous pane be the current pane. Returned result type is: void. No arguments.
print-buffer	Prints the buffer passed as an argument. Returned result type is: void. Argument types are: buffer.
print-current-buffer	Print the current buffer. Returned result type is: void. No arguments.
process-attach	Attach a shell command to the given buffer, placing the output of the command into the buffer at the current point. If not in a text object, a `verbatim` text object is added and the text is placed in that. Returned result type is: void. Argument types are: buffer, string.
process-close-stdin	Close the input of the process associated with this buffer. Returned result type is: void. Argument types are: buffer.
process-input	Feed the given string to the process associated with the given buffer. Arguments are the buffer, the string, and the length of the string. Returned result type is: void. Argument types are: buffer, string, int.
process-is-attached	Returns non-zero if there is a process bound to this buffer. Returned result type is: int. Argument types are: buffer.
process-kill	Kill the process associated with this buffer. Returned result type is: void. Argument types are: buffer.

process-run	Execute a shell command and place the output at the current point in the given buffer. If not in a text object, a `verbatim` text object is added and the text is placed in that. This command will not return until the process exits. Returned result type is: void. Argument types are: buffer, string.
process-send-region	Pass the selected region in the given buffer to the process associated with the second buffer. Returned result type is: void. Argument types are: buffer, buffer.
process-send-text-region	Pass the selected region (as text) in the given buffer to the process associated with the second buffer. Returned result type is: void. Argument types are: buffer, buffer.
process-wait	Wait for the process associated with the given buffer to terminate. Returned result type is: void. Argument types are: buffer.
pwd	See display-current-directory. Returned result type is: void. No arguments.
pwd-in-dialog	See display-current-directory-dialog. Returned result type is: void. No arguments.
read-commands	Read extension language commands from a file. The user is queried for the name of the file. Returned result type is: void. No arguments.
read-commands-from-file	Read extension language commands from a specified file. If you supply only a file name, it looks for the file starting with the current directory, and then tries all the paths specified in the `library-search-path` variable. If you supply an absolute path, such as

```
read-commands-from-file("/home/smith/SELtest");
```

it looks for the file only at that location. You may use UNIX environment variables. For example, if the environment variable `sel` is set to the value `/nfs/xyzzy/u4/smith`, then you can represent that path by using:

```
read-commands-from-file("$sel/SELtest");
```

Returned result type is: void. Argument types are: string.

read-document-in-buffer	Return the buffer containing the specified file. Returned result type is: buffer. Argument types are: string.
read-file-in-buffer	See read-document-in-buffer. Returned result type is: buffer. Argument types are: string.
read-menus-from-file	Returned result type is: void. Argument types are: string.
read-string-dialog	Display a dialog for obtaining a string from the user and return that string. This takes two arguments, the first is a message that is placed at the top of the dialog, and the second is a prompt that is placed in a box within the dialog. The dialog also has confirm and cancel buttons. If the user hits the cancel button, this function sets the global variable, global-error. Returned result type is: string. Argument types are: string, string.
redisplay	Redisplay the display. Returned result type is: void. No arguments.
replace-buffer-contents	Read a file into the current buffer. The user is queried for the file name. Returned result type is: void. No arguments.
save-all-modified-buffers	Save all the modified buffers. Returned result type is: void. No arguments.
save-buffer	Save this buffer in its associated file; if no file is associated with the buffer, then the user is queried for a file name. Returned result type is: void. Argument types are: buffer.

save-buffer-as-text	Save this buffer as text in its associated file; if no file is associated with the buffer, then the user is queried for a file name. Returned result type is: void. Argument types are: buffer.
save-buffers-and-exit	Save all the modified buffers and exit editdoc. Returned result type is: void. No arguments.
save-current-buffer	Save the current buffer. Returned result type is: void. No arguments.
save-named-buffer	If the given file is in a buffer, save it. Returned result type is: void. Argument types are: string.
scroll-down	Scroll the document display up by one line so that the next line in the document is displayed at the top of the display. Returned result type is: void. No arguments.
scroll-left	Scroll the document viewport to the left. Returned result type is: void. No arguments.
scroll-right	Scroll the document viewport to the right. Returned result type is: void. No arguments.
scroll-up	Scroll the document display down by one line so that the previous line in the document is displayed at the top of the display. Returned result type is: void. No arguments.
select-entire-document	Select the entire document. Returned result type is: void. No arguments.
select-language	Returned result type is: void. No arguments.
set	Set the value of a variable. If it exists as a document or buffer variable, set it there, otherwise set it globally. Returned result type is: void. No arguments.

set-buffer-filename	Query the user for a filename to associate with this buffer. Returned result type is: int. Argument types are: buffer.
set-buffer-of-pane	Takes two arguments, a pane number and a buffer, and sets the buffer for the pane to the supplied buffer. Returned result type is: void. Argument types are: int, buffer.
set-buffer-variable	Set the value of the given buffer-specific variable to the given value. Returned result type is: void. Argument types are: string, string.
set-buffer-variable-dialog	Set the value of a buffer-specific variable. Returned result type is: void. No arguments.
set-current-buffer	Make the given buffer the current buffer. Returned result type is: void. Argument types are: buffer.
set-document-font	Set the default document font to the given value. Returned result type is: void. Argument types are: string.
set-document-font-dialog	Select the default document language and font from a dialog. Returned result type is: void. No arguments.
set-document-variable	Set the value of the given document-specific variable. Returned result type is: void. Argument types are: string, string.
set-document-variable-dialog	Set the value of a document-specific variable. Returned result type is: void. No arguments.
set-filename-of-buffer	Set the file name associated with the specified buffer. Returned result type is: void. Argument types are: buffer, string.
set-global-error-code	Set the global variable, global-error. Returned result type is: void. Argument types are: int.
set-global-variable	Set the value of the given global variable to the given value. Returned result type is: void. Argument types are: string, string.

set-global-variable-dialog	Set the value of a global variable. Returned result type is: void. No arguments.
set-keyboard-name	Set the current keyboard table to use. Returneu result type is: void. Argument types are: string.
set-last-function-executed	Set the named function as the last function executed in the global variable global-last-function. Returned result type is: void. Argument types are: string.
set-region-classification	Set the classification level of the currently selected region. Returned result type is: void. Argument types are: int.
set-repeat-count	Set the number of times to repeat an operation in the global variable global-repeat-count. Returned result type is: void. Argument types are: int.
sheet-add-select-area	Interact with the user to extend the currently selected region in the spreadsheet. Returned result type is: void. No arguments.
sheet-auto-compute	Set automatic calculation to true and recompute the spreadsheet if it was previously false. Returned result type is: void. No arguments.
sheet-clear	Delete the currently selected region of the spreadsheet (but do not place it on the clipboard). Returned result type is: void. No arguments.
sheet-command-button	Display the top-level spreadsheet menu. Returned result type is: void. No arguments.
sheet-compute	Cause all formulas in the spreadsheet to be recalculated. Returned result type is: void. No arguments.
sheet-compute-by-column	Set the recomputation style to compute by columns (calculate all formulas in the first column, then the second, then the third, etc.). Returned result type is: void. No arguments.

sheet-compute-by-row	Set the recomputation style to compute by rows (calculate all formulas in the first row, then the second, then the third, etc.). Returned result type is: void. No arguments.
sheet-compute-dialog	Display the dialog which allows the user to set the recomputation controls. Returned result type is: void. No arguments.
sheet-compute-natural	Set the recomputation style to "natural" order (recompute each formula after any formulas which it depends on). Returned result type is: void. No arguments.
sheet-copy	Copy the currently selected region of the spreadsheet to the clipboard. Returned result type is: void. No arguments.
sheet-cursor-col	Return the index of the cursor cell column. Returned result type is: int. No arguments.
sheet-cursor-down	Move down one cell (or the current number specified by the repeat count). Returned result type is: void. No arguments.
sheet-cursor-keys	Handle escape codes generated by the Sun cursor keys. Returned result type is: void. No arguments.
sheet-cursor-left	Move left one cell (or the current number specified by the repeat count). Returned result type is: void. No arguments.
sheet-cursor-next	Move to the next cell in the currently selected region. Returned result type is: void. No arguments.
sheet-cursor-right	Move right one cell (or the current number specified by the repeat count). Returned result type is: void. No arguments.
sheet-cursor-row	Return the index of the cursor cell row. Returned result type is: int. No arguments.
sheet-cursor-up	Move up one cell (or the current number specified by the repeat count). Returned result type is: void. No arguments.

sheet-cut	Delete the currently selected spreadsheet region and copy it to the clipboard. Returned result type is: void. No arguments.
sheet-delete-cols	Delete the currently selected columns (cells to the right will move over to fill the deleted region). The deleted columns are placed on the clipboard. Returned result type is: void. No arguments.
sheet-delete-rows	Delete the currently selected rows (cells below will move up to fill the deleted region). The deleted rows are placed on the clipboard. Returned result type is: void. No arguments.
sheet-deselect-area	Clear the currently selected region. Returned result type is: void. No arguments.
sheet-edit-cell	Edit a spreadsheet cell. Returned result type is: void. No arguments.
sheet-fill-down	Copy cells in the top row of the selected region into the selected rows below them. Returned result type is: void. No arguments.
sheet-fill-right	Copy cells in the right-most column of the selected region into the selected columns to the right. Returned result type is: void. No arguments.
sheet-flush-center	All the flush routines will change justification of the selected cells if a range of cells is selected, change the default for a column if an entire column is selected and change the default for the spreadsheet if an entire row is selected. Returned result type is: void. No arguments.
sheet-flush-left	Change the selected cells to be left-justified. Returned result type is: void. No arguments.
sheet-flush-leftright	Change the selected cells to be strings-left, values-right. Returned result type is: void. No arguments.

sheet-flush-repl	Change the selected cells to be replicated-justified. Returned result type is: void. No arguments.
sheet-flush-right	Change the selected cells to be right-justified. Returned result type is: void. No arguments.
sheet-font-dialog	Display the dialog for changing the font of the spreadsheet. Returned result type is: void. No arguments.
sheet-form-col	Form a column reference from it's integer value, e.g. 0 = A. Returned result type is: string. Argument types are: int.
sheet-format-cents	This routine changes the numeric format to cents (12.34). Returned result type is: void. No arguments.
sheet-format-dialog	Display the dialog for modifying the numeric format of the selected region. Returned result type is: void. No arguments.
sheet-format-dollar	Change the numeric format to dollars ($1234). Returned result type is: void. No arguments.
sheet-format-dollarcents	Change the numeric format to dollars and cents ($12.34). Returned result type is: void. No arguments.
sheet-format-dollarthousands	Change the numeric format to dollars and comma separated thousands ($1,234). Returned result type is: void. No arguments.
sheet-format-dollarthousandscents	Change the numeric format to dollars, comma separated thousands and cents ($1,234.56). Returned result type is: void. No arguments.
sheet-format-general	Use general numeric formatting (use as many decimal places as necessary). Returned result type is: void. No arguments.
sheet-format-general-dollar	Change the format to display the $ sign but to otherwise use general numeric formatting. Returned result type is: void. No arguments.

sheet-format-int	Round off to the nearest integer (1234). Returned result type is: void. No arguments.
sheet-format-string	Display the formula instead of the value. Returned result type is: void. No arguments.
sheet-format-thousands	Display the value using thousands (1,234). Returned result type is: void. No arguments.
sheet-format-thousandscents	Display the value using thousands and cents (1,234.56). Returned result type is: void. No arguments.
sheet-get-cell	Get the contents of some cell in the spreadsheet. Returned result type is: string. Argument types are: int, int.
sheet-get-cell-value	Get the value of some cell in the spreadsheet. If the cell is a formula, the value of the formula, is returned, not the formula itself. Returned result type is: string. Argument types are: int, int.
sheet-graph-create	Display the dialog for creating a new chart from a spreadsheet. Returned result type is: void. No arguments.
sheet-graph-create-silently	Creates a line graph with the default parameters and returns the name of the created graph. Returned result type is: string. No arguments.
sheet-graph-generate-one	Display the dialog for creating a new chart from a spreadsheet. Returned result type is: void. Argument types are: int.
sheet-graph-set-dataset-field	Set a field of the given dataset of the named graph to the given value. The first argument is the graph name, the second is the dataset index (0 through 5), the third is the field name (one of range, label, legend, dolines, dosym, or lblalign). The fourth argument is the value. Returned result type is: void. Argument types are: string, int, string, string.

sheet-graph-set-field	Set a field of the named graph to the given value. The first argument is the graph name, the second is the field name (one of name, autogenerate, type, title-line1, title-line2, x-axis-label, y-axis-label, x-range, grid, xmin, xmax, xincr, ymin, ymax, or yincr). The third argument is the value. Returned result type is: void. Argument types are: string, string, string.
sheet-home	Move the current cell to cell A1. Returned result type is: void. No arguments.
sheet-insert-cols	Insert empty columns into the spreadsheet in place of the selected region. Cells in the selected region and to the right will move over. Returned result type is: void. No arguments.
sheet-insert-rows	Insert empty rows into the spreadsheet in place of the selected region. Cells in the selected region and below will move down. Returned result type is: void. No arguments.
sheet-interactive-column-change	Change the width of a column using the mouse. Returned result type is: void. No arguments.
sheet-manual-compute	Set the recomputation style to manual (formulas are only recomputed when it is explicitly requested). Returned result type is: void. No arguments.
sheet-mouse-set-cursor	Set the cursor using the mouse. Returned result type is: void. No arguments.
sheet-move	Copy the spreadsheet on the clipboard into the selected region and adjust any formulas which referenced its old location to reference the new location. Returned result type is: void. No arguments.
sheet-move-cursor	Move the current cell of the spreadsheet. Returned result type is: void. Argument types are: int, int.
sheet-parse-col	Extract the starting column from a range string (e.g. A1). Returned result type is: int. Argument types are: string.

sheet-parse-height	Extract the height of a range from a string (e.g. A1:D1 = 4). Returned result type is: int. Argument types are: string.
sheet-parse-range	Return non-zero if the string is a legal range. Returned result type is: int. Argument types are: string.
sheet-parse-row	Extract the starting row from a range string (e.g. A1). Returned result type is: int. Argument types are: string.
sheet-parse-width	Extract the width of a range from a string (e.g. A1:C1 = 3). Returned result type is: int. Argument types are: string.
sheet-paste	Paste the spreadsheet from the clipboard, including underlying formulas, into the selected region. Returned result type is: void. No arguments.
sheet-pastevalues	Paste the values only of the cells in the spreadsheet on the clipboard into the selected region. Returned result type is: void. No arguments.
sheet-read-lotus	Read a file containing a Lotus worksheet. Returned result type is: void. No arguments.
sheet-read-text	Read a file containing an ASCII table. Columns are separated by tabs and each line is a separate row. Returned result type is: void. No arguments.
sheet-read-text-file	Read an ASCII table from the specified file. Columns are separated by tabs and each line is a separate row. Returned result type is: void. Argument types are: string.
sheet-ruling-apply	Apply the current selected ruling style to the selected region. Returned result type is: void. No arguments.
sheet-ruling-dialog	Display the dialog used for setting and clearing rulings. Returned result type is: void. No arguments.

sheet-select-area	Select an area of the spreadsheet using the mouse. Returned result type is: void. No arguments.
sheet-selected-height	Return the height of the selected area. Returned result type is: int. No arguments.
sheet-selected-width	Return the width of the selected area. Returned result type is: int. No arguments.
sheet-set-cell	Set the value of a cell in the spreadsheet. The first two arguments are the column and row index. The third argument is the cell contents. The fourth argument is a flag that means, if non-zero, to interpret the cell contents as a label. Otherwise, the spreadsheet will attempt to interpret the contents as a number or formula. Returned result type is: void. Argument types are: int, int, string, int.
sheet-set-current-cell	Set the value of the current cell in the spreadsheet. The first argument is the cell contents. The second argument is a flag that means, if non-zero, to interpret the cell contents as a label. Otherwise, the spreadsheet will attempt to interpret the contents as a number or formula. Returned result type is: void. Argument types are: string, int.
sheet-set-cursor	Set the current cell of the spreadsheet. Returned result type is: void. Argument types are: int, int.
sheet-set-selection	Set the selected area of the spreadsheet. The first argument is the top column, the second is the top row, the third is the width of the selected region and the fourth is the height of the selected region. The fifth argument is non-zero if the currently selected region should be cleared first. Returned result type is: void. Argument types are: int, int, int, int, int.
sheet-sort	Display the dialog for sorting the spreadsheet. Returned result type is: void. No arguments.

sheet-top-menu	Display the top-level spreadsheet menu. Returned result type is: void. No arguments.
sheet-width-change	Query the user to change the width of the selected columns. Returned result type is: void. No arguments.
sheet-write-lotus	Write the spreadsheet out in Lotus worksheet format. This routine is only partially implemented. Returned result type is: void. No arguments.
sheet-write-text	Write the spreadsheet out in a readable format. Returned result type is: void. No arguments.
sleep	Pause for some number of milliseconds. Returned result type is: void. Argument types are: int.
split-pane-across	Divide the current pane in half horizontally. Returned result type is: void. No arguments.
split-pane-down	Divide the current pane in half vertically. Returned result type is: void. No arguments.
strcat	Append the second string to the first and return the result. The input strings are not changed. Returned result type is: string. Argument types are: string, string.
string-find	Return the index of the location of the second string in the first string. Return -1 if it is not found. Returned result type is: int. Argument types are: string, string.
string-get	Extract part of a string. The first argument is the source string. The second is an index into it and the third is the number of characters to extract. Returned result type is: string. Argument types are: string, int, int.
string-to-char	Extract a character from a string. Returned result type is: char. Argument types are: string, int.

string-to-event-code	Convert the string representation of a key into its integer code, e.g. Ctrl-left-click. Returns -1 if the string is not valid. Returned result type is: int. Argument types are: string.
strlen	Return the length of the specified string. Returned result type is: int. Argument types are: string.
template-define	Make the current buffer a user-defined template with the given name. Returned result type is: void. Argument types are: string.
template-define-dialog	Make the current buffer a user-defined template. The user is queried for the filename for storing the template. Returned result type is: void. No arguments.
template-expand-name	Expand the given template name into its full path name. The template may be a normal file or a template in the user or system template area. Returned result type is: string. Argument types are: string.
template-instantiate-from-file	Instantiate a template from the specified file. Returned result type is: void. Argument types are: string.
template-keyboard-select	Display a dialog for selecting a template for composing a document. Returned result type is: void. No arguments.
text-add-item	Takes two arguments: the name of a text format, and a Boolean which is true if the format was automatically generated. Add an instance of the specified text format. Returned result type is: void. Argument types are: string, int.
text-add-nephew-item	Takes the name of a style as an argument, adds a paragraph with that style to the document, and then groups that paragraph with the current paragraph. Returned result type is: void. Argument types are: string.

text-add-uncle-item	Takes the name of a style as an argument, adds a paragraph with that style to the document, and then ungroups that paragraph. Returned result type is: void. Argument types are: string.
text-appear	Add a new instance of the default text style to the document and insert the character typed. Returned result type is: void. No arguments.
text-at-line-end	Returns non-zero if the cursor is at the end of the line. Returned result type is: int. No arguments.
text-at-line-start	Returns non-zero if the cursor is at the start of the line. Returned result type is: int. No arguments.
text-at-paragraph-end	Returns non-zero if the cursor is at the end of the paragraph. Returned result type is: int. No arguments.
text-at-paragraph-start	Returns non-zero if the cursor is at the start of the paragraph. Returned result type is: int. No arguments.
text-backward-character	Move the cursor back one character (or the number of characters specified by the repeat count). Returned result type is: void. No arguments.
text-backward-paragraph	Move the cursor to the start of the current block of text. If the cursor is at the start of the current block of text, then move the cursor to the start of the previous block of text. Returned result type is: void. No arguments.
text-backward-word	Move the cursor to the start of the current word. If the cursor is at the start of the current word, then move the cursor to the start of the previous word. Returned result type is: void. No arguments.
text-boldify	Display the text in the selected region in a bold face. If no region is selected, then display the next characters entered in a bold face. Returned result type is: void. No arguments.

text-bolditalicize	Display the text in the selected region in a bold italic face. If no region is selected, then display the next characters entered in a bold italic face. Returned result type is: void. No arguments.
text-capitalize	Capitalize the first character of the word that the cursor is in. Returned result type is: void. No arguments.
text-character	Return the character at the point. Returned result type is: char. Argument types are: int.
text-check	Returns non-zero if the point is set and is currently in a text object. Returned result type is: int. No arguments.
text-clean-clear	Clear the selected region without checking any special conditions. Returned result type is: void. No arguments.
text-cleanup-fonts	Remove any font changes in the selected region. Returned result type is: void. No arguments.
text-clear-mark	Remove the mark from the text passage. Returned result type is: void. No arguments.
text-clear-paragraph	Delete the block of text containing the cursor. Returned result type is: void. No arguments.
text-clear-point	This causes no object to be selected. Sets the current point (returned by current-point) to 0. Returned result type is: void. No arguments.
text-clear-region	This causes no region to be selected. Sets the current region (returned by current-region) to be 0. Returned result type is: void. No arguments.
text-compress	Cause the characters in the selected region to be compressed by two points. Returned result type is: void. No arguments.

text-compute-pagewidth	Recompute the page width based on the document page-width variable. Returned result type is: void. No arguments.
text-copy-chars	Return a string containing the characters in the current region. Returned result type is: string. No arguments.
text-copy-region	Copy the selected text region on to the clipboard without removing it from the document. Returned result type is: void. No arguments.
text-create-format	Displays a dialog to create a new style based on an old one. Returned result type is: void. No arguments.
text-current-format	Return the name of the style used to format the current paragraph. Returned result type is: string. No arguments.
text-cursorkey	Handle the escape codes generated by the Sun cursor keys. Returned result type is: void. No arguments.
text-cut-paragraph	Delete the block of text containing the cursor and place it on the clipboard. Returned result type is: void. No arguments.
text-cut-region	Delete the selected text region from the document and place it on the clipboard. Returned result type is: void. No arguments.
text-decrement-pointsize	Reduce the pointsize of the selected region. Returned result type is: void. No arguments.
text-delete-back	Delete the character before the cursor. Returned result type is: void. No arguments.
text-delete-back-word	Delete characters to the beginning of the word that the cursor is in. If the cursor is at the start of a word, delete the word before the cursor. Returned result type is: void. No arguments.
text-delete-extra-white-space	Delete all but a single space in the region surrounding the cursor. Returned result type is: void. No arguments.

text-delete-forward	Delete the character after the cursor. Returned result type is: void. No arguments.
text-delete-forward-word	Delete characters to the end of the word that the cursor is in. If the cursor is at the end of a word, delete the word following the cursor. Returned result type is: void. No arguments.
text-delete-white-space	Delete any white space surrounding the cursor. Returned result type is: void. No arguments.
text-describe-object	Provide information about the format of the block of text in which the cursor resides. Returned result type is: void. No arguments.
text-deselect	Deselect any selected text region. Returned result type is: void. No arguments.
text-doubleline-off	Remove double underlines in the selected region. Returned result type is: void. No arguments.
text-doubleline-on	Double underline the selected region. Returned result type is: void. No arguments.
text-down-line	Move the cursor down one line. The cursor is placed as nearly as possible immediately below its current position. If the next line is shorter than the line the cursor is currently in, then the cursor is placed at the end of the line. Returned result type is: void. No arguments.
text-edit-format	Edit the named text format either globally or locally by assigning the specified value to the specified attribute. Returned result type is: void. Argument types are: string, int, string, string.
text-edit-global-format	Edit a text format. This command first displays a form that allows the user to select the format to change and then displays a form

with all the attributes of that format. This command makes global changes to the format; i.e., it changes all instances of text blocks in that format as well as new text blocks added in that format. Returned result type is: void. No arguments.

text-edit-local-format Edit the format of the selected region of text. If the selected region contains more than one format style, then a form is displayed that allows the user to specify which format to edit. This command displays a form with all of the attributes of the format and the user can edit the format by making the appropriate changes in the form. This command makes local changes to the format; it changes only the text in the selected region. Returned result type is: void. No arguments.

text-edit-named-global-format Edit the named text format. This command displays a form with all the attributes of the specified format. This command makes global changes to the format; i.e. it changes all instances of text blocks in that format as well as new text blocks added in that format. Returned result type is: void. Argument types are: string.

text-edit-named-local-format Edit the named format. This command displays a form with all of the attributes of the format and the user can edit the format by making the appropriate changes in the form. This command makes local changes to the format; it changes only the text in the selected region. If there is no text in the specified format in the selected region, then nothing is done. Returned result type is: void. Argument types are: string.

text-end-of-buffer Move the cursor to the end of the text in the buffer. Returned result type is: void. No arguments.

text-end-of-line Move the cursor to the end of the line that it is in. Returned result type is: void. No arguments.

text-end-of-paragraph Move the cursor to the end of the block of text that it is in. Returned result type is: void. No arguments.

text-end-of-sentence	Move the cursor to the end of the sentence containing the cursor. Returned result type is: void. No arguments.
text-enter-from-bottom	This will cause the text cursor to appear in the bottom-most block of text visible. Text operations may then be invoked from the keyboard. Returned result type is: void. No arguments.
text-enter-from-top	This will cause the text cursor to appear in the top-most block of text visible. Text operations may then be invoked from the keyboard. Returned result type is: void. No arguments.
text-enumerate	Make the selected region into an enumeration. Returned result type is: void. No arguments.
text-expand	Expand the text in the selected region. Returned result type is: void. No arguments.
text-extend-region	Extend the selected region with the mouse. Returned result type is: void. No arguments.
text-filter-region-dialog	Write the selected region as a complete document to a temporary file and query the user for a command line to invoke a tool on that file. Returned result type is: void. No arguments.
text-filter-region-to-tool	Write the selected region as a complete document to a temporary file and invoke the tool with the specified command line on that file. Returned result type is: void. Argument types are: string.
text-find-name	Takes the name of a formatting style and searches for it in the document. If found, the cursor will be set to the start of first paragraph found with that style. Returned result type is: void. Argument types are: string.
text-find-rev-name	Takes the name of a formatting style and searches backwards for it in the document. If found, the cursor will be set to the end of the first paragraph found with that style. Returned result type is: void. Argument types are: string.

text-find-style-again	Find the last text style which was specified using the top level Find command. Returned result type is: void. No arguments.
text-forward-character	Move the cursor foward one character (or the number of characters specified by the repeat count). Returned result type is: void. No arguments.
text-forward-paragraph	Move the cursor to the end of the current block of text. If the cursor is at the end of the current block of text, then move the cursor to the end of the next block of text. Returned result type is: void. No arguments.
text-forward-word	Move the cursor to the end of the current word. If the cursor is at the end of the current word, then move the cursor to the end of the next word. Returned result type is: void. No arguments.
text-free-excursion	Clear a saved excursion without restoring the previous context. Returned result type is: void. No arguments.
text-generate-table-of-contents	Generate the table of contents for the given buffer. Returned result type is: string. Argument types are: buffer.
text-get-selection	Insert the selection into the text buffer. Returned result type is: void. No arguments.
text-global-replace-dialog	Display the text search form with the operation initialized to global replace. Global replace is used to replace all instances of the specified string with another string. Returned result type is: void. No arguments.
text-group	If no region is selected, group the object in which the cursor resides with the object above it. Returned result type is: void. No arguments.
text-group-by-name	Group the selected region into the specified formatting environment which must be a list type environment. Returned result type is: void. Argument types are: string.

text-increment-pointsize	Display the text in the selected region in the next largest font size. If there is no selected region, then make the font change for new text. Returned result type is: void. No arguments.
text-initialize-keymap	The keymap for the text media type. Returned result type is: void. No arguments.
text-insert-character	Insert the specified character at the cursor position. Returned result type is: void. Argument types are: char.
text-insert-partial-string	Takes two arguments: a string to insert and the number of characters to insert. Inserts the specified number of characters from the string at the cursor. Returned result type is: void. Argument types are: string, int.
text-insert-space	Insert a space. Returned result type is: void. No arguments.
text-insert-string	Insert the specified string at the cursor position. Returned result type is: void. Argument types are: string.
text-is-between	Returns the first object if the current object is text and the mouse lies between two text objects, otherwise it returns zero. Returned result type is: object. No arguments.
text-is-empty-paragraph	Returns true if the text is in an empty paragraph. Returned result type is: int. No arguments.
text-is-whitespace	Returns true if the character following the cursor is a space or tab. Returned result type is: int. Argument types are: int.
text-italicize	Display the text in the selected region in an italic face. If no region is selected, then display the next characters entered in an italic face. Returned result type is: void. No arguments.
text-itemize	Make the selected region into an itemization. Returned result type is: void. No arguments.

text-kill-line	Remove the text from the cursor position to the end of the line; the text removed is put on the clipboard and can be re-inserted with the text-paste command. Returned result type is: void. No arguments.
text-line-to-bottom	Scroll the viewport so the line containing the cursor is moved to the bottom of the pane. Returned result type is: void. No arguments.
text-line-to-top	Scroll the viewport so the line containing the cursor is moved to the top of the pane. Returned result type is: void. No arguments.
text-mark-paragraph	Select the text paragraph in which the cursor resides. Returned result type is: void. No arguments.
text-modify-font-baseline	Increment or decrement the baseline of the text in the selected region by the given number of points. If there is no selected region, then make the font change for new text. Returned result type is: void. Argument types are: int.
text-modify-font-face	Modify the font face of the text in the selected region. If no region is selected then display the next text entered in the specified face. Returned result type is: void. Argument types are: int.
text-modify-font-family	Change the font family of the selected region. If no region is selected, choose the font family for the next characters entered. A name completion dialog appears in which the user may specify the desired font family. Returned result type is: void. Argument types are: string.
text-modify-font-pointsize	Increment or decrement the point size of the font of the selected region by the given number of points. If there is no selected region, then make the font change for new text. Returned result type is: void. Argument types are: int.

text-newline	Start a new line of text. If the cursor is in a paragraph with word-wrapping on, then the current paragraph is split into two. If word-wrapping is off, a newline is inserted into the current paragraph. Returned result type is: void. No arguments.
text-newline-check	Check if the cursor is in an empty paragraph in a list and do the proper ungrouping if that is the case. Otherwise, this routine performs the same function as text-newline. Returned result type is: void. No arguments.
text-open-line	Start a new line at the cursor position. Returned result type is: void. No arguments.
text-paste	Insert the contents of the clipboard at the cursor location. Returned result type is: void. No arguments.
text-paste-document	Insert the specified document into the current buffer at the cursor location. Returned result type is: void. Argument types are: document.
text-paste-from-file	Paste text from a named file into the current point in the document in the named format. The remaining arguments are paste with or without leading or trailing spaces and preserve newlines or replace them with spaces. Returned result type is: void. Argument types are: string, string, int, int, int.
text-query-replace-dialog	Display the text search form with the operation initialized to query replace. Query replace is used to replace all instances of the specified string with another string. The user is given the option to confirm or abort each replacement. Returned result type is: void. No arguments.
text-quote-next-character	Read the next character typed and insert it into the text. This command is used to insert characters that would otherwise be interpreted as editing characters, in other words, all the characters listed in the keybinding currently in effect. Returned result type is: void. No arguments.

text-region-size	Return the number of characters in the selected region. Returned result type is: int. No arguments.
text-remove-local-modifications	Remove any local style changes which have been applied to the selected region. Returned result type is: void. No arguments.
text-restore-excursion	Restore the old point and mark after an excursion. Returned result type is: void. No arguments.
text-romanize	Display the text in the selected region in a roman face. If no region is selected, then display the next characters entered in a roman face. Returned result type is: void. No arguments.
text-save-excursion	Save the current point and mark so it can be restored later. Returned result type is: void. No arguments.
text-scroll-to-start-of-buffer	Make the beginning of the current buffer visible at the top of the current pane. Returned result type is: void. No arguments.
text-search	Search for the specified string in the specified region and return true if the string is found and false otherwise. The arguments are the search string, the region to search and whether or not to fold case. Returned result type is: int. Argument types are: string, region, int.
text-search-forward-dialog	Display the text search form with the operation initialized to forward search. Returned result type is: void. No arguments.
text-search-forward-incrementally	Do an incremental forward search. The user is queried for the string and the document is searched forward from the cursor position to the end of the document. Returned result type is: void. No arguments.

text-search-incrementally	Takes an argument which is 0 to search forward from the current position to the end of the document and 1 to search backward from the current position to the start of the document. Does an incremental search. Returned result type is: void. Argument types are: int.
text-search-reverse-dialog	Display the text search form with the operation initialized to reverse search. Returned result type is: void. No arguments.
text-search-reverse-incrementally	Do an incremental reverse search. The user is queried for the string and the document is searched backward from the cursor position to the start of the document. Returned result type is: void. No arguments.
text-select-add	Add a block of text in a selected style. A name completion dialog appears in which the user may specify the desired style. Returned result type is: void. No arguments.
text-select-add-nephew	Display a scrolling list of styles and then add a paragraph formatted according to the selected style to the document. It then groups the new paragraph with the current paragraph. Returned result type is: void. No arguments.
text-select-add-uncle	Display a scrolling list of styles and then add a paragraph formatted according to the selected style to the document. It then ungroups the new paragraph from the current paragraph. Returned result type is: void. No arguments.
text-select-again	Re-execute the last selection routine. Returned result type is: void. No arguments.
text-select-current-line	Select the line containing the cursor. If this is invoked multiple times, it will extend the selected region by a line each time. Returned result type is: void. No arguments.

text-select-current-list	Select the list containing the cursor. If this is invoked multiple times, it will extend the selected region by the enclosing list each time. Returned result type is: void. No arguments.
text-select-current-paragraph	Select the block of text containing the cursor. If this is invoked multiple times, it will extend the selected region by a block each time. Returned result type is: void. No arguments.
text-select-current-sentence	Select the sentence containing the cursor. If this is invoked multiple times, it will extend the selected region by a sentence each time. Returned result type is: void. No arguments.
text-select-current-word	Select the word containing the cursor. If this is invoked multiple times, it will extend the selected region by a word each time. Returned result type is: void. No arguments.
text-select-font-family	Change the font family of the selected region. If no region is selected, choose the font family for the next characters entered. A name completion dialog appears in which the user may specify the desired font family. Returned result type is: void. No arguments.
text-select-this-word	Select the word that the cursor is in. If the cursor is at the first character in the word or immediately following the last character in the word, then that word is selected. If the cursor is not in a word, then the next word (if it exists) is selected. Returned result type is: void. No arguments.
text-self-insert	Insert the character typed at the current cursor position. Returned result type is: void. No arguments.
text-set-character	Replace the character following the cursor with the specified character. Does not update the display. Returned result type is: void. Argument types are: char.
text-set-cursor	Put the cursor at the mouse location. Returned result type is: void. No arguments.

text-set-mark	Set the mark at the current cursor position. Returned result type is: void. No arguments.
text-set-point	Set the cursor to the specified text location in the document. Returned result type is: void. Argument types are: marker.
text-set-pointsize	Set the point size of the font of the selected region to the given value. If there is no selected region, then make this change for new text. Returned result type is: void. Argument types are: int.
text-set-region	Put the cursor at the mouse position and then track the mouse and extend the selected region until the mouse is released. Returned result type is: void. No arguments.
text-set-selection	Set the selected region to the specified region. Returned result type is: void. Argument types are: region.
text-split	Divide the current block into two blocks at the current cursor position. Returned result type is: void. No arguments.
text-split-check	Divide the current block of text into two blocks at the current cursor position. Returned result type is: void. No arguments.
text-split-in	Like text-split, except it then groups the two split blocks. This is useful when you want to add another point under an enumeration. Returned result type is: void. No arguments.
text-split-list	Split a list at the cursor into two lists. The item in which the cursor resides becomes the last item in the first list. Returned result type is: void. No arguments.
text-split-out	Like text-split, except it then ungroups the two split blocks. This is useful to get out of a grouped item or at the end of a list. Returned result type is: void. No arguments.

text-start-of-buffer	Move the cursor to the start of the text in the buffer. Unlike text-scroll-start-of-buffer, this command makes the first text in the buffer visible. Returned result type is: void. No arguments.
text-start-of-line	Move the cursor to the start of the line that it is in. Returned result type is: void. No arguments.
text-start-of-line-absolute	Move the cursor to the start of the line. If the cursor is in a list, move it to the start of any tag which is also on this line (text-start-of-line will not move into an uneditable tag). Returned result type is: void. No arguments.
text-start-of-paragraph	Move the cursor to the start of the block of text it is in. Returned result type is: void. No arguments.
text-start-of-sentence	Move to the start of the sentence containing the cursor. Returned result type is: void. No arguments.
text-strikeout-off	Remove strikeout (the font style) in the selected region. Returned result type is: void. No arguments.
text-strikeout-on	Strikeout (which is a font style) the selected region. Returned result type is: void. No arguments.
text-stuff-selection	Insert the selection into the text buffer. Returned result type is: void. No arguments.
text-subscript	Make the characters in the selected region a subscript. Returned result type is: void. No arguments.
text-superscript	Make the characters in the selected region a superscript. Returned result type is: void. No arguments.

text-switch-format	Change the format of the selected region of text. A dialog is displayed that allows the user to select the new format. If the selected region contains text formatted in more than one style, then a dialog is displayed that allows the user to select the format to change. Returned result type is: void. No arguments.
text-switch-named-format	Switch the style of text objects in the selected region. The first argument is the name of the style to switch from, the second argument is the name of the style to switch to. Returned result type is: void. Argument types are: string, string.
text-switch-point-and-mark	Switch the location of point and mark. Returned result type is: void. No arguments.
text-tab	Insert a tab. Returned result type is: void. No arguments.
text-tolower	Make the text in the selected region lowercase. Returned result type is: void. No arguments.
text-top-menu	Display the top level text menu. Returned result type is: void. No arguments.
text-toupper	Make the text in the selected region uppercase. Returned result type is: void. No arguments.
text-transpose-characters	Transpose the two characters preceding the cursor. Returned result type is: void. No arguments.
text-underline-off	Remove underlines in the selected region. Returned result type is: void. No arguments.
text-underline-on	Underline the selected region. Returned result type is: void. No arguments.
text-undo-subscript	Make subscripted characters appear at normal size and baseline. Returned result type is: void. No arguments.

text-undo-superscript	Make superscripted characters appear at normal size and baseline. Returned result type is: void. No arguments.
text-ungroup	Make all of the items in the selected region of a list be separate items. Returned result type is: void. No arguments.
text-up-line	Move the cursor up one line. The cursor is placed as nearly as possible immediately above its current position. If the previous line is shorter than the line the cursor is currently in, then the cursor is placed at the end of that line. Returned result type is: void. No arguments.
text-write-selection	The Sun Put command. Returned result type is: void. No arguments.
toggle-object-outline	Toggle whether the outline box of the current object is displayed. Returned result type is: void. No arguments.
transliterate-region	Transliterate a region of the document from one script to another. Returned result type is: void. No arguments.
unset-buffer-variable	Unset the value of the given buffer-specific variable. Returned result type is: void. Argument types are: string.
unset-buffer-variable-dialog	Unset the value of a buffer-specific variable. Returned result type is: void. No arguments.
unset-document-variable	Unset the value of the given document-specific variable. Returned result type is: void. Argument types are: string.
unset-document-variable-dialog	Unset the value of a document-specific variable. Returned result type is: void. No arguments.
unset-global-variable	Unset the value of the given global variable. Returned result type is: void. Argument types are: string.

unset-global-variable-dialog	Unset the value of a global variable. Returned result type is: void. No arguments.
update-display	Force the screen to be updated so that it reflects the current state of the document. Normally, the editor will defer updating the screen while editing actions are taking place or extension language commands are being executed. Returned result type is: void. No arguments.
visit-document	Read the specified file in a new buffer. Returned result type is: void. Argument types are: string.
visit-document-dialog	Read a document in a new buffer asking the user for the file name. Returned result type is: void. No arguments.
visit-file	See visit-document. Returned result type is: void. Argument types are: string.
visit-file-dialog	See visit-document-dialog. Returned result type is: void. No arguments.
visit-new-document	Read the specified file in a new buffer. Returned result type is: void. Argument types are: string.
vocoder-release	Release the audio hardware so it will be available to another client Returned result type is: void. No arguments.
vocoder-shutdown	Release the audio hardware so it is unavailable to another client Returned result type is: void. No arguments.
write-buffer-as-text	Write the specified buffer as text. The user is queried for the name of the text file. Returned result type is: void. Argument types are: buffer.
write-buffer-to-file	Write the specified buffer to the specified file. Returned result type is: void. Argument types are: buffer, string.

write-buffer-to-file-as-text	Write the specified buffer to the specified file as text. Returned result type is: void. Argument types are: buffer, string.
write-buffer-to-stdout-as-text	Write the specified buffer as text to the standard output. Returned result type is: void. Argument types are: buffer.
write-current-buffer	Write the current buffer. The user is queried for the filename. Returned result type is: void. No arguments.
write-current-buffer-as-text	Write the current buffer as text. Returned result type is: void. No arguments.
write-current-buffer-to-stdout-as-text	Write the current buffer as text to the standard output. Returned result type is: void. No arguments.
write-region-as-text	Write the selected region as text. The user is queried for the name of the text file. Returned result type is: void. No arguments.
write-region-to-file	Write the selected region to the given file. Returned result type is: void. Argument types are: string.
write-region-to-file-as-text	Write the selected region as text to the given file. Returned result type is: void. Argument types are: string.
write-region-to-process	Filter the selected region. Returned result type is: void. No arguments.
write-region-to-stdout	Write current region as text to standard output. Returned result type is: void. No arguments.
write-string-to-stderr	Write a string to the standard error channel. Returned result type is: void. Argument types are: string.
write-string-to-stdout	Write the specified string followed by a newline to standard output. Returned result type is: void. Argument types are: string.

Index

BBN Software Products
A Division of Bolt Beranek and Newman Inc.

Your opinions about our products and services are important to us. Please use this self-addressed, stamped form to help us evaluate this manual. Your response to the questions listed below will help us assess our current documentation and plan ways to improve it to meet your needs. Just fill in your responses, detach this sheet, fold and tape it, and drop it in the mail. We'll pay careful attention to what you say!

Manual Title: *BBN/Slate Customizing Manual (Release 1.1)*

List your operating system: ______________________ Date: ____/____/____

General Comments:	Yes	Somewhat	No
Do you like this manual?	☐	☐	☐
Is it easy to read?	☐	☐	☐
Is the order of topics easy to follow?	☐	☐	☐
Is the information accurate?	☐	☐	☐
Can you easily find the information you need?	☐	☐	☐
Do the examples and illustrations help?	☐	☐	☐
Can you easily apply the described features to your own needs?	☐	☐	☐
Do you use on-line help?	☐	☐	☐

Comments and Suggestions (include chapter or page number where applicable):

How do you primarily use this manual?

☐ Introduction ☐ Tutorial ☐ Reference

☐ **Check here to receive information about our Education Services.**

☐ **Check here to receive the *BBN Software Quarterly* newsletter.**

Please fill in the following information:

Name ______________________	Title ______________________
Company ______________________	Street ______________________
City ______________________	State ______________________
Zip Code ______________________	Phone ______________________

Cut or fold along dotted line

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

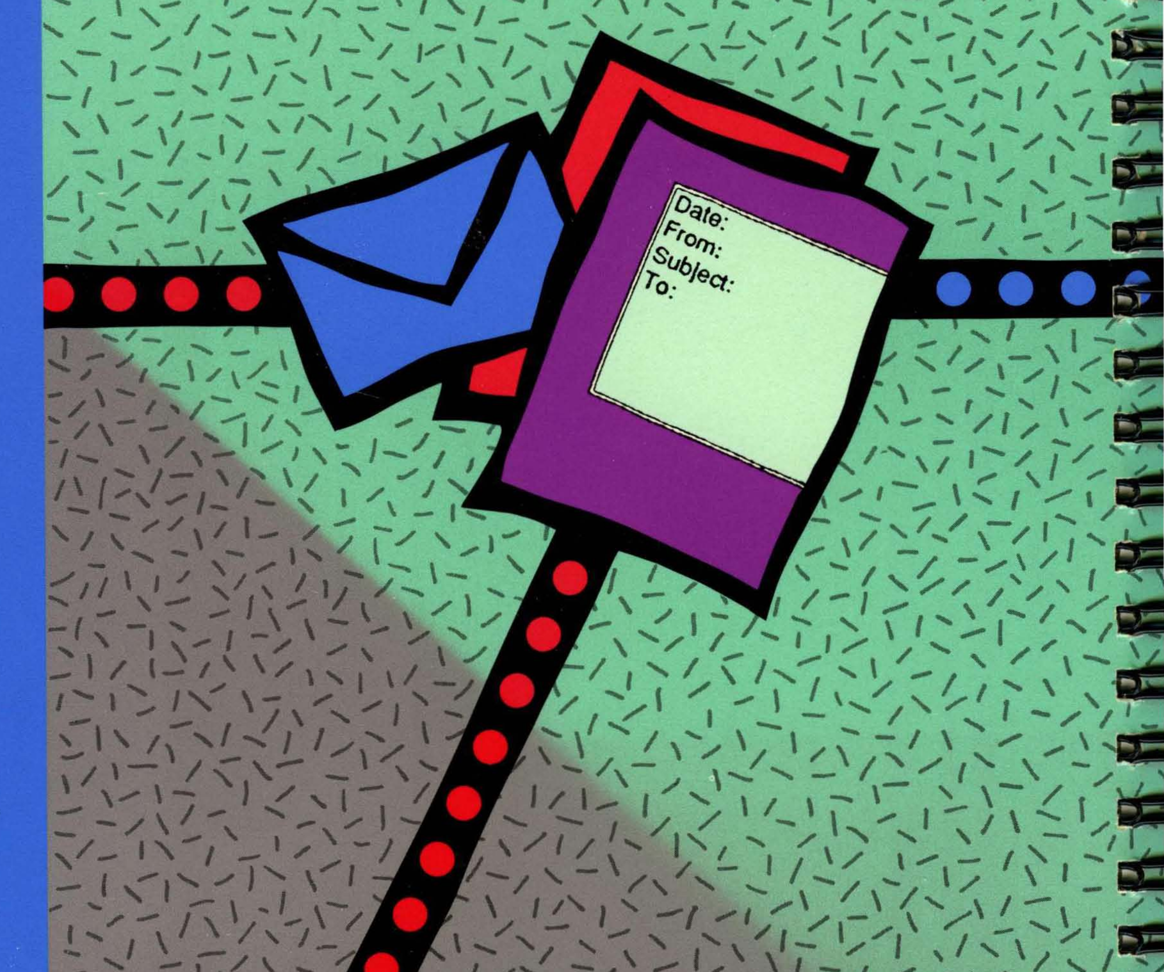
BUSINESS REPLY ENVELOPE

FIRST CLASS PERMIT NO. 36450 CAMBRIDGE, MA

POSTAGE WILL BE PAID BY ADDRESSEE

BBN Software Products
A Division of Bolt Beranek and Newman Inc.
10 Fawcett Street
Cambridge, MA 02138

Attn: Manager, Documentation



2523012

BBN Software Products
A Division of Bolt Beranek and Newman Inc.