8. USER INTERFACES

A single elvis binary can be compiled to support multiple user interfaces. For example, under UNIX elvis can be compiled to have a graphical user interface when run under X-windows, a termcap interface for use on most text terminals, an "open" interface for use on any terminal, and a "quit" interface for running scripts. Here's a comprehensive list of the user interfaces which *may* be available in your copy of elvis:

* x11 - a graphical interface under Unix and OS/2.
* windows - a graphical interface under Win32.
* termcap - a full-screen text-based interface.
* vio - OS/2-specific version of the termcap interface.
* open - a simpler text-based interface.
* quit - a pseudo-interface for non-interactive editing.
* script - a pseudo-interface for running scripts.

The exact list of available user interfaces will vary from one system to another. You can make elvis output a list of available interfaces by running "elvis -G?". This will also show you which interface elvis will use by default.

Elvis chooses the default user interface at run time by testing each user interface in turn, starting with the most desirable, and working its way down the list until it finds one that appears to be supported in the current environment. For example, if you're using elvis on a text terminal under UNIX, then elvis will bypass the "x11" interface because X-windows doesn't work on text terminals, and then elvis will find that the "termcap" interface would work, so that'll be the default.

If you don't want to use the default user interface, you can specify which interface to use via the **-G** *gui* command-line flag.

8.1 X11 Interface

The x11 interface is used under X-Windows on UNIX or OS/2 systems. (See the OS/2 section of the OS chapter for a description of what you need to run X11 under OS/2.) Subsections here are command line flags, the mouse, the toolbar, resources, keys, and icons.

The x11 interface provides a scrollbar and mouse support, and allows you to select which fonts to use. There is also a configurable toolbar. Buttons on that toolbar can even be configured to use pop-up dialog windows.

The x11 interface reads app-defaults (as listed below) but those are only used to provided default values for options and colors. You can override them with :set and :color commands. The x11-specific options are described in the options chapter.

8.1.1 X11 Command-line Flags

To specify a normal font, use **-font** *fontname* or **-fn** *fontname.* Proportional fonts are not supported. If you don't specify a normal font, then elvis will use a font named "fixed" by default.

To specify a bold font, use **-fb** *fontname.* The specified font should have the same character cell size as the normal font, but elvis does not verify this. If you don't specify a bold font, then elvis will fake it by smearing the normal font rightward one pixel.

To specify an italic font, use **-fi** *fontname.* The specified font should have the same size character cell as the normal font, but elvis does not verify this. If you don't specify an italic font, then elvis will fake it by sliding the top half of the normal font rightward one pixel.

The **-fc** *fontname* flag can be used to specify the font to be used for controls -- currently just the toolbar and statusbar, but eventually elvis will offer a scrollbar too. If you don't specify a control font, then elvis will use a font named "variable" by default.

If you want to use Courier fonts, there is a shortcut: **-courier** *size* will use the normal, bold, and italic versions of the courier font in the requested size.

You can also specify the foreground and background colors with **-fg** *color* and **-bg** *color,* respectively. All standard X color names are supported.

Elvis has a built-in icon, which is generally a good thing. Some window managers won't allow you to assign a new icon to a program that has a built-in one, so elvis has a **-noicon** flag which disables the built-in icon.

Elvis also supports the **-geometry** *WxH+X+Y* flag for specifying the size and/or position of the first window. The size is specified in characters, and the default size is 80x34. There is no default position.

The **-fork** option causes elvis to fork a new process, so you get a new shell prompt immediately.

The **-client** option causes elvis to look for an already-running elvis process on the same X server, and if there is one, send the new arguments to it. This causes the old elvis process to create new windows for file arguments. The new elvis process then exits, leaving the old one to do the real work. If there is no elvis process already running, then **-client** will act like **-fork** so that either way, you get a new shell prompt immediately.

You can change elvis' defaults by editing the elvis.ini or ~/.exrc file. You can use the :color command to assign colors to various fonts, and the cursor and scrollbar/toolbar. Most other aspects are controlled via options.

8.1.2 X11 Mouse

I've tried to reach a balance between the mouse behavior of **xterm(1)** and what makes sense for an editor. To do this right, elvis has to distinguish between clicking and dragging.

Dragging the mouse always selects text. Dragging with button 1 pressed (usually the left button) selects characters, dragging with button 2 (the middle button) selects a rectangular area, and dragging with button 3 (usually the right button) selects whole lines. These operations correspond to elvis' v, ^V, and V commands, respectively. When you release the button at the end of the drag, the selected text is immediately copied into an X11 cut buffer, so you can paste it into another application such as xterm. The text remains selected, so you can apply an operator command to it.

Clicking button 1 cancels any pending selection, and moves the cursor to the clicked-on character. Clicking button 3 moves the cursor without cancelling the pending selection; you can use this to extend a pending selection.

Clicking button 2 "pastes" text from the X11 cut butter. If you're entering an ex command line, the text will be pasted into the command line as though you had typed it. If you're in visual command mode or input mode, the text will be pasted into your edit buffer. When pasting, it doesn't matter where you click in the window; elvis always inserts the text at the position of the text cursor.

Double-clicking button 1 simulates a ^] keystroke, causing elvis to perform tag lookup on the clicked-on word. If elvis happens to be displaying an HTML document, then tag lookup pursues hypertext links so you can double-click on any underlined text to view the topic that describes that text. Double-clicking button 3 simulates a ^T keystroke, taking you back to where you did the last tag lookup.

8.1.3 Toolbar

The X11 interface supports a user-configurable toolbar. The toolbar is enabled by default; you can disable it in your ~/.exrc file by adding a "set notoolbar" command.

If enabled, you will find that the default toolbar already has some buttons defined. You can use the :gui command to reconfigure the toolbar. The following commands are supported:

:gui
This displays the :gui commands which were used to set up all toolbar buttons.

:gui *label*
This displays the :gui commands which were used to set up the toolbar button that has the given label.

:gui newtoolbar
This deletes all buttons from the toolbar.

:gui ~*label*
This deletes a single, specific button from the toolbar.

:gui gap
This leaves a small gap between the previous button and the following button.

:gui *label* : *excommand*
This creates a button named *label*. Whenever you click the button, the *excommand* will be interpreted as an ex command line. The *label* can begin with any non-whitespace character. The remaining characters can be letters, digits, or another instance of the initial character.

```
:gui Help:help
```

NOTE: If you want to have more than one line's worth of ex commands associated with a toolbar button, then you might consider defining an alias.

:gui *label* **=** *condition*
Normally buttons are drawn as though sticking out; this command gives you a way to make them selectively appear to be stuck in. The *condition* is a C-like expression. When it is true, the button will be drawn "sticking in". When it is false, the button will be drawn "sticking out". The button behaves exactly the same either way.

```
:gui List:set neglist
:gui List=list
```

:gui *label* **?** *condition*
This gives you a way to selectively disable the button. The *condition* is a C-like expression. When it is true, the button behaves as normal; when it is false, the button ignores any mouse clicks. Also, buttons which are disabled this way are displayed as being "flat", instead of the normal 3D shading that makes them appear to stick out or in.

```
:gui Quit:q
:gui Quit?!modified
```

NOTE: The *condition* expressions are re-evaluated after nearly every input event. If you have many complex expressions, this may take a noticeable amount of time. With the default toolbar, elvis seems to slow down by about 20%. Toolbar buttons which don't use *condition* expressions have no such overhead.

:gui *label* " *description*
Add a one-line description to the button. The description is shown on the statusbar when the button is pressed. It is also displayed on pop-up dialogs, as described below.

```
:gui Quit"Close this window, and maybe exit elvis
```

:gui *label* ; ;
:gui *label* ; *option* ; *...*
:gui *label* ; "*prompt*" **(***type***)** *option* **=** *value* ; *...*
This allows you to define a pop-up dialog for a given toolbar button. When you click on the toolbar button, the dialog appears immediately. You can then edit some values, and then hit the [Submit] button to store the edited values into options and run the toolbar button's ex command (if any), or hit the [Cancel]

button to do nothing.

The simplest dialog is specified by giving just a pair of semicolons after the label. This dialog will have no editable fields, but it still shows the [Submit] and [Cancel] buttons, so it is a handy way to ask for confirmation before doing something.

But usually you'll give a semicolon-delimited list of options after the toolbar button's label. The dialog will then allow you to edit those options. When you hit the [Submit] button on that dialog window, elvis will store the values into the options before running the ex command.

The default prompt for each option is its name. If you precede the option name with a quoted string, then the string is used for the prompt instead.

You can also override the option's data type. The default type for each option is the same type used by the :set command. You can override that by placing one of the following before the option name:

TYPE	**MEANING**
(oneof *list*)	Allow any single value from the space-delimited *list*
(boolean)	Same as (oneof true false)
(number)	Allow any number
(number *m*:*n*)	Allow number between *m* and *n*, inclusive
(string)	Allow any string
(file)	Allow any string, but use the **Tab** key for file name completion
(locked)	Display it, but don't allow any editing

The default value for each option is the option's actual value at the time the dialog pops up. You can override that by appending an '=' followed by an expression for some other value. Note that the option itself isn't changed unless/until you hit the [Submit] button.

NOTE: The user options "a" through "z" are useful for inputing (via the dialog) and storing temporary values. You'll almost certainly want to override the prompt and type of those options.

NOTE: Because the edited option values are stored before the toolbar button's ex command is executed, the ex command can access the options' values via commands such as :eval. Also, since elvis always subjects file names to evaluation via the simpler expression syntax, you can don't need to use :eval to expand file names. The following shows one useful example of this:

```
:gui Split;"File to load:" (file) f = filename
:gui Split:split (f)
```

NOTE: If you just want to use the dialog for adjusting options, and don't need to run an ex command afterward, then you can simply omit the ex command definition for that toolbar button. For example, the following is sufficient for editing the display options:

```
:gui Display Options; list; number; wrap; sidescroll
```

NOTE: You can display a non-editable line of text in the dialog by using ="string" without giving any option name. Here's an example:

```
:gui Save;"Save as:"(file)f=basename(file);="In current directory!"
:gui Save:w (f)
```

8.1.4 Resources

Elvis uses the following X resources. The resource values can be overridden by command-line flags, or by explicit :set or :color commands in the initialization scripts.

RESOURCE CLASS (name is lowercase of class)	TYPE	DEFAULT VALUE	PARTIAL EX COMMAND
Elvis.Toolbar	Boolean	True	set toolbar
Elvis.Statusbar	Boolean	True	set statusbar
Elvis.Font	Font	fixed	set normalfont=
Elvis.Geometry	Geometry	80x34	set firstx= xrows=
Elvis.Foreground	Color	black	color normal
Elvis.Background	Color	gray90	color normal
Elvis.MultiClickTimeout	Timeout	3	set dblclicktime=
Elvis.Control.Font	Font	variable	set controlfont=
Elvis.Cursor.Foreground	Color	red	color cursor
Elvis.Cursor.Selected	Color	red	color cursor
Elvis.Cursor.BlinkTime	Timeout	3	set blinktime=
Elvis.Tool.Foreground	Color	black	color tool
Elvis.Tool.Background	Color	gray75	color tool
Elvis.Scrollbar.Foreground	Color	gray75	color scrollbar
Elvis.Scrollbar.Background	Color	gray60	color scrollbar
Elvis.Scrollbar.Width	Number	11	set scrollbarwidth=
Elvis.Scrollbar.Repeat	Timeout	4	set scrollbartime=
Elvis.Scrollbar.Enabled	Boolean	True	set scrollbar

The "Timeout" type gives a time value, in tenths of a second.

For example, if your X resources database contains the line "elvis.font: 10x20" then the default text font would be "10x20". This value would therefore be used if the normalfont option was unset.

The method for changing a resource may vary from one X server to another. Typically, you would edit a file named ~/.Xdefaults, and then run "xrdb -merge ~/.Xdefaults". (xrdb is part of the standard X distribution.)

8.1.5 X11 Keys

If there is a standard way to map a Keysym value into a text string, then elvis will use it. This means that when you hit the <m> key, you get an "m" character. Function keys and cursor keys have no standard translation, so elvis converts them to a ^K character followed the Keysym binary value, expressed as four hex digits.

You can use the "^Kxxxx" character sequences as the first argument to a :map command. In the interest of readability and portability, elvis also allows you to use the symbolic name of a key in that context, instead of the raw characters. These are the same key names that are used by (among other things) the xmodmap command. Here are some of the more important names: Begin, End, Home, Print, Menu, Insert, Undo, Redo, Help, Break, Multi_key, Kanji, and Mode_switch.

8.1.6 X11 Icons

Elvis has a 48x32 monochrome icon compiled into it. This icon is stored in the file guix11/elvis.xbm. It is a standard X11 bitmap file.

There are also a variety of colored icons in that directory, in standard X11 pixmap files. These are not compiled into elvis. If you want to use one of these, you'll need to configure your window manager to substitute the colored icon for the compiled-in monochrome icon. Each window manager is configured in a different way, and I can't tell you about every single one out there. But I use FVWM2, and I can tell you how to configure that: In your ~/.fvwm2rc file, add a line which reads...

```
Style "elvis" Icon /usr/include/X11/pixmaps/elvis.xpm
```

Note that this expects the elvis.xpm file to be copied into /usr/include/X11/pixmaps/. When you install elvis by running **make install**, the insticon.sh shell script is run; this checks for a whole series of likely places to copy icons of various sizes, and copies them there.

The following color icons are available:

NAME	DESCRIPTION
elvis.xpm	48x32, 4 colors, same as the monochrome icon
mini.xpm	16x14, 6 colors, for fvwm95 taskbar
normal.xpm	56x46, on a shaded button for NextStep-ish WMs
small.xpm	21x18, on a shaded button for NextStep-ish WMs

The last two use many colors, but most of those colors are for the shaded button background, not the icon itself. Other shaded-button icons use the exact same colors for the shading, so the overall impact on your color table isn't too bad. But if you don't normally use icons on shaded buttons, then you should probably use only the first two icons.

8.2 Windows Interface

The windows interface works under Microsoft's Windows95, Windows98, or WindowsNT (version 3.51 or later) operating systems. It offers a full graphical interface with all the usual bells and whistles. Subsections here discuss the mouse, keys, colors, printing, and fonts.

Because Microsoft doesn't allow a single .EXE file to contain both a Windows interface and a text-based interface, the Windows version resides in a separate file named WinElvis.exe. (The text-based version is named elvis.exe, and it uses the termcap interface.)

8.2.1 Windows Mouse

In addition to all the usual mouse actions in the menubar, toolbar, and scrollbar, you can use the mouse in elvis' main text area as follows.

Dragging the mouse with the left button pressed causes elvis to select characters, like the lowercase v command. Dragging with the right button pressed causes it to select a rectangular area, like the ^V command. Dragging in the left margin (where the mouse cursor changes to up-and-right-arrow) causes whole lines to be selected.

Clicking with either the left or right mouse button will move the cursor to the clicked-on character. When you click with the left button, if a selection is highlighted then elvis will cancel the selection; clicking with the right extends the selection to include the clicked-on character.

Double-clicking on a word with the left button causes elvis to perform a tag search, like the ^] command. Double-clicking with the right button pops back to the previous position via the tag stack, like the ^T command.

8.2.2 Windows Keys

In addition to all the ASCII keys, WinElvis allows you to :map any cursor keys or function keys. In fact, the cursor keys all have rather convenient maps built-in; you can see them by running ":map" with no arguments.

All of the cursor keys and function keys send multi-character sequences to WinElvis. WinElvis then uses its standard mapping facilities to convert those sequences into something that it can recognize and act on. Since the multi-character sequences aren't standardized, and are usually hard to guess or remember, WinElvis allows you to refer to them symbolically. The following symbols are used for referring to the cursor keys:

KEY	SYMBOL	MAPPED TO
up arrow	<Up>	k
down arrow	<Down>	j
left arrow	<Left>	h
right arrow	<Right>	l

Page Up	<PgUp>	^B
Page Down	<PgDn>	^F
Home	<Home>	^
End	<End>	$
Ctrl + left arrow	<CLeft>	B
Ctrl + right arrow	<CRight>	W
Ctrl + Page Up	<CPgUp>	1G
Ctrl + Page Down	<CPgDn>	G
Ctrl + Home	<CHome>	1G
Ctrl + End	<CEnd>	G
Ctrl + Insert	<Insert>	i
Ctrl + Delete	<Delete>	x

The function keys are a different story. Vi has a traditional way to access function keys in a terminal-independent manner, so WinElvis starts with that and extends it just slightly. The benefit of this is that you can use the same function key maps in other versions of elvis, or even in other implementations of vi.

The basic function key symbols are #1 for the **F1** key, #2 for the **F2** key, and so on through #12 for the **F12** key. Combinations involving the Shift, Ctrl, and Alt keys are specified by appending "s", "c", or "a" onto the symbol. For example, **Ctrl-F1** is mapped using the symbol #1c.

8.2.3 Windows colors

WinElvis allows you use the :color command to change the colors used for the different fonts. The color names that it supports are: black, blue, cyan, green, red, magenta, brown, gray, darkgray, lightblue, lightcyan, lightgreen, lightred, lightgray, yellow, and white.

8.2.4 Windows Printing

The default value of the lptype option is "windows". This uses the standard Windows graphical print spooler and should be able to print on any printer that Windows supports. The lpout option is ignored when lptype=windows.

However you still have the option of changing lptype to one of its other values. The other values will generally print faster, and may even look slightly better, but that isn't much of a motivation. A more common reason for changing lptype would be to print into a file in a specified format.

8.2.5 Windows Fonts

WinElvis allows you to specify one base font for each window, via the font option. You can set this to the name of any fixed-pitch font, such as "courier*12".

Conceptually elvis supports six different fonts: normal, bold, italic, fixed, emphasized, and underlined. WinElvis derives these six fonts from the base font, via options named normalstyle, boldstyle, italicstyle, fixedstyle, emphasizedstyle, and underlinedstyle, respectively.

Each of these options may be "n" to use the base font without any changes, or any combination of "b" for bolding, "i" for italicizing, or "u" for underlining. By default, boldstyle and emphasizedstyle use "b", italicstyle uses "i", and underlinedstyle uses "u". You can override these via :set, and make them be different colors via the :color command.

However, when printing WinElvis will always make them all black, and always uses the default derived fonts.

8.3 Termcap Interface

The termcap interface is the one you'll use most often on non-graphic terminals. It looks and acts a heck of a lot like the traditional vi. The biggest addition is the support for multiple windows. (For more information on how to use multiple windows, start elvis and give the command :help ^W.) Subsections here discuss terminal databases, termcap fields, keys, and graphic characters.

If your terminal supports ANSI color escape sequences, then you can use the :color command to assign different colors to the six basic fonts: normal, bold, italic, underlined, emphasized, and fixed. You must assign a normal color first, e.g., ":color normal yellow".

There are three additional options when using the termcap interface: term, ttyrows, and ttycolumns. The term option contains the name of the termcap entry being used; it should correspond to the type of terminal you're using. The ttyrows and ttycolumns options give the size of the screen.

Under Win32, there is also a codepage option for detecting or changing the current code page. Win32's termcap interface also supports the mouse, using basically the same rules as the x11 interface. The only differences are that it doesn't cut & paste via the clipboard, and pressing both buttons of a two-button mouse will simulate pressing the missing middle button.

8.3.1 Termcap, Terminfo, and tinytcap

Termcap is a database of terminal characteristics, and a library of C functions for accessing that database. It was created at Berkeley to allow the original vi editor to be terminal-independent. Elvis' termcap user interface was written to use this.

AT&T created the *terminfo* database and library, adding a few minor features. Most modern UNIX systems use terminfo instead of termcap. Fortunately, terminfo's library contains functions which emulate the termcap functions, so the termcap interface can be compiled to work with the terminfo library.

The tinytcap.c file contains a simple reimplementation of the termcap library, for those systems (such as MS-DOS) which don't have either a real termcap, or terminfo. Tinytcap's database is hard-coded into it; to add or modify a terminal description, you need to edit tinytcap.c and recompile elvis.

8.3.2 Common termcap values

This section describes most of the termcap values used by elvis. The values which deal with cursor keys and graphic characters will be described in the following sections.

Termcap field names are two characters long. Some names supply Boolean values, and others supply numeric or string values. A Boolean value is made true by giving the name; the absence of its name in a terminal's entry indicates a false value for that field, for that terminal. For numeric fields, the name is followed by a '#' character and then decimal digits specifying the value. For string fields, the name is followed by a '=' character and then a string. Fields are delimited by ':' characters.

TERMCAP FIELD	DESCRIPTION
:AL=:	Insert a given number of lines before current line
:al=:	Insert one line before the current line
:am:	Automargin - cursor wraps at end-of-line
:bc=:	Move the cursor back one character
:cI=:	Set cursor shape to "insert" shape
:cQ=:	Set cursor shape to "quit" shape
:cR=:	Set cursor shape to "replace" shape
:cV=:	Set cursor shape to "vi command" shape
:cX=:	Set cursor shape to "ex command" shape
:ce=:	Clear from cursor to end-of-line
:cm=:	Move cursor to a given row/column
:co#:	Width of screen, in columns
:DC=:	Delete a given number of character at the cursor
:dc=:	Delete one character at the cursor position
:DL=:	Delete a given number of lines at the cursor
:dl=:	Delete one line at the cursor position
:IC=:	Insert a given number of characters at the cursor
:ic=:	Insert one character at the cursor position
:ke=:	Disable the cursor keypad
:ks=:	Enable the cursor keypad
:li#:	Height of screen, in lines
:md=:	Start bold text
:me=:	End bold or half-bright text
:mh=:	Start half-bright text (used for italic text)
:pt:	Terminal supports physical tabs
:se=:	End standout text
:sg#:	Width of gap required by the :so=:se=: strings
:so=:	Start standout text
:sr=:	Reverse scroll one line (limited form of :ic=:)
:te=:	String that elvis sends upon exiting
:ti=:	String that elvis sends when starting
:us=:	End underlined text
:ug#:	Width of gap required by the :us:ue:md:me: strings
:up=:	move cursor up one line
:us=:	Start underlined text
:vb=:	Visible alternative to the bell
:ve=:	Set cursor shape to "quit" shape

:vs=:	Set cursor shape to "vi command" shape
:xn:	Brain-damaged newline; ignore the :am: flag

8.3.3 Cursor Keys and Function Keys

Cursor keys and function keys generally send escape sequences when struck. Elvis needs to know what those escape sequences are, so it can recognize the keystroke and act accordingly.

The names of the fields for the arrows are pretty well standardized in termcap, but the other cursor keys are still rather unsettled. Different UNIX variants use different names for the same key. Elvis supports all common names for each key.

Function keys are even more challenging. Originally termcap only had strings which described the first 4 function keys. This was easy to extend to 9 keys, but starting with the 10th function key things get strange because termcap field names must be two characters long. Also, there was no way to describe shift-function keys, control-function keys, or alt-function keys, so I invented by own fields for them.

The following table lists all of the key field names, and the keys they refer to. For keys which may be described via more than one field name, the preferred field name is listed first. It also lists the key's label, as reported by :map and what (if anything) that key is normally mapped to.

KEY LABEL	TERMCAP NAMES	DESCRIPTION
<Up>	:ku=:	Up arrow, mapped to "k"
<Down>	:kd=:	Down arrow, mapped to "j"
<Left>	:kl=:	Left arrow, mapped to "h"
<Right>	:kr=:	Right arrow, mapped to "l"
<PgUp>	:kP=:PU=:K2=:	Previous Page, mapped to "^B"
<PgDn>	:kN=:PD=:K5=:	Next Page, mapped to "^F"
<Home>	:kh=:HM=:K1=:	Home, mapped to "^"
<End>	:kH=:EN=:K4=:	End, mapped to "$"
<Insert>	:kI=:	Insert key, mapped to "i"
<Delete>	:kD=:	Delete key, mapped to "x"
<CLeft>	:#4=:KL=:	Ctrl + Left arrow, mapped to "B"
<CRight>	:%i=:KR=:	Ctrl + Right arrow, mapped to "W"
#1	:k1=:	F1 key
#2	:k2=:	F2 key
#3	:k3=:	F3 key
#4	:k4=:	F4 key
#5	:k5=:	F5 key
#6	:k6=:	F6 key
#7	:k7=:	F7 key
#8	:k8=:	F8 key
#9	:k9=:	F9 key
#10	:k0=:ka=:k;=:	F10 key
#1s	:s1=:	Shift-F1 key
#2s	:s2=:	Shift-F2 key

#3s	:s3=:	Shift-F3 key
#4s	:s4=:	Shift-F4 key
#5s	:s5=:	Shift-F5 key
#6s	:s6=:	Shift-F6 key
#7s	:s7=:	Shift-F7 key
#8s	:s8=:	Shift-F8 key
#9s	:s9=:	Shift-F9 key
#10s	:s0=:	Shift-F10 key
#1c	:c1=:	Control-F1 key
#2c	:c2=:	Control-F2 key
#3c	:c3=:	Control-F3 key
#4c	:c4=:	Control-F4 key
#5c	:c5=:	Control-F5 key
#6c	:c6=:	Control-F6 key
#7c	:c7=:	Control-F7 key
#8c	:c8=:	Control-F8 key
#9c	:c9=:	Control-F9 key
#10c	:c0=:	Control-F10 key
#1a	:a1=:	Alt-F1 key
#2a	:a2=:	Alt-F2 key
#3a	:a3=:	Alt-F3 key
#4a	:a4=:	Alt-F4 key
#5a	:a5=:	Alt-F5 key
#6a	:a6=:	Alt-F6 key
#7a	:a7=:	Alt-F7 key
#8a	:a8=:	Alt-F8 key
#9a	:a9=:	Alt-F9 key
#10a	:a0=:	Alt-F10 key

8.3.4 Graphic characters

Elvis uses graphic characters for HTML mode's <pre graphic> and <hr> tags.

Originally termcap didn't support a way to access the terminal's graphic characters. A standard of sorts was eventually developed under the XENIX variant of UNIX. Later, the terminfo library adopted a different way to access the graphic characters, and this was worked back into the termcap standard, displacing the XENIX standard. The terminfo method is preferred, these days. Elvis supports both.

Terminfo Strings

TERMCAP FIELD	DESCRIPTION
:as=:	Start graphic text
:ae=:	End graphic text
:ac=:	Maps VT100 graphic chars to this terminal's chars

The terminfo method uses the :as=:ae=: strings for turning the graphical character attribute on and off. While in graphic mode, the value of the :ac=: string is interpreted as a list of character pairs; the first character is a VT-100 graphic character, and the following character is this terminal's corresponding graphic character. The following table lists the (text versions of) VT-100 graphic characters, and descriptions of them. It also includes IBM PC characters.

VT-100	IBM PC	DESCRIPTION
'q'	'\304'	horizontal line
'x'	'\263'	vertical line
'm'	'\300'	lower left corner (third quadrant)
'v'	'\301'	horizontal line with up-tick
'j'	'\331'	lower right corner (fourth quadrant)
't'	'\303'	vertical line with right-tick
'n'	'\305'	four-way intersection, like '+' sign
'u'	'\264'	vertical line with left-tick
'l'	'\332'	upper left corner (second quadrant)
'w'	'\302'	horizontal line with down-tick
'k'	'\277'	upper right corner (first quadrant)

So, for example, an entry describing the IBM PC would contain the following:

```
:ac=q\304x\263m\300v\301j\331t\303n\305u\264l\332w\302k\277:
```

XENIX Termcap Strings

TERMCAP FIELD	DESCRIPTION
:GS=:	Start graphic text
:GE=:	End graphic text
:GH=:	Horizontal bar
:GV=:	Vertical bar
:G3=:	Lower-left corner (i.e., third quadrant)
:GU=:	Horizontal bar with up-tick
:G4=:	Lower-right corner (i.e., fourth quadrant)
:GR=:	Vertical bar with right-tick
:GC=:	Center crosspiece (i.e., a big '+' sign)
:GL=:	Vertical bar with a left-tick
:G2=:	Upper-left corner (i.e., second quadrant)
:GD=:	Horizontal bar with a down-tick
:G1=:	Upper-right corner (i.e., first quadrant)

In Xenix, a separate string is used for each line-drawing graphic character. There are also optional :GS=:GE=: strings for starting and ending graphic mode. If the :GS=:GE=: strings aren't specified, then termcap is expected to set the MSB of each character in the graphic character strings.

8.4 VIO Interface for OS/2

The *vio* interface is an OS/2-specific text-mode interface. It should behave almost exactly like the termcap interface in all respects: same options, same colors, same windowing features, etc. Unlike the termcap interface, the vio interface must be run locally. It can't run over a network via telnet, but the termcap interface can.

8.5 Open Interface

The *open* interface was created for use on terminals which lack some necessary capability (such as the :cm=: cursor movement command), or terminals of an unknown type. The open interface is ugly; if you have a choice, you should always use the termcap interface instead.

The open interface works on all text terminals because the only control codes it uses are backspace, carriage return, and line feed.

It only allows you to edit one line at a time. When you move to a new line (e.g., by using the j or k commands), the screen scrolls up and the new line is displayed at the bottom of the screen. This is true even when you're moving the cursor back towards the beginning of the edit buffer; the lines of the buffer will appear on the screen in reverse order! The open interface can be very confusing.

However, practically all of the normal visual commands are available. The only ones missing are those that specifically affect a whole window.

8.6 Quit Interface

The *quit* interface is intended to be used for executing scripts of ex commands. It performs all of the usual initialization, and then quits. It is normally used in conjunction with the **-c** *command* flag.

For example, you can have elvis load a file, print it, and then exit via the following command line...

elvis -G quit -c lp *somefile*

Because the usual initialization guesses a file's display mode automatically, this one command can be used to format and print HTML documents, man pages, C code, and even hex dumps of binary files.

8.7 Script Interface

The *script* interface is similar to the quit interface, except that this interface reads ex commands from stdin and executes them; when it detects end-of-file on stdin, it exits. Status messages are disabled while running a script, but error messages are still output.

Rather than selecting this interface via -Gscript, you will usually select it via the -s flag (or the obsolete - flag), which has the additional side-effect of disabling all initialization scripts. This is desirable since it causes the script to behave exactly the same way for different users, regardless of any customization they have done.

Here's an example shell script that uses this feature to swap instances of the words "left" and "right" in a group of files.

```
        #!/bin/sh
        for file
        do
                elvis -s $file <<EOF
try %s/\<left\>/:TEMP:/g
try %s/\<right\>/left/g
try %s/:TEMP:/right/g
if modified
then write
EOF
        done
```

Note that the five lines between "<<EOF" and "EOF" are a series of elvis commands, and everything else is handled by the /bin/sh shell. The :try commands are used to silence error messages from any :s commands which fail to find any matching text. The :if/:then commands are used to test the buffer's modified option, so a file which contains no instances of "left" or "right" won't be rewritten needlessly.