**NAME**

guide — a hypertext system
guide_reader — a version of *Guide* that can be freely distributed and copied, but lacking authorship facilities

**SYNOPSIS**

**guide** [ [ **–** ] [ **–**<constructor> <argument> ] [ **–**<searchletter><string> ] [ **–dir** ] [ **–e** ]
[ **–E** ] [ **–G[**<digit>] ] [ **–i** ] [ **–m[**<section-name>] ] [ **–M**<menu-string> ] [ –mono]
[ **–N** <menu-file> ] [ **–p** ] [ **–P**<button-settings> ] [ **–PP** ] [ **–q** ] [ **–r** ] [ **–R** ]
[ **–s** ] [ **–S**<letter>**=**<integer> ] [ **–t** ] [ **–T** ] [ **–u**<digit> ] [ **–v** ] [ **–V** ] [ **–w** ]
[ **–Wc** ] [ **–x** ] [ file ] ] ...

where <searchletter> is **f** or **F** or **b** or **B**, and <constructor> is **do** or **echo** or **load-def** or **run**. (The name can be *guide_reader* rather than *guide*, and everything said below about *guide* also applies to *guide_reader*, except for authorship facilities.) Files can reside on remote hosts; if so they should take the form *Hostname:filename*. It is possible to pipe into *Guide*; in this case the standard input (i.e. the material in the pipe) is, by default, loaded after the initial options have been read, and before the first file is loaded, e.g. before *afile* in the command:

*program* | *guide –V afile*

Alternatively, following the normal UNIX convention, a '–' as an argument means insert the standard input here.

**SAMPLE SOURCE FILES**

*Guide* has a library of sample source files (normally in the directory */usr/local/lib/guide*). One of these is called *help*, and explains the rest of the library. To load this, simply type the command:

*guide help*

It is probably best to try this, in order to get an impression of *Guide*, before proceeding further with this explanation.

**DESCRIPTION**

*Guide* is a tool to allow readers to peruse a file, called the '*source*', at the level of detail they want, and thus to create a document tailored to their individual needs. The initial source to be perused is in the *file*(s) specified on the command to enter *Guide*. *Guide*'s main design features are as follows:

*Simplicity of use:* documents are presented as scrollable texts, not as trees or other data structures that users might get lost in. Documents can be expanded (e.g. to provide a greater level of detail) by selecting *replace-buttons* that lie within the source. Replace-buttons are displayed in a bold font — the same font that is used for buttons in the *Guide* menu. Scrolling is achieved either by using commands in the menu or by using a *thumb-bar*, which is displayed on the right of the *Guide* window.

*Editing:* the user can edit documents just by placing a *marker-bar* in the required place and then typing, in the same way as with a screen editor.

*Filling and indentation: Guide* automatically fills lines on the display in order to fit the size of window used. *Guide* input should only contain newlines where they are really wanted (e.g. at the end of a paragraph). A tab causes each displayed line to be indented until a newline character is encountered in the input; thus a tab can be used to indent a whole paragraph — provided the paragraph has been typed in the recommended way, with no embedded newlines.

*Glossaries:* jargon words in the text can be made into *glossary-button*s. These are underlined, and if the user selects one of them its definition is inserted into the glossary and displayed. The glossary is kept in alphabetical order.

*Easy authorship:* authors prepare documents in the same way that readers use them. Thus the author is the reader and the reader is the author. When an author wishes to add structure to a document (e.g. new replace-buttons with appropriate text to replace them), he selects *author-display* by using the **Author** command in the *Guide* menu; this gives an extended menu (e.g. the menu command **+Glossary**, which creates a new glossary-button) and also shows the underlying structure from which the document was created. The structure is shown by inverse-video *control characters* embedded in the text. These show the boundaries between the various parts of the document. To create a source from scratch, *Guide* should be called with a null initial source file, i.e. the *file* should be omitted from the command to enter *Guide*.

*Graphics:* the text can include rectangular bitmap pictures, in monochrome or colour.

*Multi-way replacements:* buttons can be presented as alternatives by using the '*Enquiry*' mechanism, e.g. 'Is it **Blue** or **Green**?'.

*Views:* the *Guide* window can be divided into as many as twenty separate views (sub-windows), each working on a separate source file. Normally one view is reserved for the glossary.

*Saving:* it is possible to save the complete text as viewed (the *document*) or the underlying source from which it was created. A reader who, say, wanted a print-out of his tailored document might do the former, whereas the author of a new source file would do the latter.

*Undoing:* the expansion caused by selecting a replace-button can be 'undone' by pointing the mouse and pressing the middle button (button B).

*Levels of reading:* each replace-button has an associated *asking-level*, and the user can set his *user-level*. All replace-buttons whose asking-level is below the user-level are automatically expanded. Alternatively any replace-button can have a *button-condition*, which specifies a condition under which the button should be automatically selected.

## BUTTONS AND POINTING

Selection is done by pressing the left-hand button (called button A in the Guide user manual) on the mouse. The only uses of the other mouse buttons is that the middle button (button B) is used to undo an expansion and, if the −A option is set, the right-hand button (button C) can be used to *pre-set* (i.e. to select permanently) a replace-button. If a replace-button is pre-set, then every time a replace-button with that name occurs it is automatically selected. For example, if the replace-button **More** was pre-set, then every other **More** replace-button would automatically be expanded. The names of pre-set replace-buttons are shown at the very start of the text when *Guide* is in author-display; these names can be edited just like any other text.

## WINDOWS

*Guide* must be run under a window manager. Most window managers create a new window for *Guide* when it starts. The size of the window can be changed while *Guide* is running, and *Guide* will reformat the display automatically when this happens.

## OPTIONS

The options have the following meanings:

−*<constructor> <argument>*

allows the use of some of the constructors available on extensions to *Guide* buttons. The most useful is −*do*, which allows a series of actions to be performed during the loading process. An example of its use is:

*guide afile −do 'Find&Context-library(Label) "xyz"; Advance'*

Here two *do* program-commands (represented by the last argument to *guide*, which is enclosed within single-quotes) are performed at the end of loading. The result of this is that, after *afile* has been loaded, a search is made for the string *xyz* within a *Label* context, and then the marker-bar is advanced to point after this.

(Constructor options such as −*do* are not available on New program-commands attached to *Guide* buttons; this is because the ordinary facilities for constructors attached to button can be used here.)

**−b***S* is the same as the −*f* option, but searching is confined to names of replace-buttons. The first replace-button whose button-name contains a match is expanded. (If it is desired to suppress *Guide*'s animation of the search procedure, *−B* can be used in place of *−b*.)

**−dir** means that whenever a subsequent file to be loaded is a directory that is to be set as the 'remembered directory' (see below), then the normal warning should be suppressed. (Some *Guide* applications automatically set the remembered directory in order to ensure that all loading comes from this directory rather than from the current directory; such applications use the *−dir* option. This option, unless set locally by extensions to a dependent button, will, once set, apply to the rest of the *Guide* session.)

**−e** (for 'envelope') is useful if the user is suspicious of a *Guide* document. The option checks with the user before any shell-script is run, or any writing of files is initiated from within the document (e.g. by a Save program-command). Once this option has been set, it cannot subsequently be unset within a *Guide* session.

**−E** is like the *-e* option except that running shell-scripts, etc., is forbidden. This option is useful for public *Guide* logins: it prevents hackers doing funny things from within *Guide*.

**−f***S* uses the *Guide* **Find** command to find the string *S* after loading is complete. The search starts with the file that follows this option, and continues to the end of the source (and thus covers all subsequent views). This facility is valuable for focussing the screen when *Guide* is used as a help system. (If it is desired to suppress *Guide*'s animation of the search procedure, *−F* can be used in place of *−f*.) For a more general and flexible finding facility, the *−do* constructor can be used instead.

**−G***D* set the amount of space to be used by the glossary-view to *D* lines. Normally the glossary-view has the same size of area on the screen as other views. However if the above option is set, the allocation, if greater than *D* lines, is reduced to *D* lines. (If this option is used twice, without a restart in between, the second setting adds to the first, i.e. *−G4 −G5* is equivalent to *−G9*.)

**−i** identify the version of *Guide*, its copyright, etc.

**−m** the next argument that is a file is taken to be the name of a manual page to be converted to *Guide* format (see 'preprocessor'). Thus

*guide −m ls*

displays the manual page for *ls*. Note that the conversion mechanism assumes that manual pages are in ASCII form (i.e. are not compressed), and lie in subdirectories called *man1*, *man2*, etc. (This assumes that section-names are 1, 2, etc. If names of manual sections can be more than one character, e.g. as in *man1s*, then *Guide* relies on the *whatis* Unix command to find the desired section-name, and hence the desired subdirectory.)

**−m***S* where S is a string representing a section-name, is like *−m*, but the search is confined to the section of the manual pages given by S (e.g. *guide −m2 chmod*).

**−M**<*menu-string*>

replaces the standard *Guide* main menu. Normally this is used to provide a tailored version of *Guide*, which (a) only provides a sub-set of the facilities normally in the main menu, and/or (b) uses different names for the buttons in the main menu, and/or (c) augments the main menu with buttons that perform other actions. The *menu-string* is a mixture of text and button-names — the button-names are embedded within the characters '^'. As an example, the setting of *menu-string* corresponding to the standard *Guide* main menu is:

*−M"^Quit^ ^New^ ^Down^/^Up^ ^Save^ ^Block-edit^ ^Find^/^Next^ ^Author^"*

An example of a setting of *menu-string* for *Guide* to be used as a help-system is:

*−M"− − − HELP SYSTEM − − −*

*^Quit-the-help-system^ ^^ To move around use: ^Down^ or ^Up^"*

This provides the *Guide* **Quit** menu-button (under the name **Quit-the-help-system**) and its **Down/Up** menu-buttons, but no others. The menu is embellished with some explanatory text. (Note that the second menu-button, corresponding to **New**, is null, and that everything after **Up** is omitted.) If, as in the above example, there is no **Save** command, then *Guide* does not allow users to do any editing. (However, if it is desired to allow editing without providing a **Save**, then the **Save** command can be made null — i.e. ^^ — in the *menu-string*.)

As well as being used to change *Guide*'s built-in menu, the *menu-string* can be used to augment it. Any button in the *menu-string* that has an extension is treated as a normal *Guide* action-button, and can be used to initiate shell-scripts, *Guide* actions, etc.; such a button is treated as an extra, rather than a new name for an existing menu-button. For example the *menu-string*:

*−M"^Quit^ ^Restart do New&Replace-all myfile^ ^^^Down^/^Up^ ^Launch run myprog&^ ^Save^"*

adds two commands to the menu, i.e. **Restart** and **Launch**. The **Restart** command loads a fixed starting file; the **Launch** command launches an application. The existing **Quit**, **Down**, **Up** and **Save** commands are also used. Note there is a null name corresponding to the **New** command, thus ruling this command out. (The null name is necessary to keep the *menu-string* in phase with the real built-in menu; if it were omitted the **Down** button in the *menu-string* would have acted as a **New** command.)

New commands can be added anywhere in the *Guide* menu. However if new commands are added beyond the **Author** command, these will disappear when *Guide* goes into author-display.

If it is desired to add new commands, only visible to authors, which create new constructions, this can be done by adding to the context-table a context that performs the required actions.

It is possible to have a null menu, but this must be done using the −N option (see below) rather than the −M option.

**−mono** use monochrome — this may be useful on a colour machine, when faster response time is preferred to colour.

**−N** *<menu-file>*

is similar to the *−M* option, but loads a file rather than a string as the menu. The *menu-file* is an ordinary *Guide* source file. Within the *menu-file*, all action-buttons that do not have extensions are treated as names for corresponding commands in the built-in *Guide* menu. For example if the first such action-button in the *menu-file* was **Exit**, this would be taken as a name for *Guide*'s first built-in command (**Quit**). The *menu-file* can contain any *Guide* constructions except usage-buttons. It is often used to load pictures (icons) to serve as button-names.

There are length restrictions on *menu-files*, so it is not practical (or sensible) to make them too fancy. (Fancy menus also lead to logical problems, and hence confused users. Thus *Guide* imposes some restrictions. For example, although a *menu-string* can include local-buttons, which result in a changing menu, built-in menu actions cannot lie within the replacement of such buttons.) *Menu-files* should therefore be short and simple. The **Author** command must not be within another *Guide* construction.

**−p** the next argument that is a file is taken as the filename of a manual page to be converted. (This is useful for testing draft manual pages that have not yet been incorporated into a *man* directory.)

**−P***<button-settings>*

This option allows the tailoring of the use of buttons on the pointing-device — here assumed to be a mouse. The *button-settings* is a sequence of pairs of letters: the first letter represents a *Guide* action:

- b selecting a button on the screen
- c placing the cursor
- p pre-setting a replace-button

t using the thumb-bar

u undoing

The second letter represents the mouse-button that is to be used to effect the action:

A the left-hand mouse-button

B the middle mouse-button

C the right-hand mouse-button

X no mouse-button (i.e. the action is inhibited)

As an example, the default *Guide* settings for mouse-buttons are

−PbAcApXtAuB

Note that pre-setting is by default inhibited (by *pX*). This is because beginners are often confused by pre-setting. To allow pre-setting to be effected by the right-hand mouse button use *−PpC*.

If use of the thumb-bar is inhibited (by *tX*) the thumb-bar is not shown; this can be useful for material shown in a ‘frame-based’ way.

*Guide* only allows limited tailoring of mouse-buttons, and some combinations, e.g. using the same mouse-button for everything, are likely to lead to unsatisfactory results. It is however possible to have two alternative mouse-buttons for the same operation, e.g. *−PbAbC* would allow either the left or right mouse-button to be used to select a button on the screen.

**−PP** This option is a special case of the − option that applies to buttons on the screen: normally when a replace-button is selected its replacement is briefly highlighted, thus causing a flashing effect on the screen; this option suppresses such highlighting.

**−q** (for quiet) suppresses the warning message that the user gets when he tries to edit some transient material.

**−r** means that buttons should remain when selected, i.e. both the button-name and its replacement will be seen. (In this case the button-name is enclosed within backward-arrow characters, and selecting it will cause an undo).

**−R** cancels the −r option.

**−s** if a transcript is replayed, replay it slowly, repainting the screen after each action (normally the screen is only repainted when the transcript has finished).

**−S**<*letter*>**=**<*integer*>

Guide supports 28 *button-condition-variables*, called U, V and a, b, c, ... , z, which can be used to select replace-buttons automatically. They have integer values and are all initially 0. This option sets such variables, e.g. −Sx=−36 sets x to −36. The variable U (for Universal) relates to the facility whereby every replace-button has the implied button-condition ‘?U > 0’; hence −SU=1 expands all replace-buttons. V is like U, but only applies to buttons that do not have an explicit button-condition — see the Guide User Manual for details.

**−t** write a transcript to the file ‘transcript’ (or to the file named by GUIDETRANSCRIPT if it is set). Writing ends at the end of the *Guide* session, or earlier if the **New** command is used. A transcript is an ordinary *Guide* source file and can be loaded in the normal way. Note that transcripts in *Guide* are an EXPERIMENTAL feature.

**−T** is like ‘−t’, except that the user is not given the option of stopping after a **New** command, i.e. the recording of the transcript can be hidden from the user.

**−u***D* set the user-level to the value of the digit *D*. This means that, on each subsequent source file that is loaded, any replace-button with an asking-level less than *D* is automatically expanded.

**−v** is an invaluable option for allowing authors to check *Guide* sources before releasing them. It verifies, during loading of material, that definitions exist for each usage-button or glossary-button which depends on a definition; it also checks the *Guide* syntax of all button-conditions and extensions. Matters not within *Guide*, e.g. the existence of files referenced by a 'load' extension, or the correctness of a shell-script on a 'run' extension, are not checked at load-time. (However this checking can be done for usage-buttons by loading with the −*SU=1* (or −*u4*) option, which causes all replace-buttons to be replaced, and then viewing the whole source. As each part of the source is painted on the screen, any relevant error messages will appear. An automatic way of scanning the whole source, which saves a lot of time when checking files that would take many screenfuls to paint, is to Save it in a gash file using the 'Just save the text' option.) For a world-definition-button, the −*v* option checks that no world-definition of the same effective name has already been loaded, either from within the current file or a previously loaded file. (Duplicated regional-definitions, even within the same scope, are not flagged as errors.)

Any form of pre-run checking faces a fundamental dilemma: its very purpose is to perform comprehensive checking in advance, but, because it is done in advance, the environment may be different from the actual one. Guide's comprehensive checking is performed at load-time, and the environment may be different from that at paint-time (when a button is first painted on the screen, i.e. when its button-condition is evaluated) or select-time (when a button is selected). The dilemma applies particularly to button-conditions, as illustrated by the following examples:

*? @[Y] > 3*
*? @d[X] && @[Z] == "s"*

The first example tests if the definition of Y has a value greater than 3; it is therefore an error if Y does not exist or has a string value. The second example uses Z only if X is defined; this could be an error, though it may well be that X and Z are in the same module and the test is safe. To resolve the dilemma *Guide* makes the following two assumptions: (a) that the values of button-condition-variables and environment variables do *not* change between load-time and paint-time/select-time; (b) the values of other nested-elements (such as *@[Y]* above) *may* change, and during load-time *Guide* assumes that their values are correct (though it does check for existence of a definition). Thus the first example only gives a load-time error if there is no definition for Y; (however *@{Y} > 3* would be completely checked since, following rule (a) above, its value is assumed to be known at load-time). The second example never gives a load-time error because it is assumed that *@d[X]* will be false if Z does not exist. In all cases complete checks are, of course, performed when and if the button is painted on the screen or selected. No load-time checking is performed when the values of nested-elements are used within button extensions (e.g. *load-def @[dynamic-name]*), since these values are not necessarily known at load-time.

Clearly the above assumptions are imperfect, but they have been found in practice to give a reasonable balance.

**−V** is like −*v*, but has the additional check that a warning is given for every local-button or definition-button that has a null replacement.

**−w** create a new view, and use this for any subsequent files to be loaded.

**−Wc** give a warning if there is a context clash, i.e. two different context-specs are loaded for the same context and the same view.

**−x** place the marker-bar at this point — it is equivalent to a 'find' of the null-string (see −*f*).

Options may also be specified within the list of filenames supplied when using the **New** command. In general, such option settings supplement rather than override the previous settings (i.e. if an option has been previously set it remains in effect).

When running under *suntools*, *Guide* accepts the standard *suntools* options (such as *Wp* and *Ws*). The option −*blanket* can be used to cause *Guide* to run on top of the existing window, instead of in a new window.

When running under *X11*, *Guide* accepts most of the standard options, such as −*geometry* and −*name*.

**SIGNALS**

If *Guide* receives an USR1 signal, it treats this as a request to execute a sequence of program-commands. In this case the file */tmp/guidefind*N, where N is the PID of *Guide*, should contain the sequence. This mechanism is used by *callguide* (q.v.).

**PREPROCESSOR**

*Guide* can have a *preprocessor* incorporated into it. *Guide* is a presentation tool, and the role of a preprocessor is to extract the material to be presented; this may, for example, come from a database.

The standard preprocessor issued within *Guide* is an interface to two further sets of information: (1) some public *Guide* source files; (2) the UNIX manual pages. This preprocessor looks at each filename that the user requests as a *Guide* source for loading. We will first cover case (1) above, i.e. when the filename is *not* to be treated as a manual page. We will use the term *starting-directory* to mean the current directory from which the *Guide* session was started. The starting-directory is made available in the environment variable GUIDECWD — see below.

**PREPROCESSOR: FINDING GUIDE FILES**

If the filename is a directory, this becomes the *remembered directory* for the current view. The remembered directory, though only occasionally used, provides a way for the user to request that a sequence of subsequent filenames is to be loaded from this directory. (When a new view is created its remembered directory is set to the most recent remembered directory, or to the starting-directory if no remembered directory has been set.)

If the filename is not a directory, then the preprocessor searches a sequence of directories to try to find the filename. Usually the exact mechanism of the search should not concern the average user: his view is that he specifies a filename, *bicycle* say, and the preprocessor magically loads a source file that contains information about bicycles. However those who create source files need to know how to exploit the searching mechanism, so we explain it here.

The search proceeds as follows:

(A) if the filename contains a ‘/’, no preprocessing takes place.

Otherwise a search for the filename is made in the following sequence of directories:

(B1) the remembered directory for the current view, or, for an embedded file reference (see below) the embedded-relative-directory.

(B2) the starting-directory.

(B3) the */usr/local/lib/guide* directory (or the directory named by the GUIDELIB environment variable (q.v.)). This directory, or its sub-directories, should be used for all public *Guide* source files.

(B4) the sub-directories of */usr/local/lib/guide*. These are searched in alphabetical order. Such sub-directories are useful for sets of related source files: thus all the source files concerned with office procedures might be put in */usr/local/lib/guide/office*. Two sub-directories are omitted from the search: these are the sub-directory *etc*, which can be used for *Guide* files that are not sources, e.g. fonts, conversion programs, etc., and the sub-directory *man*, which contains caches of manual pages. Moreover sub-directories of sub-directories are not covered by this search, so these can also contain arbitrary files.

If at any stage the search finds the required filename (and it has read permission) this is taken as the source file to be loaded.

**PREPROCESSOR: CONVERTING MANUAL PAGES**

If the −*m* or −*p* option precedes a filename, it is treated as a reference to a UNIX manual page; for example if the filename is *ls*, then the manual page for *ls* will be taken. For the −*m* option, manual pages are sought within each of the directories in the colon-separated list of directory names given by the MANPATH environment variable, or, if MANPATH does not exist, in */usr/man*. The preprocessor searches subdirectories of each main directory (just as the Unix *man* command does) to find its quarry; the manual page, when found, is automatically converted to *Guide* format before it is loaded.

The exact details of the above conversion process are as follows. To save repeated conversions, the preprocessor keeps the converted files in a cache under the directory */usr/local/lib/guide/man*. Within this directory, the sub-directory *cache1* contains the converted manual pages for Chapter 1 of the *UNIX Programmer's Manual* (and thus contains cached material derived from */usr/man/man1*), the sub-directory *cache2* contains converted manual pages for Chapter 2, etc. (On some System V implementations, where section-names can be more than one character, cache directories can have names such as *cache1s*.) If a cache directory does not exist, or is not writeable, no caching is done.

If a subsequent attempt is made to load the same manual page, the cache is used to speed loading by avoiding a re-conversion of the manual page. The cached file is not, however, used if the manual page has been changed since the cached file was created.

The option −*mS*, where *S* is a section-name, can be used to constrain the preprocessor to look within a particular Section of the manual (and its corresponding cache).

## PREPROCESSOR: FILENAMES EMBEDDED WITHIN GUIDE SOURCE FILES

A large *Guide* hyperbase will often be spread over several directories and subdirectories, and/or may involve files belonging to several different users. When one *Guide* file refers to another, we call this an *embedded file reference*.

Embedded file references always arise with an extension to a dependent button. In particular an embedded file reference may be used when a button:

(1) loads a definition from a definitions-file. (The definitions-file may be implicit (*glossary-guide*) or it may be specified using the *in* facility.)

(2) loads a whole file.

(3) runs a shell-script that reads or writes files. The shell-script itself may be stored in a file.

(4) runs a program-command-sequence that loads files. (Incidentally the filename specified on a *Save* program-command is not treated as an embedded file reference, as the filename always relates to the starting directory.)

The following example shows an instance of each of these

**Example 1** in /x/f1 load f2 run y/f3 do New&New-view f4

Embedded file reference may use *absolute* filenames, which in UNIX begin with the character '/', or *relative* filenames. In our example the first filename ('/x/f1') is absolute, and the other three filenames are relative. At any one time *Guide* may be displaying a lot of diverse material, with material loaded from different files interspersed. The question therefore arises: if R is a relative filename, then what directory is R relative to? Possible answers are:

(1) the starting-directory.

(2) the directory from which the material containing R was loaded, i.e. if the reference to R was loaded from the file /D/E/F, then the directory would be /D/E.

Clearly the answer for file names typed by the user (e.g. in response to the **New** or **Save** menu-commands) is (1): the user should be unaware of the various machinations with files that occur when he selects a button, and it would be upsetting if the default directory for loading kept changing as internal files were loaded. (The normal reader is unaware of embedded file references: they occur as extensions to buttons and these are only visible in author-display.)

For embedded file references, on the other hand, the natural answer is (2). In our example, when a usage-button, say, loads file /D/E/F, and within that file another usage-button loads file R, then what is meant is /D/E/R. (In particular this has the advantage that is the directory E is moved to E1, the reference will still work.)

*Guide* adopts these natural approaches. Indeed they are usually so natural that the effect is exactly what authors would expect, without considering the machinations involved. There are, however, some more complex cases — arguably ones that good authors will avoid — where the internal machinations of case (2) need to be understood.

However for authors who are not interested in details at this time there is a simple rule that: *if a Guide source file F contains a relative filename R, then R is taken relative to the directory in which F lies.*

**PREPROCESSOR: DETAILS ABOUT THE EMBEDDED-RELATIVE-DIRECTORY**

*Guide* has the concept of a default directory to be used for relative embedded file references. This is called the *embedded-relative-directory*. Each view has its own embedded-relative-directory. In essence the rule for setting the embedded-relative-directory is that by default it is the directory from which the most recently loaded file in that view came, but within the replacement of a dependent button this default can be overridden.

The details are as follows.

*1: the view*

For each view the embedded-relative-directory is set according to the most recent file loaded in that view by means of arguments to Guide or filenames typed by the user for the **New** command. (If no files have yet been loaded, the embedded-related-directory is the starting-directory.) Thus if the user types the command

guide /X/Y/Z

the embedded-relative-directory for the first view will be /X/Y, and relative embedded file references will relate to this. If the user types the following command, which has two files as its arguments:

guide N /X/Y/Z

then *Guide* loads the two files end to end, as if they were a single entity: it does *not* record the boundary between them. Thus a single embedded-relative-directory applies to both of them, and the 'most recent file loaded' rule says, somewhat arbitrarily, that it will be /X/Y, and relative embedded file references within file N will relate to this — an undesirable effect, but then it is not generally good practice to load two separate files end-to-end in a view (unless you are an author creating a new hyperdocument out of existing material).

*2: local settings for dependent buttons*

The above setting for a view can be overridden within the replacement of individual dependent-buttons. Assume for example that a usage-button within the file /X/Y/Z is

**Load-it** load /J/K

or equally well

**Load-it** in /J/K

(meaning that the definition of **Load-it** is to be sought in the definitions-file /J/K). In both cases, within the replacement of **Load-it**, the embedded-relative-directory is /J, rather than /X/Y, which is the default for the rest of the view. Likewise if the usage-button involves nested usage-buttons, /J may itself be overridden within the scope of the nested replacements. The same rule applies to the replacement of glossary-buttons. A glossary-view normally consists of several different replacements of glossary-buttons, and these may each have a different embedded-relative-directory. (The embedded-relative--directory for the glossary-view is the starting-directory, but this is not normally used, as it is overridden by the replacements with the view.)

Occasionally the replacement of a dependent button can be constructed from material from several different files, e.g. as in our **Example 1** button or in:

**Example 2** load A load-this-def load B

In such cases *Guide* uses its rule that it sets the embedded-relative-directory from the most recent file loaded (i.e. the last *load*, *load-def*, *load-this-def*, or implicit *load-this-def*). If no loading is done (e.g. the replacement is generated by a combination of *do*, *run* and *echo* constructors), then the embedded-relative-directory will be the same as the containing environment. (To be exact it is the containing environment when the constructors are evaluated, though this will not change unless the button is moved around.)

All the above rules are applied using the current position, on the *Guide* screen, of the button to which an embedded file reference is attached. Thus if a usage-button contains an embedded file reference to X (e.g. load X), and the usage-button is moved from one view to another using **Block-edit**, then the meaning of X will change if the new view has a different embedded-relative-directory from the old. This new meaning will come into effect when (and if) X is next loaded.

## PREPROCESSOR: EVEN MORE DETAILS ABOUT EMBEDDED FILENAMES

There are a few cases where the above rules need further explanation. They are:

- when a loaded file is a raw picture, it cannot, by the nature of the way *Guide* treats pictures, contain embedded file references. Therefore it does not count, in the above rules, as the 'most recently loaded file'.
- the **Save** menu contains a command to **Dump** the current document. The result can be that material from diverse sources is dumped into a single source file. This can cause relative filenames to be interpreted in a new way when the dumped file is reloaded.
- Assume that a dependent button B uses a definition that has already been loaded. When B is selected, *Guide* makes a copy of its definition. Within this copy, embedded file references are relative to the directory from which the definition was originally loaded — a natural rule, though this may require some thought.

## PREPROCESSOR: FINDING GLOSSARY FILES

The preprocessor is also used to pick the correct definitions-file. *Guide* definitions-files are called *glossary.guide* (unless, of course this default has been overridden by the author by an *in* extension), and typically such a file will exist within most directories used for *Guide* source files. The preprocessor looks for the definitions-file in the embedded-relative-directory for the current view. Thus if the embedded-relative-directory was */usr/local/lib/guide/office* the definitions-file will be taken as */usr/local/lib/guide/office/glossary.guide*. The preprocessor does not conduct a search for a definitions-file; if the required one does not exist it gives an error.

## NON-INTERACTIVE USE

Occasionally it may be useful to run *Guide* in a batch mode. As one example, *Guide* might be used within a makefile to prepare a document for later processing: it might load some material, perform some actions and then save the result as the required document. As a second example, there may be calls of *Guide* in a non-interactive test suite which validates *Guide* documents. Such examples are likely to use the facility to specify, within the *Guide* command line, some actions to be done, e.g. "*−do Save&With adoc*".

To make *Guide* non-interactive, the GUIDEOUTPUT environment variable should be set. It must be set before *Guide* is started, because any changes in its value during a run are ignored. GUIDEOUTPUT can either be set to a filename or to null (meaning standard output). When *Guide* runs non-interactively, no window is created, and all text that would normally be painted on the screen is instead written to GUIDEOUTPUT. (Two details: pictures, etc., are ignored; if the GUIDEOUTPUT file already exists it is overwritten.) There are no size limits on the material written, unlike on a screen, where typically only a few dozen lines can be painted.

When in non-interactive mode, if *Guide* reaches the stage it would interact with the user, it does the following:

- if there is a built-in menu-button called "OK" (upper and/or lower case), it is selected. This covers most *Guide* error messages.
- otherwise the first built-in menu button is selected; usually this is "Quit", but it could be, e.g.,

"Cancel-this".

If the above does not cause a quit, *Guide* continues by repainting again in the normal way, but writing the output on the end of GUIDEOUTPUT rather than on the screen.

**REMOTE FILES**

Files from remote hosts are, by default, accessed by anonymous ftp. The preprocessor rules for embedded filenames also apply to filenames embedded in remote files, and this has a highly beneficial effect: *even if a hyperdocument on a host consists of a lot of inter-linked files, rather than just one file, the hyperdocument will still generally be usable remotely without any change*. All that the owner of the hyperdocument needs to do to make it internationally usable is to make it accessible by anonymous ftp. Further details are discussed below. If a remote file is a directory, *Guide* treats it rather differently to a local directory: since the user may well be exploring the filing system of the remote host, a directory is presented as a set of buttons, one corresponding to each file within the directory; expanding a button reveals the corresponding file.

If the file name on the remote host ends with the suffix **.Z**, *Guide* automatically uncompresses it before loading it.

Whole *Guide* applications are often run using files on remote hosts; such applications are often launched by a shell-script, which may set environment variables, specify a special menu, etc., before starting the application. *Guide* therefore has a facility to run a shell-script that is stored on a remote host (within the anonymous ftp area); the script is automatically copied to the local host and run there rather than remotely. This facility is invoked by putting the prefix ‘run:’ on the front of the name of the remote file, e.g.:

*run:unix.hensa.ac.uk:/adir/ascript*

Remotely run shell-scripts behave as follows:

(1) They are run as if they were in the starting-directory on the local host.

(2) The environment variable GUIDESCRIPTHOST (q.v.) is set, before the run starts, to specify the remote host, and the directory on the remote host where the shell-script lies (e.g. *adir* in the above example). This means that if the script launches *Guide*, as it usually will, *Guide* will take its input files from the remote host; relative pathnames supplied as arguments should be relative to the directory where the shell-script lies.

(3) Apart from cases covered by (2), the shell-script should only refer to files on the local host.

(4) If the shell-script produces any output this is treated like a normal ‘run’ constructor in *Guide*, i.e. the effect as if a file containing this output were loaded.

The ‘run:’ prefix can alternatively be written as ‘brun:’ or ‘qrun:’. The ‘brun:’ prefix runs the shell-script in the background. The ‘qrun:’ prefix does likewise but also quits from the current *Guide* process. For example if ahost:ascript is a shell-script that launches a *Guide* application, then

*guide qrun:ahost:ascript*

is a convenient way of starting it. Indeed, if you have an application that you want others to try, you just move it into an area that is accessible by anonymous ftp, and suggest to remote users that they execute a command such as that above.

**MORE DETAILS ON REMOTE FILES**

The following are some more specific points about the use of remote files.

- If the remote file contains a shell-script, this is run on the local host, and thus any references to files on the remote host will not work — unless they are supplied as arguments to a call of *Guide* itself (see GUIDESCRIPTHOST below). Instead filenames will be taken relative to the starting-directory on the local host. Shell-scripts from remote hosts present a virus danger: if you are worried about this, use the −e option.
- Since this facility is for UNIX *Guide*, it is mainly designed for UNIX hosts and may not work on others.

- The remote facility is only for the *loading* of files; files cannot be saved on remote hosts.
- The *Guide* library will be taken from the local host, not the remote one. (However if the hyperdocument on the remote host has its own special version of *glossary.guide* or *contexts.guide*, then these will be used in the normal way, thus following the principle that such hyperdocuments can be used without change.)
- The mechanism for getting a remote file is that *Guide* calls a shell-script called *guide_get*. It searches for this first in the starting directory and then in the *etc* sub-directory of the *Guide* library, where the default version lies. Users can therefore provide their own version of *guide_get*, e.g. to provide non-anonymous ftp.

**USING GUIDELOAD TO CONVERT FILENAMES BEFORE LOADING**

The preprocessor allows authors some flexibility in the way filenames for loading are to be treated. Indeed they need not be treated as filenames at all: they could, for example, be search keys for a database. The mechanism is that the author supplies a shell-script which takes an original "filename", and returns the name of the file that is actually to be loaded. The original filename is supplied as the environment variable GUIDEFILE. Thus the shell-script

*echo $GUIDEFILE*

would have no effect at all, since it just echoes the original filename. The shell-script could however use the original filename to access a database, a remote host, an information service, ..., and place the output from this in a new file. The name of the new file is then returned by the shell-script. The result of the shell-script must always be a single filename; the filename can, however, be followed by a suffix "+delete", in which case *Guide* will delete the file after it has been read. The deletion feature is obviously useful if the result of the shell-script is a temporary file. The shell-script is supplied by the author as the value of the environment variable GUIDELOAD.

As an example, GUIDELOAD might be set (assuming the Bourne shell) as:

*GUIDELOAD='my_translation $GUIDEFILE > /tmp/t#$$; echo /tmp/t#$$ +delete'*
*export GUIDELOAD*

In this case *my_translation* processes the original file, and sends the output to a file that is then read as if it was *Guide*'s input file. *Guide* deletes this file when it has read it, because of the "+delete" suffix. Note that the GUIDELOAD shell-script is called for all files that *Guide* tries to load; it even applies when *Guide* tests if local versions of the library files *glossary.guide* and *contexts.guide* exist, though in these cases, *GUIDE* ignores any error messages from the shell-script — it just assumes the file does not exist locally, and uses the library version. Note also that when GUIDELOAD is set the setting is usually enclosed within single quotes: this is because the value of GUIDEFILE is not to be inserted until *Guide* runs the shell-script.

Before calling the shell-script, *Guide* adds the *etc* sub-directory of its library to the search path. Thus the shell-script can easily use the script *guide_get* (see above), which resides in that directory.

To avoid tedious errors, *Guide* strips newlines and white-space from the end of the filename returned by GUIDELOAD.

**PICTURES**

Pictures can be Sun rasters (in standard format, but *not* run-length-encoded format) or, for monochrome pictures only, X bitmaps. Because of problems of portability, *Guide* does not save colourmaps associated with pictures. Therefore colour (or grey-scale) pictures should not be directly included in *Guide* source files; instead they should be loaded dynamically by a 'load' construction attached to a usage-button or glossary-button (see examples in the file *advanced_button_demo* within the *Guide* library). Alternatively the 'run' construction can be used to make a *Guide* button start a shell-script that invokes a picture-displaying utility such as *xloadimage*; in this case the picture will come up in a separate window.

**FONTS**

*Guide* needs three different fonts: *guidefont* is the normal font used for text; *buttonfont* is the font used for replace-buttons; *ghostfont* is the font used for ghosts. *Guide* runs on a lot of different machines and window systems, and each tends to represent fonts in a different way. To cater for this, the font files used by *Guide* have various prefixes and suffixes according to the machine and window system in use. Sometimes the user may want to use different fonts from the standard ones supplied with *Guide*. This can be done by embedding context-tables within individual source files, as described in the User Manual. Users might, however, want to make more lasting changes. If so, the way to achieve this is either to use the environment variables GUIDEFONT, GUIDEBUTTONFONT and GUIDEGHOSTFONT or, in the X11 version, the X defaults *guide*font*, *guide*buttonfont* and *guide*ghostfont*. With SunView *Guide*, GUIDEFONT, GUIDEBUTTONFONT and GUIDEGHOSTFONT can be set to filenames which contain the desired fonts. In X11 *Guide*, GUIDEFONT, GUIDEBUTTONFONT, GUIDEGHOSTFONT and the X defaults *guide*font*, *guide*buttonfont* and *guide*ghostfont* should set to a font name rather than a file name.

SunView *Guide* first looks at the environment variables GUIDEFONT, GUIDEBUTTONFONT and GUIDEGHOSTFONT. If any of these variables is not set, then SunView *Guide* uses the corresponding font in the */etc/fonts* sub-directory of the *Guide* library.

X11 *Guide* also looks at the environment variables first. If one is not set, then X11 *Guide* will look at the corresponding X default: *guide*font*, *guide*buttonfont* or *guide*ghostfont*. If both the environment variable and the X default are not set for a given font, then X11 *Guide* will look in */etc/guide_defaults* within the *Guide* library; the format of this file should be self-explanatory.

*Guide* uses its default fonts to create some subsidiary ‘fonts’. For example glossary-buttons are represented in *guidefont*, but are underlined, and menu-buttons are represented in *buttonfont* though in reality they are somewhat different from replace-buttons. The user can change these subsidiary fonts by setting the environment variables GUIDEGLOSSARYFONT, GUIDEMENUBUTTONFONT, GUIDEACTIONFONT (for action-buttons) and GUIDEDEFAUTHORFONT (for definition-buttons in author-display).

Any of the environment variables representing fonts can be set to specify underlining. This is done by prefixing underline characters, e.g. "_ *fontname*" means use *fontname* but underline it, and "__ *fontname*" means the same but underline twice (top and bottom). As a further example, "__" means use the font that *Guide* would normally use but underline it twice. If you explicitly want *no* underlining even if underlining is the default (as for action-buttons, say) you use "!_".

In the experimental *curses* implementation of *Guide* the values of the environment variables should be set to built-in *curses* ‘fonts’ such as BOLD, DIM, REVERSE, etc.

**ENVIRONMENT VARIABLES**

There are two classes of environment variable: (a) those that help the user control *Guide*, and (b) those that *Guide* sets internally so that shell-scripts initiated from within *Guide* can find out about *Guide*’s state. Those in class (a) are:

GUIDEDULL — If this is defined, *Guide* suppresses its messages that warn the user that text may be displayed in non-standard colours because the colours have run out.

GUIDEFASTFIND — If this is defined, *Guide* suppresses its animation of Find commands, i.e. it proceeds to the destination without animating the path to getting there.

GUIDEFINDTRACE — If this is defined, *Guide* gives a warning before every Find program-command is executed. It can be useful in unravelling (poorly-designed?) documents that keep jumping about. When GUIDEFUNCTIONKEYS is set, the first function key selects the first item in the menu, the second function key selects the second menu item, and so on. Normally only visible items in the menu count, i.e. items with null names do not; however if an invisible item’s name begins with a ghost containing as the first 4 characters ‘Fkey’, then the item does count. As a result it is possible for *Guide* to have a totally invisible menu, controlled by function keys. As a secondary option, GUIDEFIRSTFUNCTIONKEY can be set to give the function key to be taken as the first one — normally this is the F1 key. For example if GUIDEFUNCTIONKEYS is 3 and

GUIDEFIRSTFUNCTIONKEY is 5, then F5, F6 and F7 can be used to select the first three menu items.

GUIDEFONT, GUIDEBUTTONFONT, etc.
These are the fonts that *Guide* uses (see '**FONTS'** above).

GUIDEFUNCTIONKEYS
The *Guide* menu can be selected by function keys as well as by point-and-select. In the former case, GUIDEFUNCTIONKEYS must be set to say how many function keys are allowed (the default being none).

GUIDEHELPROFF
If this is defined, *Guide* saves in a special format that makes it easier for the saved file to be processed by *troff*. (One facet of this is that artificial line breaks are avoided: lines are made as long as they need to be, and the value of GUIDESAVELENGTH is ignored.) The special format is totally compatible with the normal saving format.

GUIDELEADING
By default, *Guide* sets the 'leading' (i.e. the space between lines) using an algorithm that gives a larger leading for wider windows. This can be overridden by placing a fixed integer value in GUIDELEADING; this value (in pixels) is then used as the leading.

GUIDELEFTMARGIN
is the number of pixels that left margins are indented. *Guide* sets the right margin so that it roughly matches the left margin, and thus that text is centred. If the window is too narrow the full margins will not be used.

GUIDELIB
If the *Guide* library is not in */usr/local/lib/guide*, then GUIDELIB can be set to say where it is. If GUIDELIB is set to null, this means there is no *Guide* library; *Guide* will suppress error messages about the non-existence of its library. Many applications can be successfully run without the presence of a library.

GUIDELOAD
This is a shell-script for converting name of file to be loaded.

GUIDEOUTPUT
(see above description of non-interactive use.)

GUIDEPAGES
This overwrites MANPATH or */usr/man* as the place to search for manual pages; it should be a colon-separated list of directory names.

GUIDEPICVERSION
At release 5 (April 1994) *Guide* changed the format of pictures saved in source files, though the old format is still accepted. The new format is 7-bit clean, and therefore allows much freer use of files. *Guide* by default saves pictures in the new format (which is not accepted by earlier *Guide* releases) unless the original source file contained pictures that were all in the old format. (In the latter case it is assumed that the author wishes, for compatibility reasons, to continue to use the old format.) GUIDEPICVERSION overrides the default behaviour: a value of 1 causes all files to be saved in the old format, so that all releases of *Guide* can read them; a value of 2 causes all files to be saved in the new format.

GUIDESAVELENGTH
This is the (approximate) length of line to be used when a *Guide* source file is saved; the default value is 80. Beware giving GUIDESAVELENGTH a big value: some UNIX utilities (e.g. spelling check) fail if they are given very long lines.

GUIDESCRIPTHOST
Assume that a hyperdocument that is stored in directory D on remote host H contains a button that launches a shell-script by using the extension:

*run guide helpfile&*

If this file is used remotely, the shell-script will be run on the local host. Thus, if nothing special were done, it would try to find *helpfile* on the local host rather than on host H. *Guide* has a facility to get around this. When it starts a shell-script within a file that originally came from a remote host, it sets the environment variable GUIDESCRIPTHOST to the remote host and directory (i.e. to H:D in our example). If, when *Guide* starts, GUIDESCRIPTHOST is set, then *Guide* prefixes the value of GUIDESCRIPTHOST to the name of every file it loads (unless there is already a hostname specified on the filename, or, for a directory specified by GUIDESCRIPTHOST, unless the pathname is an absolute one). Thus in our example, *helpfile* would automatically be sought from directory D on host H, which was what was intended. The purpose of the facility is therefore to allow existing hyperdocuments to be able to be used remotely *without the author having to make any changes*.

The author can, if desired, also set GUIDESCRIPTHOST explicitly, or even set it to null in order to override the default setting when files *should* come from the local host.

GUIDETABWIDTH — By default *Guide* sets tab stops every 8 spaces across the screen; GUIDETABWIDTH can be used to override this default. Note that *Guide* puts out a minimum of two spaces on a tab.

Note that this facility will only work with filenames that are processed by *Guide* itself when launched from within the shell-script. If the shell-script executed *cat xxx* than *xxx* would be sought on the local host where the shell-script was running.

GUIDETRANSCRIPT — The file to which transcripts are to be written.

GUIDEVTABS — This gives the positions for vertical tabs; a sample value is "7 10 16". The numbers refer to lines within the current frame-of-view, where the first line is line 0. If GUIDEVTABS is not set, the first vertical tab position is put at line 10. Any other vertical tab position that is not set is put 10 lines beyond the previous one. A vertical tab goes down to the next tab position (or to the bottom of the current frame-of-view, if this comes first — therefore a huge number as a tab position causes vertical tabs to behave like form-feeds); if it is already beyond the current position, a single newline is output.

KSTMONO — If this is defined, the effect is the same as setting the −mono option.

The environment variables in class (b) are:

GUIDEACTUALLIB — gives the full pathname to the *Guide* library. (This will be derived from GUIDELIB, if it exists, or otherwise from the default library.)

GUIDEBARLINE — gives the line number of the mark-bar (where the top line of the view is line 0, the second line 1, etc.); the value −1 means the marker-bar is not placed.

GUIDEBARVIEW — gives the view in which the marker-bar is placed; has value −1 if the marker-bar is not placed.

GUIDEBUTTONVIEW — gives the view of the currently selected button (which can be useful if the button launches a shell-script).

GUIDECWD — gives the full pathname to the directory from which *Guide* was started. *Guide* only sets this if it is not already set; this means that if a shell-script within *Guide* starts another *Guide* then the child *Guide* will inherit the same GUIDECWD. It also means that the user can set GUIDECWD before *Guide* is called.

GUIDEFILE — is set before *GUIDE* calls GUIDELOAD.

GUIDEMOUSEBUTTONS — gives the mouse-button used to select the current button: 0 means none (e.g. the button was automatically selected); 1 means left-hand mouse-button; 2 means middle; 3 means right. (See −P for allowing alternative use of mouse-buttons.)

GUIDENAME — This is the name of the UNIX command used to start *Guide*; usually it will be 'guide' or 'guide_reader'. If a *Guide* button initiates another *Guide* session it is valuable to use GUIDENAME, e.g. 'run $GUIDENAME a-document&'. *Guide* initialises GUIDENAME only if it is not already set. (If you run a private version of *Guide* that is not in a directory on your execution path, you should set GUIDENAME to the full pathname of your private file.)

GUIDEPID — is *Guide*'s process ID. It is available to shell-scripts called from within *Guide*, and is useful for creating unique filenames.

GUIDEPIPEPID — has the same value as GUIDEPID, except that, if a shell-script is initiated in the background, it is set to a null value.

GUIDEPLANES — gives the number of colour planes available to *Guide*.

GUIDETAB — is the tab character. It is useful within extensions to buttons, because *Guide* uses a tab to terminate each extension. If a tab is required within an extension it can be specified by '@{GUIDETAB}'. (Alternatively the extension can be enclosed in '|' characters, and a real tab can be used.)

GUIDEVIEWCOUNT — is the count of views on the screen.

**ENVIRONMENT VARIABLES FOR SAVING TEXTUAL FILES**

The Guide User Manual explains that, when a file is saved in text form (i.e. with the Guide structure stripped out), various strings can be inserted in place of the structural elements that are lost. These strings are controlled by environment variables whose names are related to the names used as mark-up in a Guide source file (e.g. *Bu* at the start of a button and *bU* at the end, and *En* at the start of an enquiry and *eN* at the end). Thus the environment variable GUIDE_Bu gives the string to be output where a button started. By default this string is null but

```
setenv GUIDE_Bu "\fB"
```

would cause '\fB' to be placed on the front of each string that was a button-name. Similarly GUIDE_bU could be set to put a string on the end of each former button-name.

In addition to the set of environment variables based on *Guide* request names, there are the further variables that insert strings into saved text files:

GUIDE_Nu — gives the string to replace the 'null marker' that appears on the *Guide* screen when, for example, the replacement is null.

GUIDE_La — gives the string to replace the left-arrow characters that surround the names of replaced buttons when *Guide*'s −r option is used.

**X DEFAULTS**

As well as reading the standard X defaults the X11 implementation of *Guide* also recognizes the X defaults '*guide*font*', '*guide*buttonfont*' and '*guide*ghostfont*'; see '**FONTS**' above.

**FILES**

(It is here assumed that the *Guide* library is */usr/local/lib/guide*.)

sh — shell used by *Guide* when it runs shell-scripts. *Guide* uses the *sh* that is found using the current path; thus the normal Bourne shell can be overridden (e.g. by a restrictive shell).

context_library — the default *context-library*. Usually this is found in (the *etc* sub-directory of) the *Guide* library, but it can be augmented by

| | |
|---|---|
| | a *context_library* file in a current directory. |
| glossary.guide | the default definitions-file. Usually this is taken from the *Guide* library, but it can be augmented by a *glossary.guide* file in a current directory. |
| /usr/local/lib/guide/etc/guide_defaults | list of font names for X11 *Guide* to attempt to open on startup. |
| /usr/local/lib/guide/README | summary of the contents of */usr/local/lib/guide* and its sub-directories. |
| /usr/local/lib/guide | recommended directory for public source files. In particular this may contain demonstration files, including files for teaching how to use *Guide* (*learn*), and the file *local.guide*, which describes any characteristics of the current implementation that are not covered by this manual page. The file *help* gives a guide to the content of the library. |
| /usr/local/lib/guide/man/cache* | directories containing cache files created by the standard preprocessor. |
| /usr/local/lib/guide/etc/moption.script | shell-script used by the standard preprocessor to convert manual pages to *Guide* format. |
| /usr/local/lib/guide/etc/moption.macros | *nroff* macros used by the standard preprocessor to convert manual pages. |
| /usr/local/lib/guide/etc/fonts/*font* | fonts used by SunView *Guide*. |

**SEE ALSO**

callguide(1), man(1), spell(1).

P.J. Brown, *Guide user manual*, which is available from the Software Tools Centre, Computing Laboratory, The University, Canterbury, Kent CT2 7NF, UK.

P.J. Brown, *Using Guide as a help-system*.

P.J. Brown, *The format of Guide source files* (also available, in Guide format, within the *doc* subdirectory of the *Guide* library).

**BUGS**

The thumb-bar sometimes appears to behave oddly; this is because *Guide* deals with dynamic documents that may be continually changing in size.

When the preprocessor converts a manual page it uses *nroff*, which has a maximum output line length of a few hundred characters. When *nroff* reaches its limit it throws in a newline; a consequence is that long converted paragraphs from manual pages have a few extra newlines in them.

When the preprocessor finds a file that is represented by a symbolic link, then *Guide* may know the file under the destination name. This may confuse the user when he tries to save the file: *Guide* will present him with a default filename that is different from the name of the file loaded (though in reality both names represent the same file).